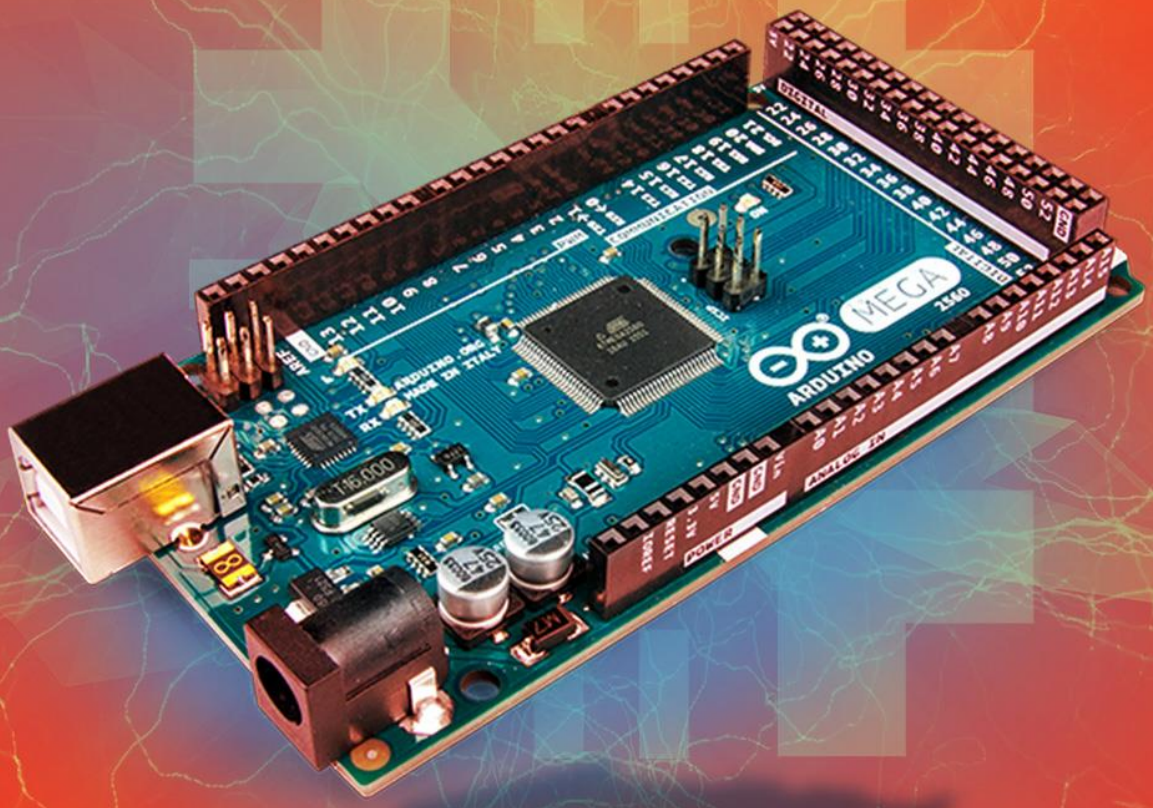
ELETROGATE

APOSTILA KIT

ARDUINO ADVANCED



V1.5 – Outubro/2024

Olá, Maker!

Parabéns pelo primeiro passo rumo a uma jornada incrível!

Esta apostila foi desenvolvida especialmente para você, servindo como um guia teórico e prático para o Kit Arduino Advanced. O kit possui todos os componentes necessários para os projetos propostos e, assim, permitirá que você tenha uma experiência completa.

Se você ainda é novo no universo Arduino, não se preocupe! Vamos te guiar desde o começo, com uma introdução à plataforma, instalação, configuração e o passo a passo para você dominar todos os recursos da sua placa Arduino. E o melhor: com projetos práticos que vão te levar da programação à montagem, até ver o seu projeto ganhar vida!

Mas as oportunidades não param por aí. Com o Kit Arduino Advanced, as possibilidades são infinitas. Você poderá criar inúmeros projetos utilizando sensores, módulos, e muito mais. Isso e muito mais você encontrará no site da Eletrogate.

Então, se você ainda não garantiu o seu, não perca mais tempo! Acesse agora o nosso site e garanta já o seu Kit Arduino Advanced para desbloquear todo o seu potencial criativo! Tudo o que você precisa para soltar sua criatividade, aprender e se divertir você encontrará neste Kit.

Prepare-se para uma aventura repleta de aprendizado com conteúdo de qualidade e projetos práticos. Estamos ansiosos para ver tudo o que você vai criar.

Aproveite bastante e bons estudos!

Sumário

Parte I – Conceitos elétricos, introdução aos componentes e instalação da IDE

Introdução

A primeira parte da apostila faz uma revisão sobre circuitos elétricos, com destaque para questões práticas de montagem e segurança que surgem no dia a dia do usuário do Kit Advanced para Arduino.

O conteúdo de circuitos elétricos aborda divisores de tensão e corrente, conversão de níveis lógicos, grandezas analógicas e digitais, níveis de tensão e cuidados práticos de montagem.
Em relação aos componentes básicos, é feita uma breve discussão sobre resistores elétricos, capacitores, leds, diodos, chaves e protoboards.

O Arduino Mega é o principal componente do Kit e é discutido e introduzido em uma seção à parte, na Parte II da apostila.

Todos os conteúdos da Parte I são focados na apostila Arduino Advanced, tendo em vista a utilização adequada dos componentes e da realização prática de montagens pelos usuários. No entanto, recomenda-se a leitura das referências indicadas ao final de cada seção para maior aprofundamento.

O leitor veterano, já acostumado e conhecedor dos conceitos essenciais de eletrônica e eletricidade, pode pular a Parte I e ir direto a Parte III, na qual são apresentadas uma seção de exemplo de montagem para cada sensor ou componente importante da apostila.

Preparado? Vamos começar!

Revisão de circuitos elétricos

A apostila Advanced, bem como todas as outras apostilas que tratam de Arduino e eletrônica em geral, tem como conhecimento de base as teorias de circuitos elétricos e de eletrônica analógica e digital.

Do ponto de vista da teoria de circuitos elétricos, é importante conhecer os conceitos de grandezas elétricas: Tensão, corrente, carga, energia potência elétrica. Em todos os textos sobre Arduino ou qualquer assunto que envolva eletrônica, você sempre terá que lidar com esses termos. Para o leitor que se inicia nessa seara, recomendamos desde já que mesmo que a eletrônica não seja sua área de formação, que conheça esses conceitos básicos.

Vamos começar pela definição de "circuito elétrico". ***Um circuito elétrico/eletrônico é uma interconexão de elementos elétricos/eletrônicos***. Essa interconexão pode ser feita para atender a uma determinada tarefa, como acender uma lâmpada, acionar um motor, dissipar calor em uma resistência e tantos outras.

O circuito pode estar **energizado** ou **desenergizado**. Quando está energizado, é quando uma fonte de tensão externa ou interna está ligada aos componentes do circuito. Nesse caso, uma corrente elétrica fluirá entre os condutores do circuito. Quando está desenergizado, a fonte de tensão não está conectada e não há corrente elétrica fluindo entre os condutores.

Mas atenção, alguns elementos básicos de circuitos, como os capacitores ou massas metálicas, são elementos que armazenam energia elétrica. Em alguns casos, mesmo não havendo fonte de tensão conectada a um circuito, pode ser que um elemento que tenha energia armazenada descarregue essa energia dando origem a uma corrente elétrica transitória no circuito. Evitar que elementos do circuito fiquem energizados mesmo sem uma fonte de tensão, o que pode provocar descargas elétricas posteriores (e em alguns casos, danificar o circuito ou causar choques elétricos) é um dos motivos dos sistemas de aterramento em equipamentos como osciloscópios e em instalações residenciais, por exemplo.

Em todo circuito você vai ouvir falar das grandezas elétricas principais, assim, vamos aprender o que é cada uma delas.

Carga e corrente elétrica

A grandeza mais básica nos circuitos elétricos é a carga elétrica. Carga é a propriedade elétrica das partículas atômicas que compõem a matéria (prótons, nêutrons e elétrons), e é medida em Coulombs.

Do conceito de carga elétrica obtemos o **conceito de corrente elétrica, que nada mais é do que a taxa de variação da carga ao longo do tempo**, ou seja, quando você tem um fluxo de carga em um condutor, a quantidade de carga (Coulomb) que atravessa esse condutor por unidade de tempo, é chamada de corrente elétrica. A medida utilizada para corrente é o **Ampére(A).**

Aqui temos que fazer uma distinção importante. Existem corrente elétrica contínua e alternada:

- **Corrente elétrica contínua (VCC/VDC):** É uma corrente que permanece constante e em uma única direção durante todo o tempo.
- **Corrente elétrica alternada (VAC/VCA)**: É uma corrente que varia de forma senoidal ao longo do tempo.

Com o Arduino Mega e na maioria dos componentes eletrônicos, lidamos com a corrente elétrica contínua. É diferente da corrente e tensão elétrica da tomada de sua casa, que são alternadas.

Outro conceito importante ao falarmos de corrente elétrica é o sentido do fluxo. Corrente elétrica é o fluxo de carga elétrica através de um condutor, e é transportada por elétrons em um circuito. Convencionalmente, o sentido da corrente elétrica foi definido como o fluxo de cargas do lado positivo para o lado negativo. Essa convenção foi estabelecida antes da descoberta dos elétrons e simplifica a análise de circuitos.

Na realidade, os elétrons, que possuem carga negativa, se movem do polo negativo para o polo positivo. Isso ocorre porque, em um circuito, o excesso de elétrons em um lado (polo negativo) cria uma região negativa em relação ao lado com menos elétrons (por conter menos elétrons, dizemos que essa região está positiva). Essa diferença de potencial cria uma força que faz com que os elétrons se movam em direção ao lado positivo.

Quando conectamos os dois polos por meio de uma carga (como um LED, resistor, motor, por exemplo), a corrente elétrica é gerada e flui devido a essa diferença de

potencial (conceito que será explicado mais adiante). A corrente é a forma como a energia é transportada e utilizada pelos componentes do circuito.

Tensão elétrica

Para que haja corrente elétrica em um condutor, é preciso que os elétrons se movimentem por ele em uma determinada direção, ou seja, é necessário "alguém" para transferir energia para as cargas elétricas para movê-las. Isso é feito por uma força chamada **força eletromotriz (fem.),** tipicamente representada por uma bateria. Outros dois nomes comuns para força eletromotriz são **tensão elétrica** e **diferença de potencial.**

O mais comum é você ver apenas "*tensão*" nos artigos e exemplos com Arduino. Assim, definindo formalmente o conceito: Tensão elétrica é a energia necessária para mover uma unidade de carga através de um condutor, e é medida em **Volts (V).**

Potência e energia

A tensão e a corrente elétrica são duas grandezas básicas, e juntamente com a potência e energia, são as grandezas que descrevem qualquer circuito elétrico ou eletrônico. A potência é definida como a variação de energia (que pode estar sendo liberada ou consumida) em função do tempo, e é medida em **Watts (W).** A potência está associada ao calor que um componente está dissipando e a energia que ele consume.

Nós sabemos da vida prática que uma lâmpada de 100W consome mais energia do que uma de 60 W. Ou seja, se ambas estiverem ligadas por 1 hora por exemplo, a lâmpada de 100W vai implicar numa conta de energia mais cara.
A potência se relaciona com a tensão e corrente pela seguinte fórmula:

$$P = V \times I$$

Essa é a chamada potência instantânea. Com essa fórmula, para saber qual a potência dissipada em um resistor, por exemplo, basta informar a tensão aplicada nos terminais do resistor e a corrente que passa por ele. O conceito de potência é importante pois muitos hobbistas acabam não tendo noção de quais valores de resistência usar, ou mesmo saber especificar componentes de forma adequada.

Um resistor de 33 ohms de potência igual a 1/4W, por exemplo, não pode ser ligado diretamente em circuito de 5V, pois nesse caso a potência dissipada nele seria maior que a que ele pode suportar.

Vamos voltar a esse assunto em breve, por ora, tenha em mente que é importante ter uma noção da potência dissipada ou consumida pelos elementos do circuito que você irá montar.

Por fim, a energia elétrica é o somatório da potência elétrica durante todo o tempo em que o circuito esteve em funcionamento. A energia é dada em **Joules (J)** ou **Wh (watt-hora).** A unidade Wh é interessante pois mostra que a energia é calculada multiplicando-se a potência pelo tempo (apenas para os casos em que a potência é constante).

Essa primeira parte é um pouco conceitual, mas é importante saber de onde vieram todas as siglas que você irá encontrar nos manuais e artigos na internet. Na próxima seção, vamos discutir os componentes básicos que compõem o Kit Advanced para Arduino.

Conversão de níveis lógicos e alimentação correta de circuitos

É muito comum que hobbistas e projetistas em geral acabem cometendo alguns erros de vez em quando. Na verdade, mesmo alguns artigos na internet e montagens amplamente usadas muitas vezes acabam por não utilizar as melhores práticas de forma rigorosa. Isso acontece frequentemente com situações em que os níveis lógicos dos sinais usados para integrar o Arduino com outros circuitos não são compatíveis entre si.

Como veremos na seção de apresentação do Arduino Mega, ele é alimentado por um cabo USB ou uma fonte externa entre 7V e 12V. A placa do Arduino possui reguladores de tensão que convertem a alimentação de entrada para 5V e para 3,3V. Os sinais lógicos enviados pelas portas de saída digitais do Arduino operam com sinais de 0V ou 5V.

Isso significa que quando você quiser usar o seu Arduino Mega com um sensor ou CI que trabalhe com 3.3V, é preciso fazer a adequação dos níveis de tensão, pois se você enviar um sinal de 5V (saída do Arduino) em um circuito de 3.3V (CI ou sensor), você poderá queimar o pino daquele componente.
Em geral, sempre que dois circuitos que trabalhem com níveis de tensão diferentes forem conectados, é preciso fazer a conversão dos níveis lógicos. O mais comum é ter que abaixar saídas de 5V para 3.3V. Subir os sinais de 3.3V para 5V na maioria das vezes

não é necessário pois o Arduino entende 3.3V como nível lógico alto, isto é, equivalente a 5V.

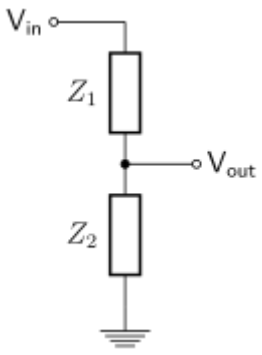
Para fazer a conversão de níveis lógicos você tem duas opções:

- Usar um divisor de tensão;
- Usar um **CI conversor de níveis lógicos**;

O divisor de tensão é a solução mais simples, mas usar um CI conversor é mais elegante e é o ideal. O divisor de tensão consiste em dois resistores ligados em série (Z1 e Z2), em que o sinal de 5V é aplicado em um dos terminais de Z1. O segundo terminal de Z2 é ligado ao GND, e o ponto de conexão entre os dois resistores é a saída do divisor, cuja tensão é dada pela seguinte relação:

$$V_{\text{out}} = \frac{Z_2}{Z_1 + Z_2} \cdot V_{\text{in}}$$

Nessa equação, e Z1 e Z2 são os valores dos resistores da figura abaixo.



Um divisor de tensão muito comum é fazer Z1 igual 1KΩ e Z2 igual 2KΩ. Dessa forma a saída Vout fica sendo 3.33V. Como o Kit Advanced para Arduino não contém resistores de 2K, você pode associar dois resistores de 1K ligados em série para formar o resistor Z2. Fazendo isso, teremos Z1 = 1KΩ e Z2 = (1KΩ + 1KΩ), resultando numa saída de 3.33V segundo a fórmula mostrada (Vout = (1KΩ + 1KΩ)/(1KΩ+(1KΩ + 1KΩ)) x 5).

Vamos exemplificar como fazer um divisor de tensão como esse na seção de exemplos da parte II da apostila.

Revisão de componentes eletrônicos

O Kit Advanced para Arduino possui os seguintes componentes básicos para montagens de circuitos:

- Buzzer Ativo 5V;
- LED Vermelho/ Verde/ Amarelo;
- Resistor 330Ω/ 1KΩ/ 10KΩ;
- Diodo 1N4007;
- Potenciômetro 10KΩ;
- Capacitor Cerâmico 10 nF/ 100 nF;
- Capacitor Eletrolítico 10uF/ 100uF;
- Chave Táctil (Push-Button);
- Transistor NPN BC548;
- Transistor PNP BC558;
- Display 7 segmentos.

Vamos revisar a função de cada um deles dentro de um circuito eletrônico e apresentar mais algumas equações fundamentais para ter em mente ao fazer suas montagens.

Resistores elétricos

Os resistores são componentes que se opõem à passagem de corrente elétrica, ou seja, oferecem uma resistência elétrica. Dessa forma, quanto maior for o valor de um resistor, menor será a corrente elétrica que fluirá por ele e pelo condutor a ele conectada. A unidade de resistência elétrica é o **Ohm (Ω)**, também simbolizado pela letra R maiúscula, que é a unidade usada para especificar o valor dos resistores.

Os resistores mais comuns do mercado são construídos com fio de carbono e são vendidos em várias especificações. Os resistores do Kit são os tradicionais de 1/4W e 5% de tolerância. Isso significa que eles podem dissipar no máximo 1/4W (0,25 watts) e seu valor de resistência pode variar em até 5% para mais ou para menos, ou seja, o resistor de 1KΩ pode então ter um valor mínimo de 950Ω e um valor máximo de 1050Ω para ser considerado em condições de uso.

Em algumas aplicações você pode precisar de resistores com precisão maior, como 1%. Também há casos em que a precisão não é tão importante, podendo ser utilizados resistores com tolerância de 10%. Em alguns casos, pode ser necessário usar resistores

com maior potência, como 1W, enquanto em outros a potência pode ser menor, como 1/8W. Essas variações dependem da natureza específica de cada circuito."

Em geral, para as aplicações típicas e montagens de prototipagem que podem ser feitos com o Kit Advanced para Arduino, os resistores tradicionais de 1/4W e 5% de tolerância são mais que suficientes.

Outro ponto importante de se mencionar aqui é a Lei de Ohm, que relaciona as grandezas de tensão, corrente e resistência elétrica. A lei é dada por:

$$V = R \times I$$

Ou seja, se você sabe o valor de um resistor e a tensão aplicada em seus terminais, você pode calcular a corrente elétrica que fluirá por ele. Juntamente com a equação para calcular potência elétrica, a lei de Ohm é importante para saber se os valores de corrente e potência que os resistores de seu circuito estão operando estão adequados.

Para fechar, você deve estar se perguntando, como saber o valor de resistência de um resistor? Você tem duas alternativas: Medir a resistência usando um multímetro ou determinar o valor por meio do código de cores do resistor.

Se você pegar um dos resistores do seu kit, verá que ele possui algumas faixas coloridas em seu corpo. Essas faixas são o código de cores do resistor. As duas primeiras faixas dizem os dois primeiros algarismos decimais. A terceira faixa colorida indica o multiplicador que devemos usar. A última faixa, que fica um pouco mais afastada, indica a tolerância.

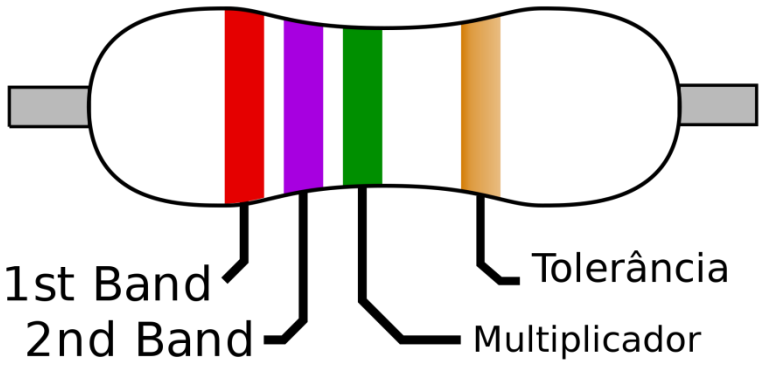


Figura 1: Faixas coloridas em um resistor

Na figura 2 apresentamos o código de cores para resistores. Cada cor está associada a um algarismo, um multiplicador e uma tolerância, conforme a tabela. Com a tabela você pode determinar a resistência de um resistor sem ter que usar o multímetro.

Mas atenção, fazer medições com o multímetro é recomendado, principalmente se o componente já tiver sido utilizado, pois pode ter sofrido algum dano ou mudança que não esteja visível.

Cor	Dígito	Multiplicador	Tolerância
Prata	-	x 0,01	± 10%
Dourado	-	x 0,1	± 5%
Preto	0	x 1	-
Marrom	1	x 10	± 1%
Vermelho	2	x 100	± 2%
Laranja	3	x 1K	-
Amarelo	4	x 10K	-
Verde	5	x 100K	± 0,5%
Azul	6	x 1M	± 0,25%
Violeta	7	x 10M	± 0,1%
Cinza	8	-	± 0,05%
Branco	9	-	-

Figura 2: Código de cores para resistores

Aplicando a tabela da figura 2 na imagem da figura 1, descobrimos que o resistor é de 2,7MΩ (Mega ohms) com tolerância de 5% (relativo à cor dourado da última faixa).

Capacitores

Os capacitores são os elementos mais comuns nos circuitos eletrônicos depois dos resistores. São elementos que armazenam energia na forma de campos elétricos. Um capacitor é constituído de dois terminais condutores e um elemento dielétrico entre esses dois terminais, de forma que quando submetido a uma diferença de potencial, um campo elétrico surge entre esses terminais, causando o acúmulo de cargas positivas no terminal negativo e cargas negativas no terminal positivo.

São usados para implementar filtros, estabilizar sinais de tensão, na construção de fontes retificadores e várias outras aplicações.

O importante que você deve saber para utilizar o Kit é que os capacitores podem ser de quatro tipos:

- Eletrolíticos;
- Cerâmicos;
- Poliéster;
- Tântalo.

Capacitor eletrolítico

As diferenças de cada tipo de capacitor são a tecnologia construtiva e o material dielétrico utilizado. Capacitores eletrolíticos são feitos de duas longas camadas de alumínio (terminais) separadas por uma camada de óxido de alumínio (dielétrico). Devido a sua construção, eles possuem **polaridade,** o que significa que você obrigatoriamente deve ligar o terminal positivo (o maior) no polo positivo da fonte de alimentação, e o terminal negativo (marcado no capacitor por uma faixa com símbolos de "-") obrigatoriamente no polo negativo da fonte. Do contrário, o capacitor será danificado.

Capacitores eletrolíticos costumam ser da ordem de micro Farad, sendo o **Farad** a unidade de medida de capacitância, usada diferenciar um capacitor do outro. A Figura 3 ilustra um típico capacitor eletrolítico.

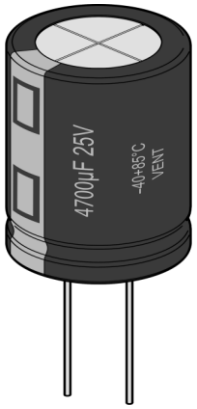


Figura 3: Capacitor eletrolítico 4700 micro Farads / 25 V

Capacitores cerâmicos

Capacitores cerâmicos não possuem polaridade, e são caracterizados por seu tamanho reduzido e por sua cor característica, um marrom claro um tanto fosco. Possuem capacitância da ordem de **pico Farad**. Veja nas imagens abaixo um típico capacitor cerâmico e sua identificação:

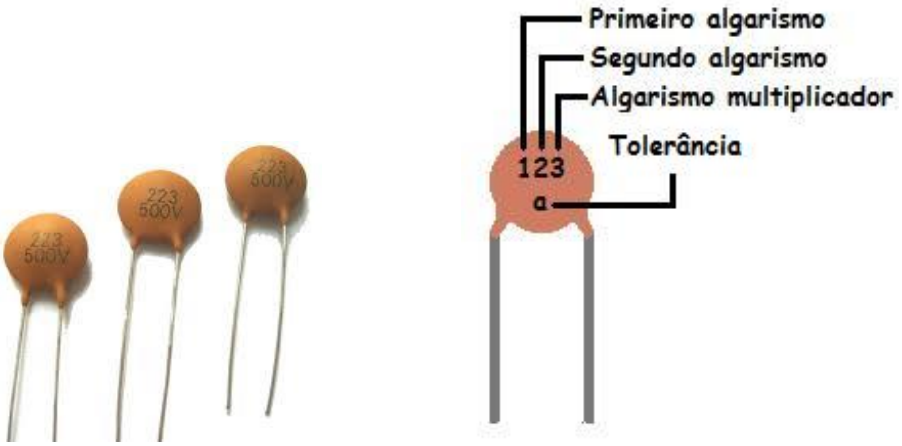


Figura 4: Capacitores cerâmicos

Na imagem da esquerda, os capacitores possuem o valor de **22 nano Farads** para a faixa de tensão de até 500V.

223 = 22 x 1000 = 22.000 pF = 22 nF

Por fim, há também os capacitores de poliéster e de tântalo. No Kit Advanced para Arduino, você receberá apenas exemplares de capacitores eletrolíticos e cerâmicos.

Diodos e LEDs

Diodos e LEDs são tratados ao mesmo tempo pois são, na verdade, o mesmo componente. Diodos são elementos semicondutores que só permitem a passagem de corrente elétrica em uma direção.

São constituídos de dois terminais, o **Anodo(+)** e o **catodo(-)**, sendo que para que possa conduzir corrente elétrica, é preciso conectar o Anodo na parte positiva do circuito, e o Catodo na parte negativa. Do contrário, o diodo se comporta como um circuito aberto, bloqueando a passagem dos elétrons.

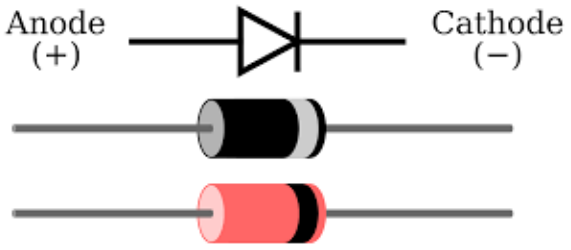


Figura 4: Diodo e seus terminais

Na figura 4, você pode ver que o componente possui uma faixa indicadora no terminal catodo, este é o polo negativo. O diodo do Kit Advanced para Arduino é um modelo tradicional e que está no mercado há muitos anos, o 1N4007.

O LED é um tipo específico de diodo - **Light Emitter Diode**, ou seja, um diodo que emite luz. Trata-se de um diodo que quando polarizado corretamente, emite luz para o ambiente externo. O Kit Advanced para Arduino vem acompanhado de LEDs nas cores vermelha, verde e amarela, as mais tradicionais.

Nesse ponto, é importante você saber que sempre deve ligar um led junto de um resistor, para que a corrente elétrica que flua pelo led não seja excessiva e acabe por queimá-lo. Além disso, lembre-se que por ser um diodo, o led só funciona se o Anodo estiver conectado ao polo positivo do sinal de tensão.

Para identificar o Anodo do Led, basta identificar o terminal mais longo do componente, como na imagem abaixo:

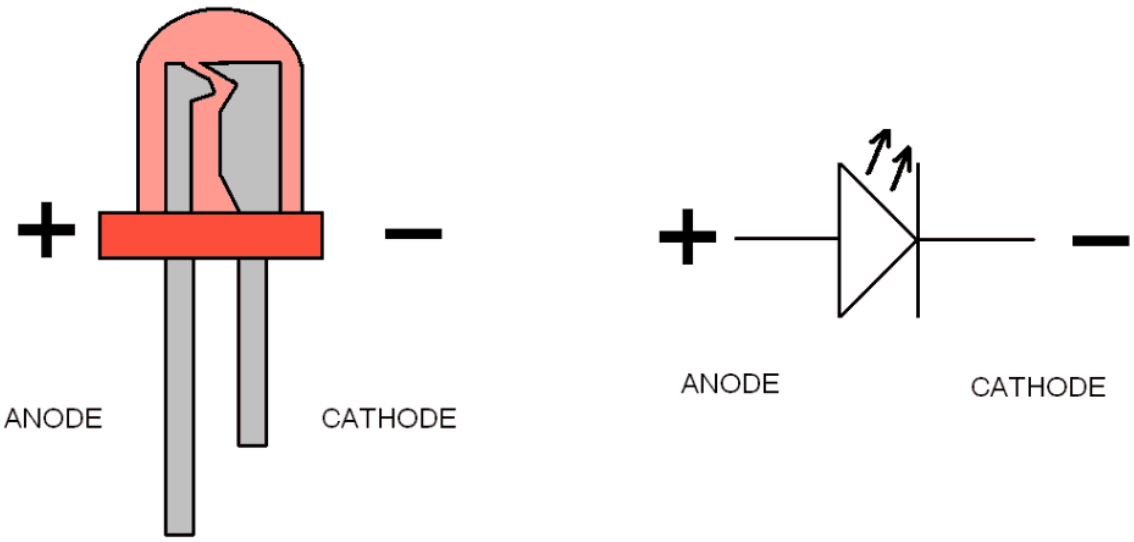


Figura 5: Terminais de um Led. Créditos: Build-eletronic-circuits.com

Transistores

Os transistores são componentes eletrônicos fundamentais na eletrônica moderna, podendo atuar como amplificadores ou interruptores. Um transistor é um dispositivo semicondutor que controla a corrente elétrica em um circuito (chamada corrente de coletor) através da aplicação de uma corrente em seu terminal de controle (chamada corrente de base).

Essa corrente é muito pequena em relação à corrente de coletor, graças a um parâmetro de cada transistor chamado ganho (também chamado Beta (β) ou h_{FE}), sendo esse o responsável por ditar em quantas vezes a corrente de base será amplificada dependendo do modo de operação.

O ganho do transistor resulta da divisão entre a corrente de coletor e a corrente de base, representada pela fórmula $\beta = \frac{Ic}{Ib}$. Isso significa que a corrente de coletor é β vezes maior que a corrente de base.
Vamos começar apresentando os tipos de transistores, que são classificados em duas categorias:

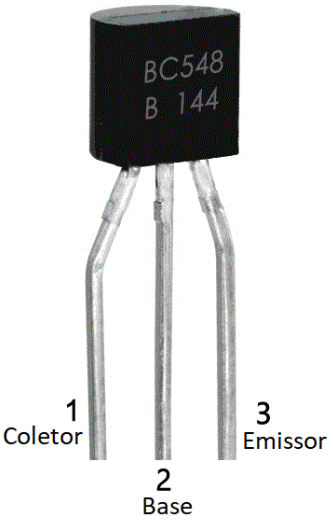
- ***Transistores de Efeito de Campo (FET)***:
Os transistores do tipo FET também se subdividem em dois grupos e possuem características diferentes dos transistores BJTs, porém não iremos abordá-los nessa apostila. São subdivididos em:

  - *Junction FET (JFET)*: Utiliza um campo elétrico para controlar a condutividade de um canal.
  - *Metal-Oxide-Semiconductor FET (MOSFET)*: Tem um isolamento de óxido que controla o fluxo de corrente em um canal semicondutor.

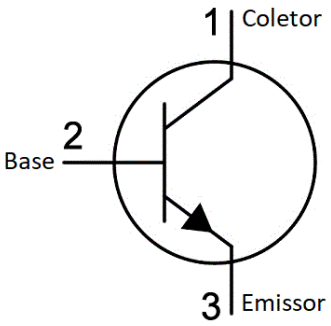
- ***Transistores Bipolares de Junção (BJT)***:
Os BJTs são o foco dessa apostila e possuem três terminais denominados **coletor, base e emissor**. Cada um desses terminais tem uma função específica que varia dependendo do tipo de transistor. São eles:

- **NPN:** No transistor NPN, a corrente principal entra pelo coletor e sai pelo emissor (considerando o sentido convencional da corrente). Para que o transistor NPN conduza, é necessário aplicar um sinal positivo na base. Esse sinal permite que a corrente flua do coletor para o emissor, habilitando a condução do transistor.

Encapsulamento TO-92

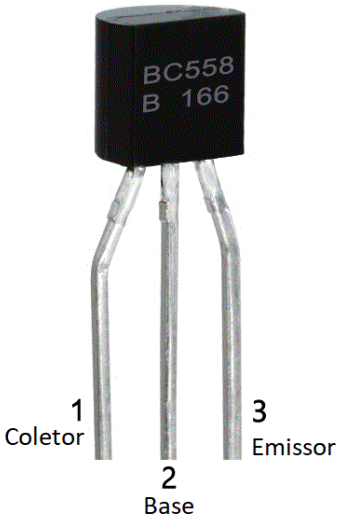


Transistor NPN

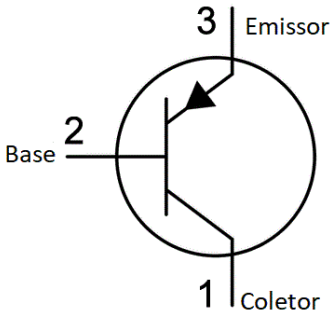


- **PNP:** O funcionamento do transistor PNP é oposto ao NPN. Nele, a corrente principal entra pelo emissor e sai pelo coletor. Entrando em condução ao aplicarmos um sinal negativo na base. Esse sinal negativo na base permite que a corrente flua do emissor para o coletor, habilitando a condução do transistor.

Encapsulamento TO-92



Transistor PNP



Modos de operação

Para facilitar a compreensão do funcionamento do transistor, vamos associar o tipo NPN a uma torneira. Nesta analogia, o coletor representa a entrada de água na torneira, o emissor representa a saída de água e, a base, equivale ao registro. Para o PNP, a analogia é a mesma, diferenciando-se apenas na função do coletor e do emissor.

Dito isso, vamos explorar os três modos de operação do transistor:

Modo Ativo: Chamamos de região ativa porque o transistor está funcionando como um amplificador. Neste estado, a corrente de base controla a corrente de coletor, mas o transistor não está completamente saturado. A corrente de coletor é proporcional à corrente de base, multiplicada pelo ganho do transistor. Isso faz com que a tensão entre o coletor e o emissor seja alta o suficiente para permitir essa amplificação. Associando à torneira, você abre o registro parcialmente, permitindo um fluxo controlado e proporcional da água. Da mesma forma, no transistor, a corrente de base controla a quantidade de corrente que flui do coletor (entrada da água) para o emissor (saída da água).

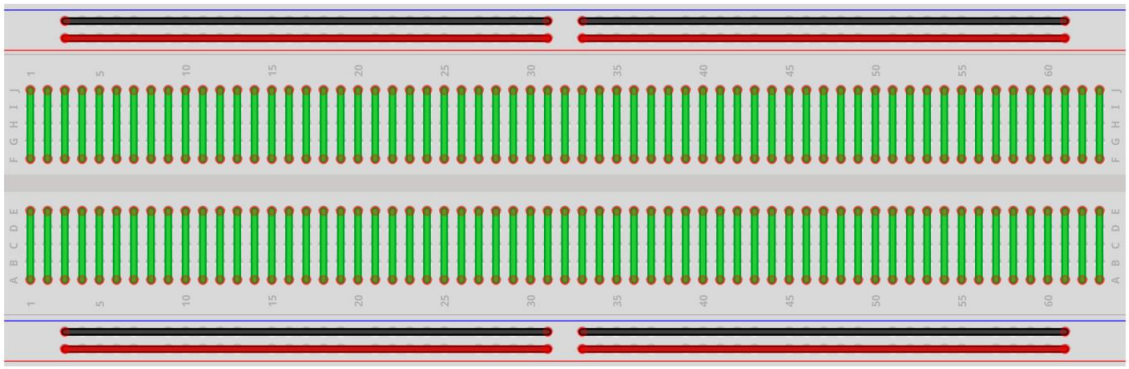
Modo Saturação: Modo saturação: Nesse modo, o transistor se comporta como um interruptor fechado (acionado). A base fica "inundada" com elétrons devido à corrente nela injetada, que é maior que a necessária para o transistor atuar na região ativa. Como resultado, a resistência entre o coletor e o emissor se torna muito baixa, assim como a tensão entre esses terminais, significando que o transistor está totalmente "ligado" e conduzindo a corrente quase sem impedimento.

Embora o ganho (β) do transistor ainda esteja presente, sua influência é menor neste modo já que existe um excesso de elétrons na base. A corrente de coletor passa a ser mais influenciada pela configuração do circuito do que pela corrente de base. Na torneira, é como se o registro estivesse completamente aberto, permitindo o fluxo máximo de água.

Modo Corte: O transistor está completamente desligado. Não há corrente fluindo entre o coletor e o emissor, e a tensão entre esses terminais é alta. O transistor atua como um interruptor aberto, interrompendo a passagem de corrente. Isso ocorre porque a corrente de base é insuficiente para ativar o transistor, e o ganho do transistor não tem efeito, pois o transistor está completamente desligado. Esse modo é equivalente à torneira completamente fechada, bloqueando totalmente o fluxo de água.

Protoboard

A protoboard é uma ferramenta essencial no desenvolvimento de circuitos eletrônicos, especialmente em projetos experimentais e de prototipagem. Sua principal vantagem é a facilidade de montagem e modificação de circuitos sem a necessidade de solda, permitindo testar e ajustar componentes rapidamente. Na imagem abaixo, mostramos as ligações internas da protoboard para que você entenda como os pinos são organizados e onde permitem contato:



Note que, nessa orientação, temos colunas numeradas de 1 a 63, enquanto as linhas são nomeadas de A a E no segmento inferior e de F a J no segmento superior. A ligação entre os pinos é bem simples de entender: em cada coluna numérica, as 5 linhas correspondentes estão interligadas entre si (linhas verdes), mas não há conexão com as colunas vizinhas (números), nem entre os segmentos superior e inferior (grupos A-E e F-J).

Já nas linhas de energia, representadas pelas linhas vermelhas e pretas, a ligação é feita no sentido horizontal, percorrendo toda a largura da protoboard. Essas linhas geralmente são usadas para fornecer alimentação positiva (VCC) e negativa (GND) aos componentes do circuito.

Os demais componentes do kit (Buzzer, potenciômetro, display de 7 segmentos e push-buttons) serão explicados em seções de exemplos, nas quais iremos apresentar um circuito prático para montagem onde será apresentado o funcionamento de cada um deles, assim como será feito para os sensores de luz (LDR) e temperatura (NTC). O Micro Servo 9g SG90 TowerPro também terá uma seção de exemplo dedicada a cada ele.

Parte II – ARDUINO MEGA 2560

Após o Arduino ter sido lançado em 2005 e ter um grande sucesso no mundo inteiro, a equipe do Arduino percebeu a necessidade de lançamento de outros modelos. O Arduino Mega foi lançado em 2010. E em relação à todos os modelos de Arduino, o Mega é o segundo mais famoso, após o Arduino Mega.

A grande diferença do Arduino Mega é que ele usa um outro Microcontrolador : **ATmega 2560**, que tem maior quantidade de memória, maior número de pinos e funções, apesar de usar o mesmo processador e o mesmo clock.

Atmel ATmega2560 datasheet:

https://www.microchip.com/wwwproducts/en/ATmega2560

Características do Microcontrolador **ATmega 2560** :

- Processador RISC com até 16 MIPS,
- 256 KBytes de memória Flash (programas),
- 8 KBytes de memória estática SRAM,
- 4 KBytes de memória não-volátil EEPROM,
- 2 Timers/Contadores de 8 bits,
- 2 Timers/Contadores de 16 bits,
- 1 Contador Real Time,
- 1 Conversor ADC de 10 bits com 16 canais,
- Quatro canais PWM de 8 bits e 12 canais PWM de 16 bits,
- Quatro interfaces seriais, uma interface I2C e uma interface SPI.
- e mais alguns outros recursos.

Características do Arduino Mega

A placa Arduino Mega 2560 foi desenvolvida para projetos mais complexos. Com 54 pinos digitais de Entrada / Saída e 16 entradas analógicas, é a placa recomendada para impressoras 3D e projetos de robótica.

Link oficial do Arduino Mega:

https://store.arduino.cc/usa/arduino-mega-2560-rev3

O Arduino Mega 2560 possui as seguintes características :

- Micro-controlador **ATmega 2560** com clock de 16 MHz,
- Regulador de 5V (AMS1117 - 1 A),
- Regulador de 3,3V (LP2985 com apenas 150 mA),

- 4 **portas seriais de hardware** :
  Serial 0 = TX0 (D1) e RX0 (D0)

  Serial 1 = TX1 (D18) e RX1 (D19)

  Serial 2 = TX2 (D16) e RX2 (D17)

  Serial 3 = TX3 (D14) e RX3 (D15)
- Uma porta **I2C** : I2C : SDA (D20) e SCL (D21)
- Uma porta **SPI**: MOSI (D51), MISO(D50), SCK(D52) e SS(D53),
- 16 portas analógicas do conversor **ADC** (A0 até A15),
- 12 portas **PWM** de 16 bits (D2 a D13),
- 32 portas Digitais multi-função,
- Um Led para TX0 e um para RX0 (interface serial 0) ,
- Um Led conectado ao pino D13.

A alimentação poderá ser feita através do conector USB ou do conector de energia (tensão recomendada para a entrada de 7 a 12V). O conector USB é protegido por um fusível de 500 mA.

O consumo de corrente através da porta USB (alimentação 5V) é de aproximadamente 75 mA (Arduino Mega rodando o programa de exemplo Blink). Cada porta digital do Arduino Mega pode suportar até 20 mA e ser usada como entrada ou como saída.

A placa tem um botão de RESET e um conector ICSP para gravação de firmware (opcional).

Observação importante : todos os pinos Digitais e Analógicos funcionam com tensões de 0 a 5V ! Não use tensões acima de 5V.

A placa tem também um conector **ICSP** conectado à interface SPI do ATmega2560. Esse conector poderá ser usado se preferir gravar seu firmware (programas) diretamente no Microcontrolador.

Comunicação USB-Serial :

A comunicação serial entre o PC e Microcontrolador **ATmega 2560** é feita através de um outro microcontrolador, o **ATmega 16U2**. De um lado vem os dados da interface USB do PC e o ATmega 16U2 transporta esses dados para a interface Serial conectada à **Serial 0** do **ATmega**

2560. A placa tem também um conector ICSP conectado ao ATmega16U2. Esse conector poderá ser usado para regravação do bootloader.

Em outros clones do Arduino Mega, podem existir outros tipos de interface USB-Serial. O driver para o PC deverá ser instalado adequadamente, dependendo do modelo dessa interface.

Esse é o diagrama esquemático do Arduino Mega 2560 (o circuito poderá variar, dependendo da versão):

Diagrama Arduino Mega2560 R3:

https://www.arduino.cc/en/uploads/Main/arduino-mega2560_R3-sch.pdf

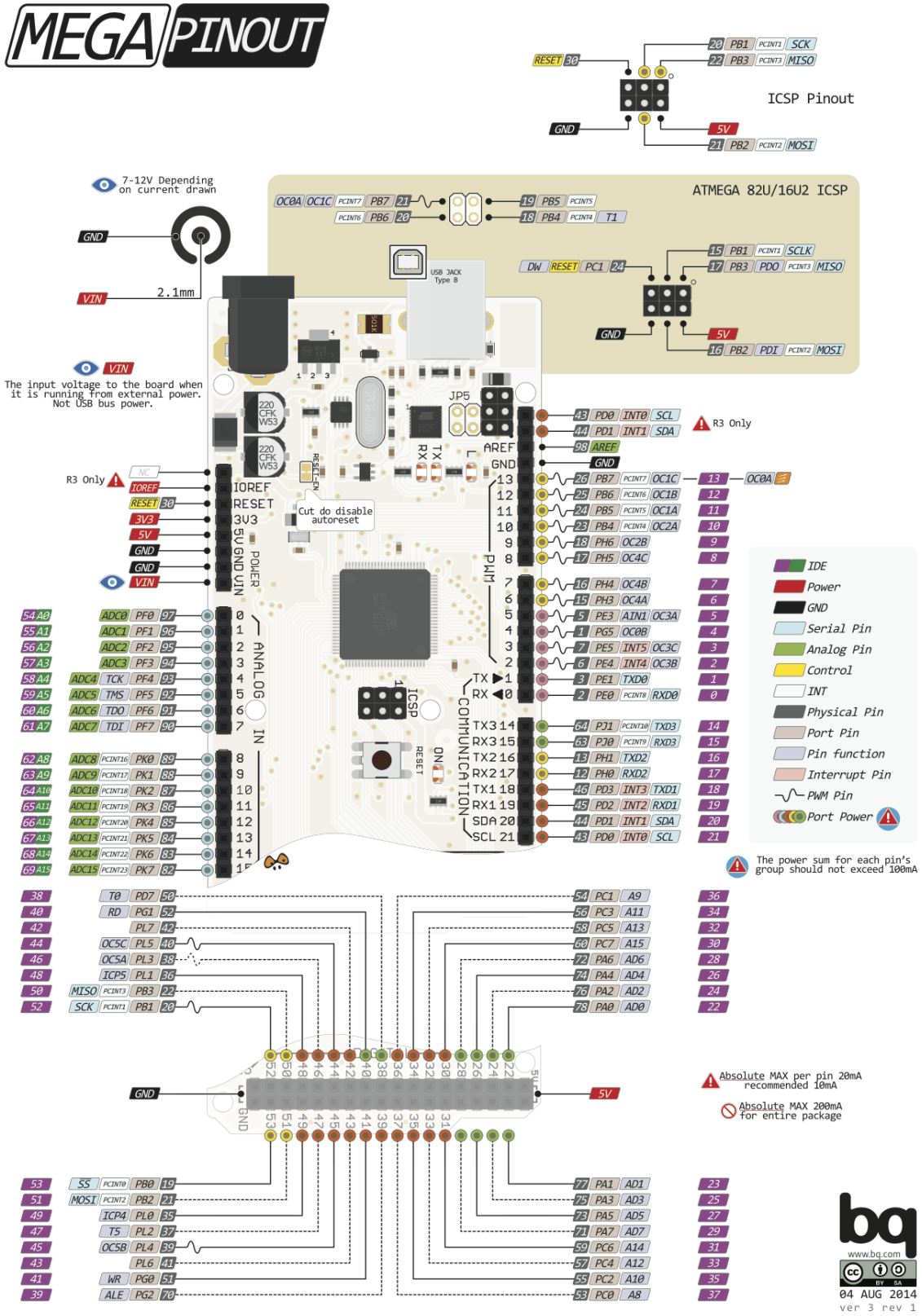
Pinout do Arduino Mega

Esse é o diagrama com todos os pinos e conexões do Arduino Mega 2560.

Para uma melhor visualização baixe o PDF :

http://blog.eletrogate.com/wp-content/uploads/2018/08/mega-v3.1-pinout.pdf

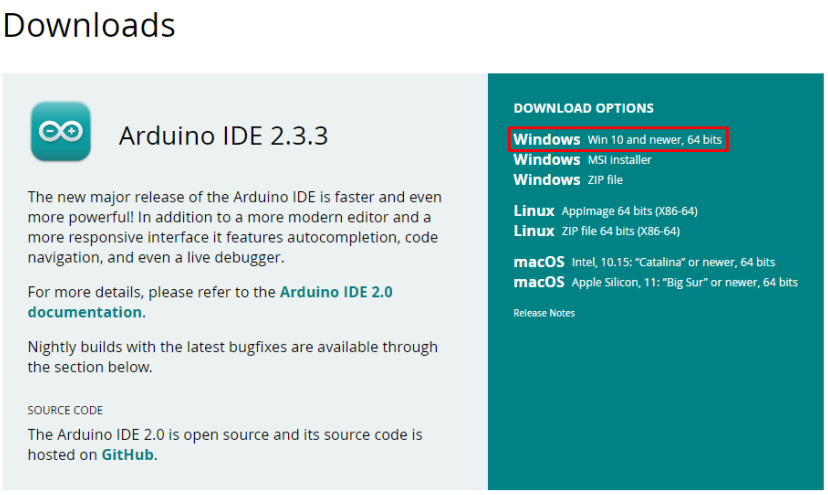
MEGA PINOUT


ICSP Pinout
ATMEGA 82U/16U2 ICSP
7-12V Depending on current drawn
GND
VIN
2.1mm
The input voltage to the board when it is running from external power. Not USB bus power.
R3 Only
IOREF
RESET
3V3
5V
POWER
ANALOG IN
PWM
COMMUNICATION
ICSP
Cut do disable autoreset
IDE
Power
GND
Serial Pin
Analog Pin
Control
INT
Physical Pin
Port Pin
Pin function
Interrupt Pin
PWM Pin
Port Power
The power sum for each pin's group should not exceed 100mA
Absolute MAX per pin 20mA recommended 10mA
Absolute MAX 200mA for entire package
www.bq.com
04 AUG 2014
ver 3 rev 1

Parte III - Instalando e conhecendo a Arduino IDE

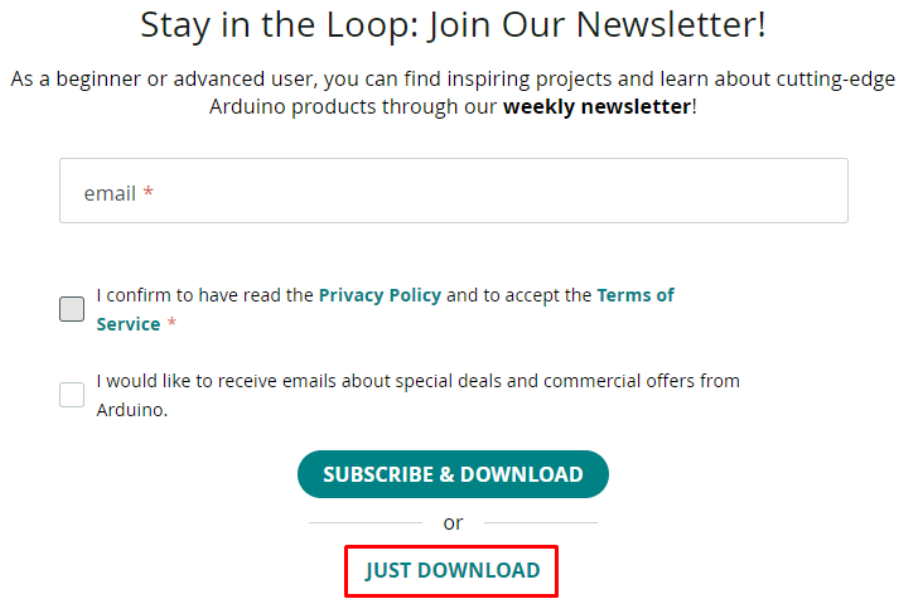
A Arduino IDE é a plataforma oficial para programação do Arduino. Vamos começar sua instalação fazendo o download do instalador, que pode ser encontrado no site oficial – clicando aqui.

Recomendamos sempre que você instale a versão estável mais recente disponível para seu sistema operacional. Como estamos utilizando o Windows 10 em nosso computador, vamos selecionar a opção ***Windows*** *Win 10 and newer, 64 bits*:

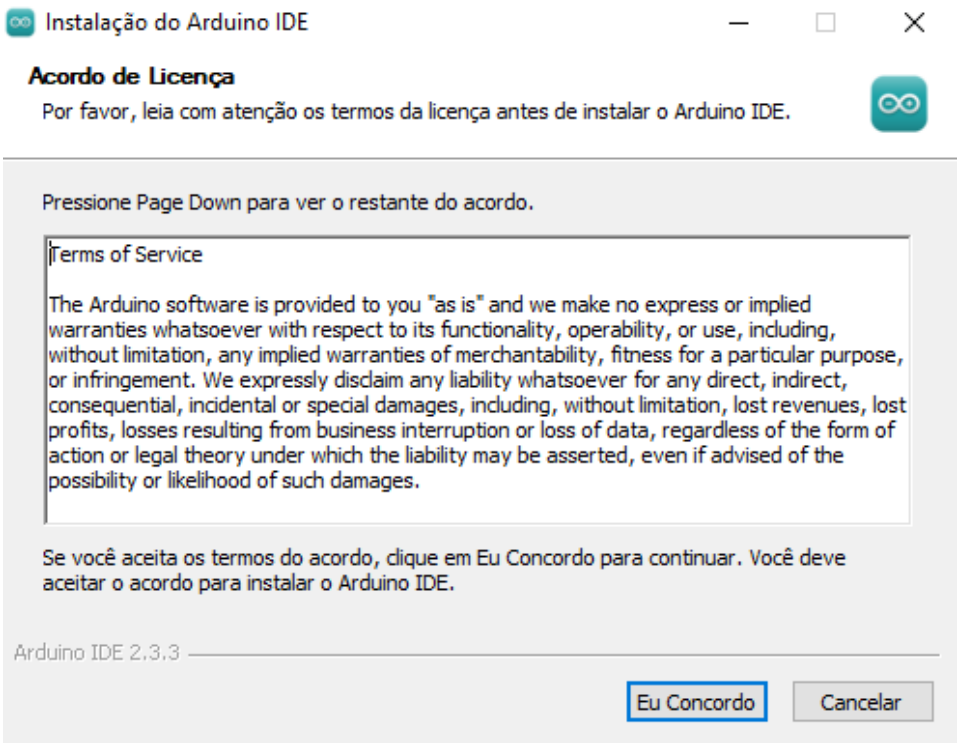


A tela seguinte nos propõe fazer uma contribuição para a plataforma, pode clicar em "Just Download" – Aqui é de suma importância que seu navegador não traduza a página, pois essa opção não é exibida quando traduzida.

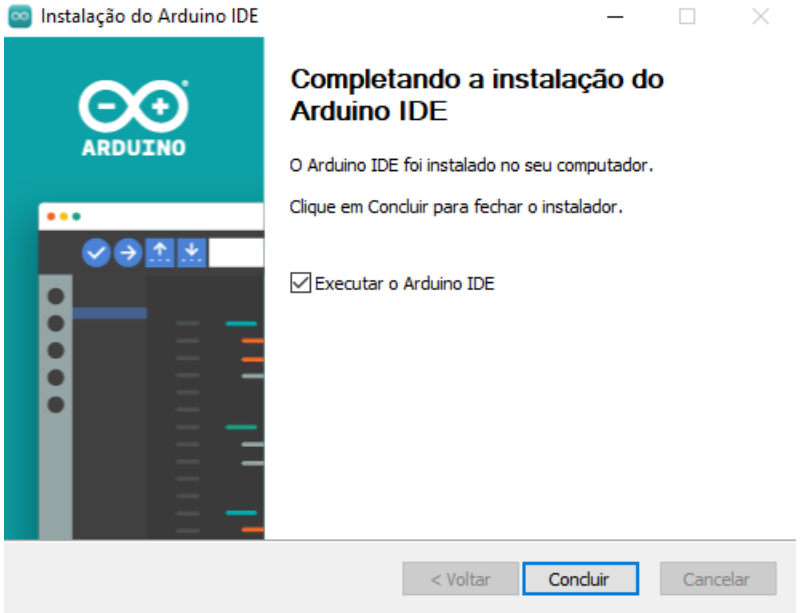
Na tela seguinte, clique novamente em “Just Download”.



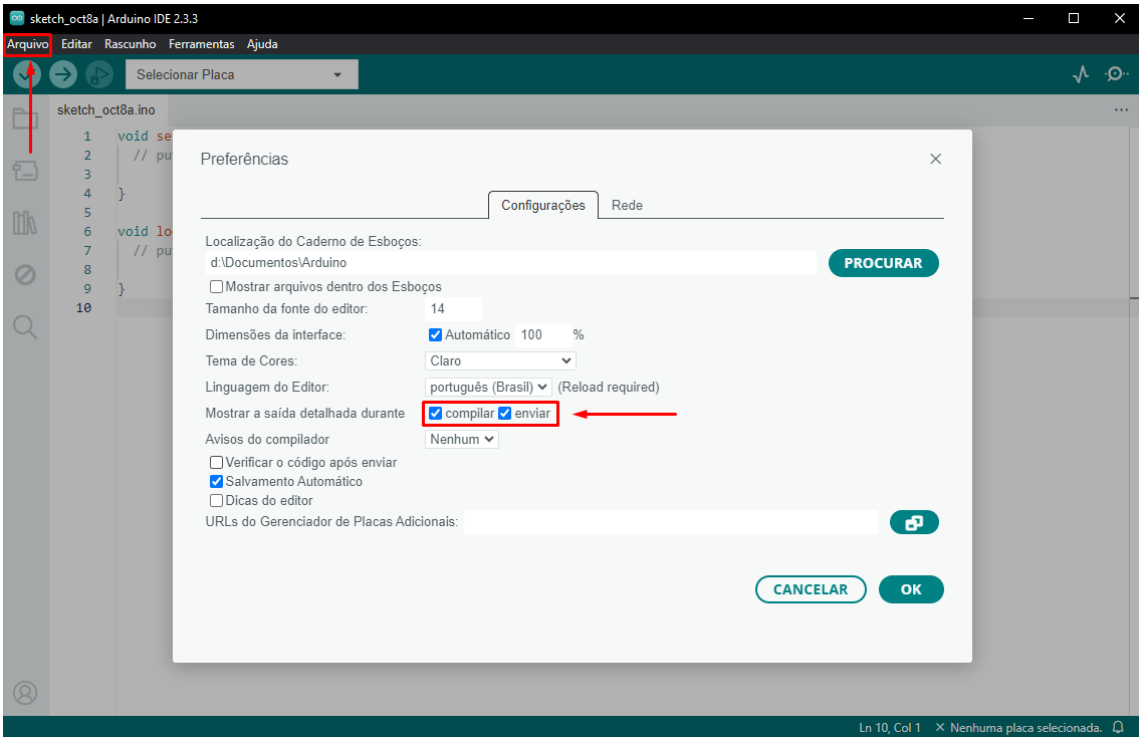
Após esses passos, o download irá iniciar automaticamente. Quando terminar, execute o arquivo baixado e clique em “Eu concordo” na janela exibida.



Nas telas seguintes, basta clicar em “Próximo” e “Instalar”. Ao término, clique em concluir para abrir a IDE.

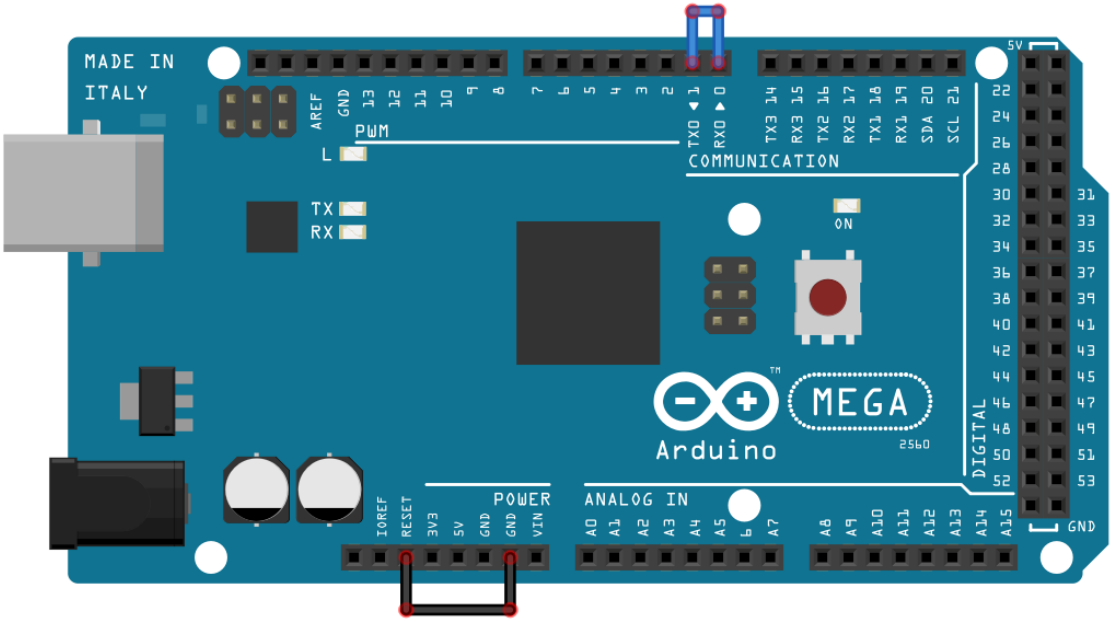


Com a IDE instalada e aberta, vamos conhecer os recursos que serão úteis nessa apostila. Primeiramente, vamos fazer uma pequena configuração, que será útil futuramente. Clique em “Arquivo”, na parte superior e vá em “Preferências”. A tela abaixo será exibida:

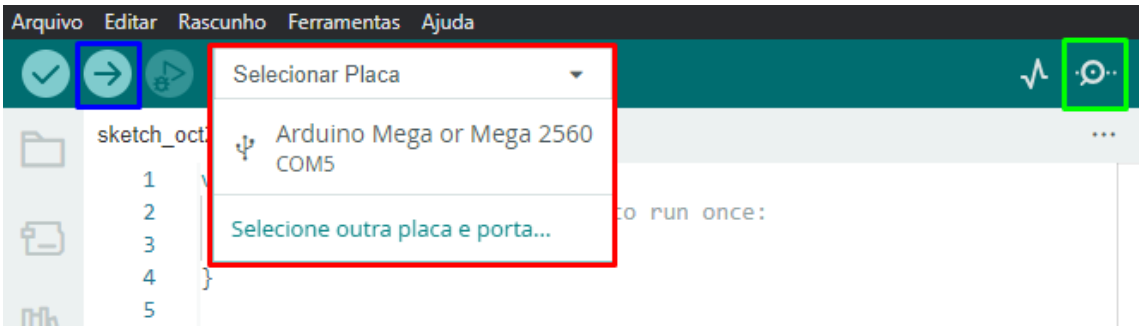


Nela, marque as caixinhas “compilar” e “enviar”. Elas são responsáveis por exibir logs e possíveis erros quando começarmos a programar o Arduino. Você também pode alterar o idioma, ativar o modo escuro e alterar a fonte, conforme sua preferência. Clique em “OK” para salvar as alterações.

Agora, vamos fazer um simples teste para verificar o funcionamento da placa. Primeiro, precisamos conectar o pino RESET do Arduino ao GND e conectar os pinos 0 (RX) e 1 (TX) entre si, usando jumpers. Isso nos permite realizar um teste chamado loopback, que nada mais é do que um teste de comunicação entre o Arduino e o computador, onde enviamos uma mensagem e a placa nos retorna exatamente o que foi enviado se tudo estiver funcionando corretamente. Abaixo, o diagrama de conexão:

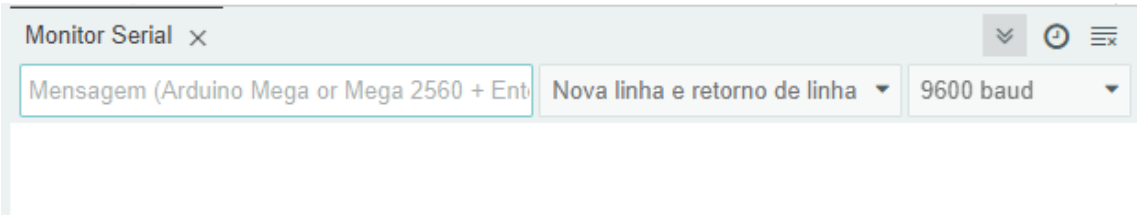


Com a conexão feita, vamos conectar a placa ao computador e preparar a Arduino IDE. Nela, precisaremos apenas selecionar o modelo de placa e a porta COM em que seu Arduino está conectado e abrir um recurso chamado Monitor Serial. Veja abaixo onde encontrá-lo:



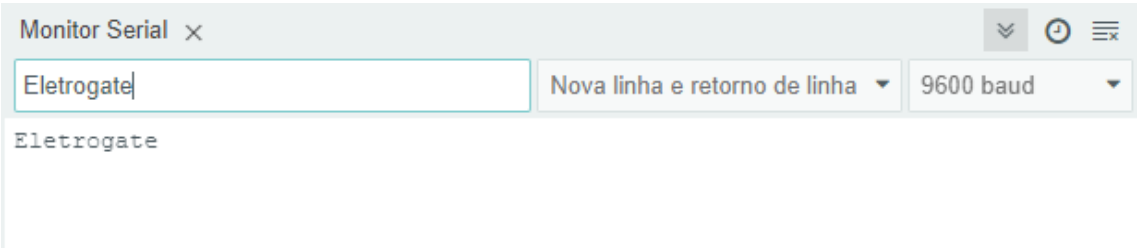
Azul: Botão “Upload”, usado para enviar o código para o Arduino;
Vermelho: Aqui você visualiza todas as placas conectadas ao computador. Selecione a que for correspondente ao seu Arduino;
Verde: Botão para abrir o monitor serial.

Especificamente nesse teste, não precisaremos carregar nenhum código, basta selecionar corretamente a placa e abrir o monitor serial. A seguinte aba será aberta na parte inferior da IDE:



Nela, podemos usar o campo de texto para enviar comandos/mensagens para o Arduino e configurar a velocidade de comunicação. Para os exemplos dessa apostila, podemos manter em 9600 bauds mesmo.

Para fazer o teste de loopback, você só precisa digitar uma mensagem e apertar a tecla Enter, para enviá-la. O Arduino deverá retorná-la na sequência, como mostrado abaixo:



Se tudo funcionou corretamente, você está pronto para iniciar a montagem e programação dos exemplos!

Parte IV - Seção de Exemplos Práticos

Agora vamos entrar nas seções de exemplos em si. Os conceitos da Parte I são importantes caso você esteja começando a trabalhar com eletrônica. No entanto, devido ao aspecto prático da montagem, não necessariamente você precisa de ler toda a parte introdutória para montar os circuitos abaixo, mas recomendamos que estude os componentes e suas particularidades para complementar seu conhecimento.

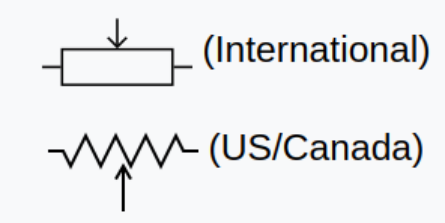
Em cada exemplo vamos apresentar os materiais utilizados, o diagrama, o código e mais: em um exemplo específico, deixamos um desafio para que você complete o funcionamento do projeto com base nos conhecimentos adquiridos.

Vamos começar!

Exemplo 1 - Leitura de Sinais Analógicos com Potenciômetro

O potenciômetro nada mais do que um resistor cujo valor de resistência pode ser ajustado de forma manual. Existem potenciômetros slides e giratórios. Na figura abaixo mostramos um potenciômetro giratório dos mais comumente encontrados no mercado.

Em ambos, ao variar a posição da chave manual, seja ela giratória ou slide, o valor da resistência entre o terminal de saída e um dos outros terminais se altera. O símbolo do potenciômetro é o mostrado na imagem a seguir:



Nesse exemplo, vamos conectar a saída de um potenciômetro a uma entrada analógica da Arduino Mega. Dessa forma, vamos ler o valor de tensão na saída do potenciômetro e vê-la variando de 0 a 1023. Mas, como assim, 0 a 1023?

Isso ocorre da seguinte maneira: ao aplicarmos uma tensão de 5V nos terminais do potenciômetro, a entrada analógica do Arduino converte esse sinal de tensão externo em um valor digital. Esse valor é representado por um número inteiro que varia de 0 a 1023, resultado da resolução do conversor analógico-digital (ADC) do Arduino, que trabalha com 10 bits. Em sistemas digitais, a base é 2 porque os dados são representados em binário, com dois estados possíveis: 0 e 1. Com 10 bits, o ADC pode representar 1024 níveis diferentes. Assim, o intervalo vai de 0 a 1023, totalizando 1024 possíveis valores. Esse total de 1024 resulta da potência 2^{10}, onde 2 é a base do sistema binário e 10 é o número de bits de resolução do ADC.

Resumindo, o Arduino divide o valor de tensão de referência (5V) em 1024 unidades (0 a 1023), totalizando 0,00488 volts por unidade. Assim, se a tensão lida na entrada analógica for de 2,5V, o valor capturado pelo Arduino será dado por 2,5/0,00488, que resulta em 512. Se for 0V, será 0, e ser for 5V, será 1023, e assim proporcionalmente para todos os valores. Assim, digamos que um valor de tensão fictício é dado por V. O valor que o Arduino vai te mostrar será o resultado da divisão de V por 5 (tensão de referência) multiplicado por 1024 (resolução do ADC). Você pode conferir a fórmula a seguir:

Valor = (V/5)*1024

Em nosso código, queremos saber o valor de tensão na saída do potenciômetro, e não um número entre 0 e 1023. Para isso, podemos rearranjar a equação da seguinte forma:

Tensão = Valor*(5/1024)

Bacana, né? Agora, vamos à montagem em si.

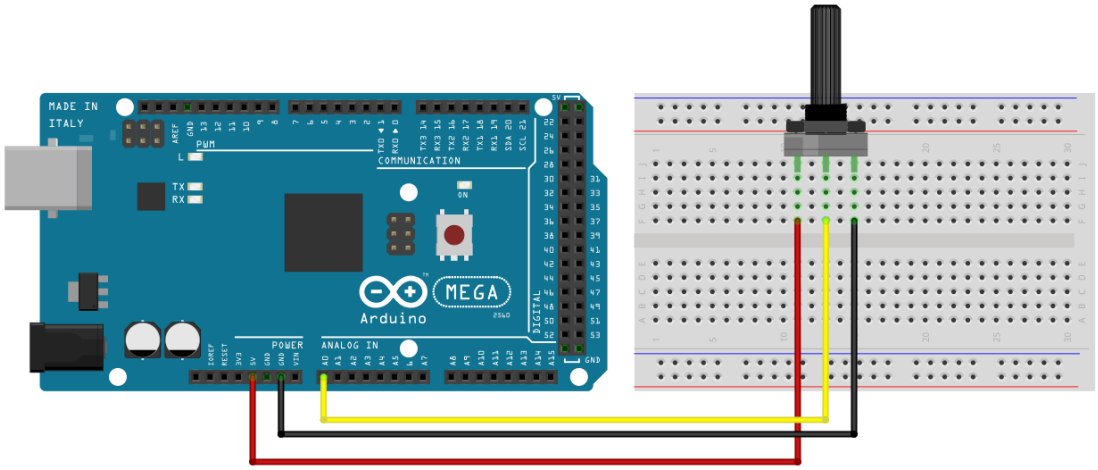
Lista de materiais:

Para esse exemplo você vai precisar de:

- Arduino Mega;
- Protoboard;
- Potenciômetro 10K;
- Jumpers.

Diagrama de circuito

Monte o circuito conforme diagrama abaixo e carregue o código de exemplo:



O Potenciômetro possui 3 terminais, sendo que o do meio é o que possui resistência variável. A ligação consiste em ligar os dois terminais fixos a uma tensão de 5V. Assim, o terminal intermediário do potenciômetro terá um valor que varia de 0 a 5V à medida que você gira seu knob.

O terminal intermediário é ligado diretamente a uma entrada analógica do Arduino (A0). Como a tensão é de no máximo 5V, então não há problema em ligar direto.

Carregue o código abaixo no Arduino e você verá as leituras no Monitor serial da IDE.

```
// Exemplo 1 - Usando potenciômetro para fazer leituras analógicas
// Apostila Eletrogate - KIT ADVANCED

#define sensorPin A0      // define entrada analógica A0

int sensorValue = 0;      // variável inteiro igual a zero
float voltage;            // variável número fracionário

void setup(){
  Serial.begin(9600);     // monitor serial - velocidade 9600 Bps
  delay(100);             // atraso de 100 milissegundos
}

void loop(){
  sensorValue = analogRead(sensorPin);       // leitura da entrada analógica A0
  voltage = sensorValue * (5.0 / 1024);      // cálculo da tensão

  Serial.print("Tensão do potenciômetro: "); // imprime no monitor serial
  Serial.print(voltage);                     // imprime a tensão
  Serial.print("    Valor: ");               // imprime no monitor serial
  Serial.println(sensorValue);               // imprime o valor
  delay(500);                                // atraso de 500 milissegundos
}
```

No código, nós declaramos a variável **sensorValue** para armazenar as leituras da entrada analógica A0 e a variável **Voltage** para armazenar o valor lido convertido para tensão. Declaramos como sendo do tipo **float** pois precisamos trabalhar com casas decimais, o que não é permitido no tipo **int**.

Na função **void setup()**, nós inicializamos o terminal serial com uma taxa de transmissão de 9600 bits por segundo. Na função **void loop(),** primeiro faz-se a leitura da entrada analógica A0 com a função **analogRead(SensorPin)** e armazenamos a mesma na variável **sensorValue**. Em seguida, aplicamos a fórmula para converter a leitura (que é um número entre 0 e 1023) para o valor de tensão correspondente. O resultado é armazenado na variável **Voltage** e em seguido mostrado na interface serial da IDE Arduino.

Exemplo 2 - Divisores de Tensão com Resistores

Esse exemplo é para ilustrar como usar um divisor de tensão. Sempre que você precisar abaixar um sinal lógico de 5V para 3.3V você pode usar esse circuito. Explicamos o divisor de tensão na seção de introdução, mais especificamente, quando conversamos sobre conversão de níveis lógicos.

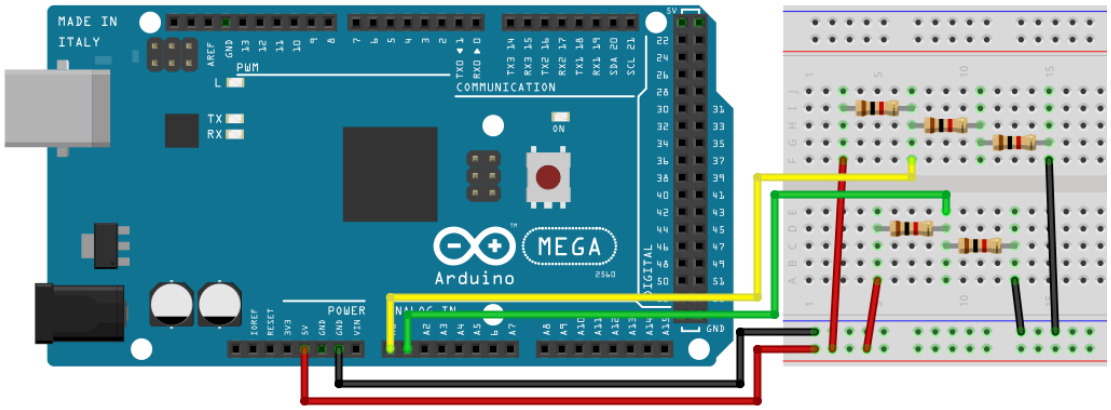
Esse circuito será útil sempre que você tiver que abaixar as saídas do Arduino de 5V para 3.3V.

Lista de materiais:

Para esse exemplo você vai precisar:

- 5 resistores de 1KΩ;
- 1 Arduino Mega;
- Protoboard;
- Jumpers.

Diagrama de circuito:



Esse é um diagrama com dois divisores de tensão. O primeiro é composto por três resistores, sendo cada um de 330R. Assim, o resistor Z1 é de 330R e o resistor Z2 é formado pela associação em série dos outros dois, resultando numa resistência equivalente de 660R.

Dessa forma, a tensão de saída do divisor é:

((1000+1000) / 1000 + (1000+1000)) * 5 = 3,33V

O segundo divisor é formado por dois resistores de 1K, dessa forma, a tensão de saída é a tensão de entrada dividida pela metade:

(1000/ (1000 + 1000)) * 5 = 2,5 V

O código para o exemplo 2 é uma extensão do código usada na seção anterior para ler valores de tensão do potenciômetro. Nesse caso, nós fazemos a leitura de dois canais analógicos (A0 e A1), fazemos as conversões para as tensões e mostramos os resultados de cada divisor na interface serial.

Carregue o código abaixo e observe os dois valores no Monitor Serial da IDE Arduino.

```
// Exemplo 2 - Divisor de tensão
// Apostila Eletrogate - KIT ADVANCED

#define sensorPin1 A0           // define entrada analógica A0
#define sensorPin2 A1           // define entrada analógica A1

int sensorValue1 = 0;           // variável inteiro igual a zero
int sensorValue2 = 0;           // variável inteiro igual a zero
float voltage1;                 // variável número fracionário
float voltage2;                 // variável número fracionário

void setup()
{
  Serial.begin(9600);           // monitor serial - velocidade 9600 Bps
  delay(100);                   // atraso de 100 milissegundos
}

void loop()
{
  sensorValue1 = analogRead(sensorPin1);   // leitura da entrada analógica A0
  sensorValue2 = analogRead(sensorPin2);   // leitura da entrada analógica A1
  voltage1 = sensorValue1 * (5.0 / 1024);  // cálculo da tensão 1
  voltage2 = sensorValue2 * (5.0 / 1024);  // cálculo da tensão 2
  Serial.print("Tensão do divisor 1: ");   // imprime no monitor serial
  Serial.print(voltage1);                  // imprime a tensão 1
  Serial.print("  Tensão do divisor 2: "); // imprime no monitor serial
  Serial.println(voltage2);                // imprime a tensão 2
  delay(500);                              // atraso de 500 milissegundos
}
```

Exemplo 3 - Controle de LEDs com Botões

Os push-buttons (chaves botão) e LEDs são elementos presentes em praticamente qualquer circuito eletrônico. As chaves são usadas para enviar comandos para o Arduino e os LEDs são elementos de sinalização luminosa.

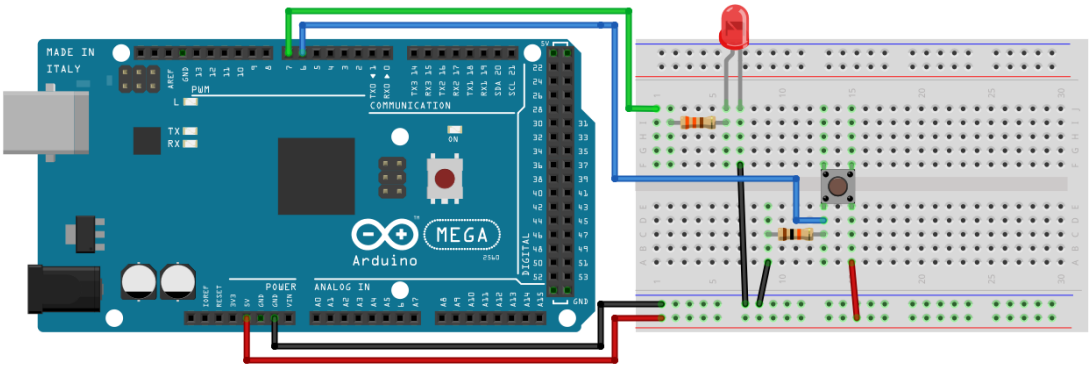
Esses dois componentes são trabalhados por meio das entradas e saídas digitais do Arduino. Neste exemplo vamos fazer uma aplicação básica que você provavelmente vai repetir muitas vezes. Vamos ler o estado de um push-button e usá-la para acender ou apagar um LED.

Lista de materiais:

Para esse exemplo você vai precisar:

- Resistor de 330Ω;
- Resistor de 1KΩ;
- LED vermelho (ou de outra cor de sua preferência);
- Push-button (chave botão);
- Arduino Mega;
- Protoboard;
- Jumpers.

Diagrama de circuito:



Esse pequeno código abaixo mostra como ler entradas digitais e como acionar as saídas digitais. Na função **void setup(),** é preciso configurar qual pino será usado como saída e qual será usado como entrada.

Depois de configurar os pinos, para acioná-los basta chamar a função **digitalWrite(pino, HIGH).** A função **digitalWrite()** liga ou desliga um pino digital dependendo do valor passado no argumento. Se for “HIGH”, o pino é ativado. Se for “LOW”, o pino é desativado.

Na função **void loop(),** fizemos um if no qual a função **digitalRead** é usada para saber se o push-button está acionado ou não. Caso ele esteja acionado, nós acendemos o LED, caso ele esteja desligado, nós desligamos o LED.

Carregue o código abaixo e pressione o botão para acender o LED.

```
// Exemplo 3 - Entradas e saídas digitais - push-button + led
// Apostila Eletrogate - KIT ADVANCED

#define PinButton 6                    // define pino digital D6
#define ledPin 7                       // define pino digital D7

void setup()
{
  pinMode(PinButton, INPUT);           // configura D8 como entrada digital
  pinMode(ledPin, OUTPUT);             // configura D7 como saída digital
  Serial.begin(9600);                  // monitor serial - velocidade 9600 Bps
  delay(100);                          // atraso de 100 milissegundos
}

void loop()
{
  if (digitalRead(PinButton) == HIGH)  // se chave = nível alto
  {
    digitalWrite(ledPin, HIGH);         // liga LED com 5V
    Serial.println("Acendendo LED");      // imprime no monitor serial
  }
  else                                  // senão chave = nível baixo
  {
    digitalWrite(ledPin, LOW);          // desliga LED com 0V
    Serial.println("Desligando LED");     // imprime no monitor serial
  }
  delay(100);                           // atraso de 100 milissegundos
}
```

Exemplo 4 - Acionamento de buzzer com um botão

O buzzer é um dispositivo eletrônico que que transforma energia elétrica em som, através da vibração de uma membrana interna. Essa vibração é feita por um circuito chamado oscilador, que pode ser interno (quando presente no corpo do buzzer, recebendo o nome de **buzzers ativos**) ou externo, quando é necessário que esse circuito seja conectado ao buzzer (recebendo o nome de **buzzers passivos**).

Os buzzers ativos diferenciam-se dos passivos também na frequência do som emitido, já que, por conter internamente o circuito oscilador, geralmente não é possível variar a frequência em que esse circuito oscila. Já nos buzzers passivos, como o oscilador é externo ao componente, temos total controle das frequências emitidas, sendo esse o modelo ideal para projetos em que necessitamos emitir sons específicos.

Buzzer passivo e buzzer ativo, respectivamente

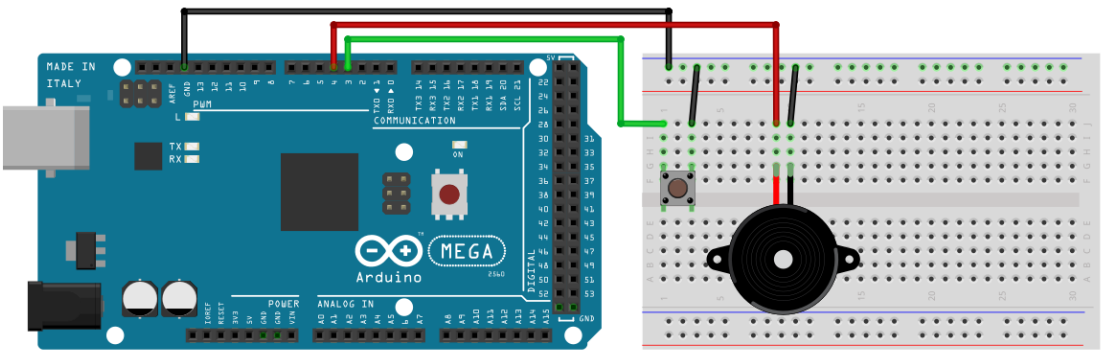
Repare que o circuito do buzzer ativo não é acessível, estando coberto com resina. Já o buzzer passivo você pode visualizar uma pequena placa de circuito interna, onde os terminais estão conectados.

Nosso kit traz o buzzer ativo de 5V, que é mais fácil de acionar e desenvolver projetos, já que, para emitir som, precisa apenas ser energizado. No projeto a seguir, vamos utilizar o Arduino e um botão para fazer o acionamento desse buzzer.

Lista de materiais:

- Arduino Mega;
- Protoboard;
- Push Button;
- Jumpers.

Diagrama de circuito:



Carregue o código abaixo em seu Arduino:

```
//Exemplo 4 - Acionamento de buzzer com um botão
//Apostila Eletrogate - KIT ADVANCED

#define BUTTON 3
#define BUZZER 4

void setup(){
  pinMode(BUTTON, INPUT_PULLUP);
  pinMode(BUZZER, OUTPUT);
  Serial.begin(9600);
  delay(100);
}

void loop(){
  if (digitalRead(BUTTON) == LOW){
    digitalWrite(BUZZER, HIGH);
  } else {
    digitalWrite(BUZZER, LOW);
  }
  delay(100);
}
```

Exemplo 5 - Acionamento de LED conforme do Luz Ambiente

O sensor LDR é um sensor de luminosidade. LDR é a sigla para **Light Dependent Resistor**, ou seja, um resistor cuja resistência varia com a quantidade de luz que incide sobre ele. Esse é seu princípio de funcionamento.

Quanto maior a luminosidade em um ambiente, menor a resistência do LDR. Essa variação na resistência é medida através da queda de tensão no sensor, que varia proporcionalmente (de acordo com a lei Ohm, lembra?) com a queda na resistência elétrica.

A imagem abaixo mostra o sensor em mais detalhes:

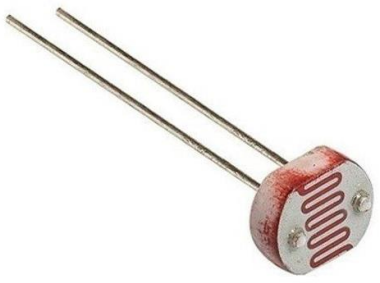


Foto resistor (LDR)

É importante considerar a potência máxima do sensor, que é de 100 mW. Ou seja, com uma tensão de operação de 5V, a corrente máxima que pode passar por ele é de 20 mA. Felizmente, com 8KΩ (que medimos experimentalmente com o ambiente bem iluminado), que é a resistência mínima, a corrente ainda está longe disso, sendo 0,625mA. Dessa forma, podemos conectar o sensor diretamente ao Arduino.

**Nota: Em suas medições, pode ser que encontre um valor de resistência mínimo diferente, pois depende da iluminação local e da própria fabricação do componente em si.*

Este sensor de luminosidade pode ser utilizado em projetos com o Arduino e outros microcontroladores para diversas aplicações, como alarmes, automação residencial, sensores de presença e vários outros.

Nesse exemplo, vamos usar uma entrada analógica do Arduino para ler a variação de tensão no LDR e, consequentemente, saber como a luminosidade ambiente está se comportando. Veja na especificação que, com muita luz, a resistência fica em torno de 10-20KΩ, enquanto no escuro pode chegar a 1MΩ.

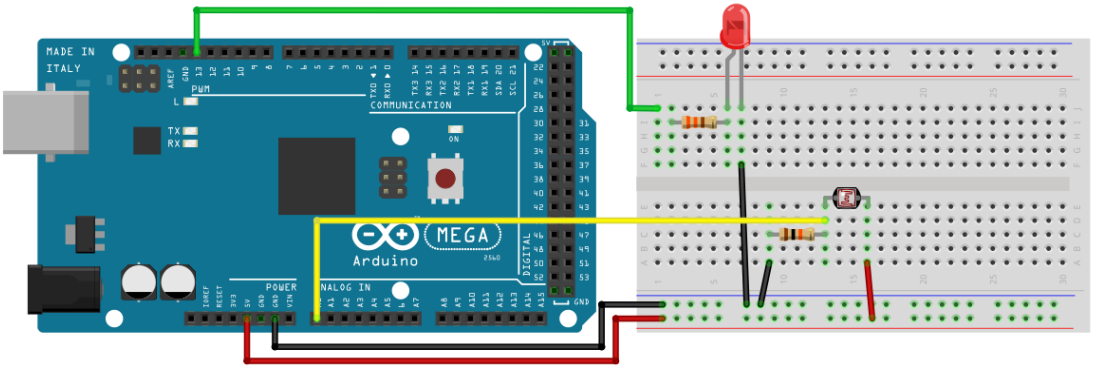
Para podermos ler as variações de tensão resultantes da variação da resistência do LDR, vamos usar o sensor como parte de um divisor de tensão. Assim, a saída do divisor será dependente apenas da resistência do sensor, pois a tensão de entrada e a outra resistência são valores conhecidos. Em nosso caso, vamos usar um resistor de 10K e uma tensão de operação de 5V. Assim, o sinal que vamos ler no Arduino terá uma variação de 2,2V (quando o LDR for 8KΩ) e 5V (quando o LDR tiver resistências muito maiores que o resistor de 10KΩ).

Lista de materiais:

Para esse exemplo você vai precisar:

- LDR;
- Resistor de 10KΩ;
- 1 Arduino Mega;
- Protoboard;
- Jumpers.

Diagrama de circuito:



No diagrama, o sensor é ligado como parte de um divisor de tensão no pino analógico A0, de forma que a tensão de saída do divisor varia de acordo com a variação da resistência do sensor. Assim, vamos identificar as variações na intensidade de luz pelas variações na tensão do sensor.

Quanto maior a intensidade de luz, menor a resistência do sensor e, consequentemente, menor a tensão de saída.

Carregue o código abaixo e varie a luminosidade sobre o LDR.

```
// Exemplo 5 - Sensor de luz LDR
// Apostila Eletrogate - KIT ADVANCED

#define AnalogLDR A0                    // define pino analógico A0
#define Limiar 1.5                      // define constante igual a 1.5
#define ledPin 13                       // define pino digital D13

int Leitura = 0;                        // variável inteiro igual a zero
float VoltageLDR;                       // variável número fracionário
float ResLDR;                           // variável número fracionário

void setup()
{
  pinMode(ledPin, OUTPUT);                  // configura D13 como saída digital
  Serial.begin(9600);                       // monitor serial - velocidade 9600 Bps
  delay(100);                               // atraso de 100 milissegundos
}

void loop()
{
  Leitura = analogRead(AnalogLDR);          // leitura da tensão no pino analógico
A0
  VoltageLDR =  Leitura * (5.0/1024);       // cálculo da tensão no LDR
  Serial.print("Leitura sensor LDR = ");    // imprime no monitor serial
  Serial.println(VoltageLDR);               // imprime a tensão do LDR

  if (VoltageLDR > Limiar)                  // se a tensão LDR maior do que limiar
    digitalWrite(ledPin, HIGH);             // liga LED com 5V
  else                                      // senão a tensão LDR < limiar
    digitalWrite(ledPin, LOW);              // desliga LED com 0V
  delay(500);                               // atraso de 500 milissegundos
}
```

No código acima usamos funcionalidades de todos os exemplos anteriores. Como o sensor é um elemento de um divisor de tensão, efetuamos sua leitura do mesmo modo que nos exemplos do potenciômetro e divisor de tensão.

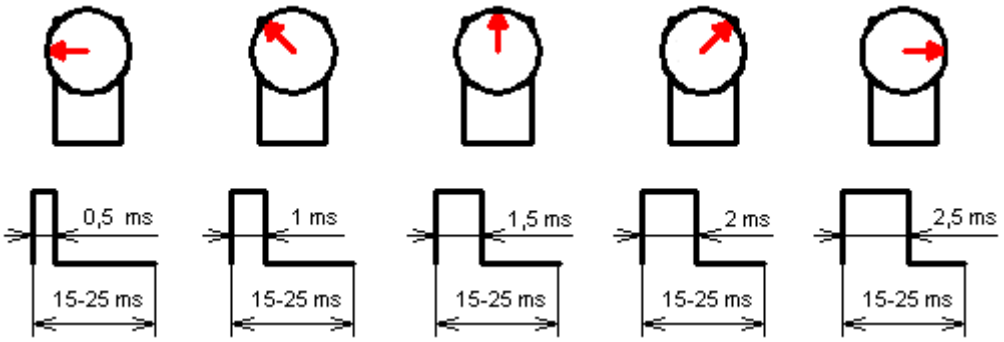
Nesse caso, definimos uma tensão de limiar, a partir da qual desligamos ou ligamos um LED para indicar que a intensidade da luz ultrapassou determinado valor. Esse limiar pode ser ajustado por você para desligar o led em intensidades diferentes de luz ambiente.

É importante que o sensor esteja exposto à iluminação ambiente e não sofra interferência de fontes luminosas próximas, que não sejam parte do ambiente.

Exemplo 6 - Controle de Servo Motor com Potenciômetro

Um servomotor é um equipamento eletromecânico que possui um encoder e um controlador acoplado. Diferentemente de motores tradicionais, como os de corrente contínua, o servo motor apresenta movimento rotativo proporcional a um comando de forma a atualizar sua posição. Ao invés de girar continuamente como os motores de corrente contínua convencionais, o servo, ao receber um comando, gira até a posição especificada pelo comando recebido.

Ou seja, o servo motor é um atuador rotativo para controle de posição, que atua com precisão e velocidade controlada em malha fechada. De acordo com a largura do pulso aplicado no pino de controle PWM, a posição do rotor é definida (0 a 180 graus) . Os pulsos devem variar entre 0,5 ms. e 2,5 ms.



Posição do Servo de acordo com a largura do pulso
Fonte : electronics.stackexchange.com

Existem dois tipos de Servomotores :

- Servomotor analógico
- Servomotor digital

Servomotores analógicos são os mais comuns e mais baratos. O controle da posição do rotor utiliza um método analógico através da leitura de tensão sobre um potenciômetro interno.

No caso dos servomotores digitais, mais caros e mais precisos, o controle da posição do rotor utiliza um encoder digital.

A figura abaixo mostra um típico **servo motor analógico**. Trata-se do **Micro Servo Tower Pro SG90 9G** que acompanha o Kit Advanced para Arduino.

Os Servos são acionados por meio de três fios, dois para alimentação e um correspondente ao sinal de controle para determinar a posição. Em geral, os três fios são:

- Marrom ou preto: GND;
- Vermelho: Alimentação positiva;
- Laranja ou branco: Sinal de controle PWM.

Lista de materiais:

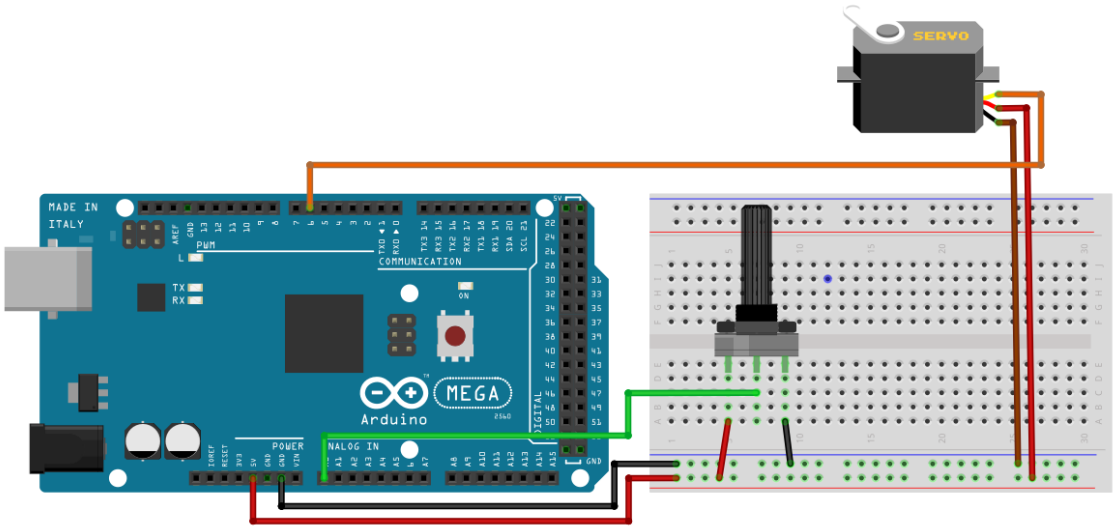
Inicialmente, vamos separar os componentes que precisamos para montar o servomotor junto com o Arduino. A nossa lista de componentes é a seguinte:

- Micro Servo 9g SG90 TowerPro;
- Arduino Mega + cabo USB;
- Potenciômetro de 10KΩ;
- Jumpers para conexão na protoboard;
- Push-button.

A montagem é simples. O servomotor em si deve ser ligado à alimentação conforme as cores apresentadas na introdução. Para esse exemplo específico não é necessário utilizar nenhuma fonte externa para alimentar o servo, no entanto, **é de suma importância que a utilize em aplicações que demandam movimentações mais complexas ou que exijam que o servo permaneça ligado por muito tempo.** Nesse caso, basta conectar o positivo da fonte no fio vermelho do servo e o GND da fonte no fio marrom/preto do servo e no GND do Arduino.

Diagrama de circuito

A montagem para uma aplicação de controle do servo em qualquer posição, fica da seguinte forma:



Carregue o código a seguir e gire o potenciômetro para o Servo motor girar:

```
// Exemplo 6 - Acionando o Micro Servo TowerPro
// Apostila Eletrogate - KIT ADVANCED

#include <Servo.h>                      // usando biblioteca Servo

Servo myservo;                          // cria o objeto myservo

#define potpin A0                       // define pino analógico A0

int val;                                // variável inteiro

void setup()
{
  myservo.attach(6);                    // configura pino D6 - controle do Servo
}

void loop()
{
  val = analogRead(potpin);             // leitura da tensão no pino A0
  val = map(val, 0, 1023, 0, 179);      // converte a leitura em números (0-179)
  myservo.write(val);                   // controle PWM do servo
  delay(15);                            // atraso de 15 milissegundos
}
```

O aspecto mais importante desse software é a utilização da biblioteca **servo.h**. Bibliotecas nada mais são que conjuntos de código prontos para facilitar a programação, e essa possui as funções necessárias para posicionar a servo para a posição desejada. Na função **Void Setup()** nós associamos o objeto servomotor, do tipo Servo, a um pino do Arduino. Na mesma função, nós inicializamos o servo na posição 0º, utilizando o método **write** do objeto que acabamos de criar.

Na função **Void Loop(),** o procedimento consiste em ler o valor do potenciômetro e usá-lo como referência para atualizar a posição do servo. A leitura analógica do potenciômetro retorna um valor entre 0 e 1023. Para controlar o servo nós usamos valores de 0 a 179, correspondendo ao meio giro de 180º do mesmo. Assim, é necessário usar a função **Map()** para traduzir a escala de 0-1023 para a escala de 0-179. Dessa forma, os valores lidos do potenciômetro podem ser usados como referência para determinar a posição do servomotor.

Para atualizar a posição do servo a cada contagem do loop, nós chamamos o método **write()** do objeto servomotor, passando como parâmetro o valor lido do potenciômetro traduzido para a escala de 0-179. Assim, sempre que mexermos no potenciômetro, o servo motor irá atuar e atualizar a sua posição.

Exemplo 7 - Medindo a Temperatura do Ambiente

O sensor de temperatura NTC pertence a uma classe de sensores chamada de Termistores. São componentes cuja resistência é dependente da temperatura. Para cada valor de temperatura absoluta há um valor de resistência.

Assim, o princípio para medir o sinal de um termistor é o mesmo que usamos para medir o sinal do LDR. Vamos medir na verdade, a queda de tensão provocada na resistência do sensor.

Existem dois tipos básicos de termistores:

- **PTC (Positive temperature coeficient):** Nesse sensor, a resistência elétrica aumenta à medida que a temperatura aumenta;

- **NTC (Negative Temperature Coeficient):** Nesse sensor, a resistência elétrica diminui à medida que a temperatura aumenta.

O termistor que acompanha o kit é do tipo NTC, como o próprio nome diz. Esse é o tipo mais comum do mercado.

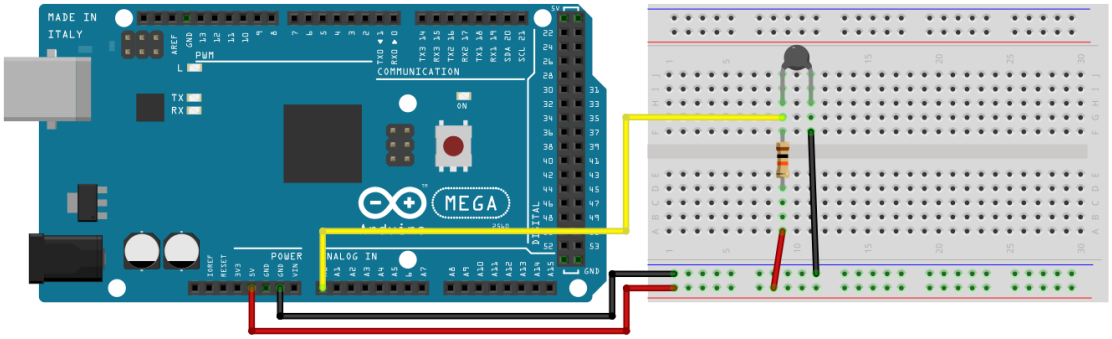
O grande problema dos termistores é que é necessária fazer uma função de calibração, pois a relação entre temperatura e resistência elétrica não é linear. Existe uma equação, chamada equação de **Stein Hart-Hart** que é usada para descrever essa relação (vide referências).

O código que vamos usar nesse exemplo utiliza a equação de Stein-Hart conforme a referência 1.

Lista de materiais:

- 1 Sensor de temperatura NTC;
- Arduino Mega + cabo USB;
- 1 resistor de 10KΩ;
- Protoboard;
- Jumpers para conexão no protoboard.

Diagrama de circuito:



Carregue o código abaixo e meça a temperatura com o NTC:

```
// Exemplo 7 - Sensor de temperatura NTC
// Apostila Eletrogate - KIT ADVANCED

#define Vin 5.0                 // define a tensão de entrada igual a 5.0
#define T0 298.15               // define constante igual a 298.15 Kelvin
#define Rt 10000                // Resistor do divisor de tensão
#define R0 10000                // Valor da resistência inicial do NTC
#define T1 273.15               // [K] in datasheet 0º C
#define T2 373.15               // [K] in datasheet 100° C
#define RT1 35563               // [ohms] Resistência in T1
#define RT2 549                 // [ohms] Resistência in T2
float beta = 0.0;               // parâmetros iniciais [K]
float Rinf = 0.0;               // parâmetros iniciais [ohm]
float TempKelvin = 0.0;         // variable output
float TempCelsius = 0.0;        // variable output
float Vout = 0.0;               // Vout in A0
float Rout = 0.0;               // Rout in A0

void setup(){
  Serial.begin(9600);                       // monitor serial - velocidade 9600 Bps
  beta = (log(RT1 / RT2)) / ((1 / T1) - (1 / T2));     // cálculo de beta
  Rinf = R0 * exp(-beta / T0);              // cálculo de Rinf
  delay(100);                               // atraso de 100 milissegundos
}

void loop(){
  Vout = Vin * ((float)(analogRead(0)) / 1024.0); // cálculo de V0 e leitura de A0
  Rout = (Rt * Vout / (Vin - Vout));              // cálculo de Rout
  TempKelvin = (beta / log(Rout / Rinf));         // cálculo da temp. em Kelvins
  TempCelsius = TempKelvin - 273.15;              // cálculo da temp. em Celsius
  Serial.print("Temperatura em Celsius: ");       // imprime no monitor serial
  Serial.print(TempCelsius);                      // imprime temperatura Celsius
  Serial.print("   Temperatura em Kelvin: ");     // imprime no monitor serial
  Serial.println(TempKelvin);                     // imprime temperatura Kelvins
  delay(500);                                     // atraso de 500 milissegundos
}
```

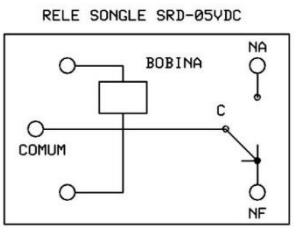
Exemplo 8 - Acionamento de uma Lâmpada com Relé

O Módulo Relé nada mais é do que um ou mais relés acionados por sinais digitais de 5V. Esse componente é indicado para acionar cargas que utilizam correntes maiores do que a suportada pelo Arduino, ou então uma carga de um circuito de corrente alternada (rede elétrica).

O módulo de relé pode ser usado para vários tipos de aplicações:

- ativação de eletroímãs,

- ativação de motores CC,
- ativação de motores CA,
- ligar/desligar lâmpadas.



O terminal de controle do relé (bobina) é ligado a um dos pinos digitais do Arduino. O outro terminal da carga é ligado ao outro terminal da tomada (ou ao outro terminal da bateria).

Uma das aplicações mais comuns dos módulos relés para Arduino é no acendimento de lâmpadas em sistemas de automação residencial. O módulo funciona exatamente da mesma forma que uma chave ou interruptor. Ao lado temos a pinagem de um relé comum:

- C - Terminal Comum
- NA - Contato Normalmente Aberto
- NF - Contato Normalmente Fechado

O terminal C é o Comum do Contato. Os outros dois terminais usamos para ligar ou desligar a carga. Se ligamos a carga no terminal NA, a carga será acionada quando enviarmos um sinal. Se ligarmos a carga no pino NF, ela será ativada setando o pino de controle do Arduino para nível baixo, e desligada setando o pino de controle para nível alto.

A capacidade corrente em cada contato é de 10 A. Não ultrapasse esse valor, para não danificar o seu relé.

Os outros dois terminais são para alimentação da bobina do relé – no caso 5V . O terminal de controle do Módulo está conectado nessa bobina. O consumo de corrente da bobina do relé é de aproximadamente 60 mA, quando energizada.

No mercado você pode encontrar módulos de 1 , 2, 4 e 8 canais. Cada canal corresponde a um relé montado no módulo. Ou seja, um módulo relé de 5V e 8 canais, é um conjunto de 8 relés acionados por sinais separados de 5V. Um módulo relé de 12V e 4 canais, nada mais do que um conjunto de 4 relés, cada qual acionado por sinais de 12V.

Na imagem abaixo temos como exemplo um relé de 4 canais.

Repare que cada relé possui os três terminais que explicamos mais acima. Além deles, há outros 6 pinos usados para interfacear o módulo com o Arduino. São eles:

- **GND,**
- **IN1,**
- **IN2,**
- **IN3,**
- **IN4,**
- **VCC.**

Cada pino **INx** serve para acionar um dos relés. **VCC** e **GND** são os pinos de alimentação do Módulo do relé : VCC deve ser conectado no +5V e o GND no negativo, ambos da fonte. **O GND do Módulo Relé deve estar conectado também no GND do Arduino !**

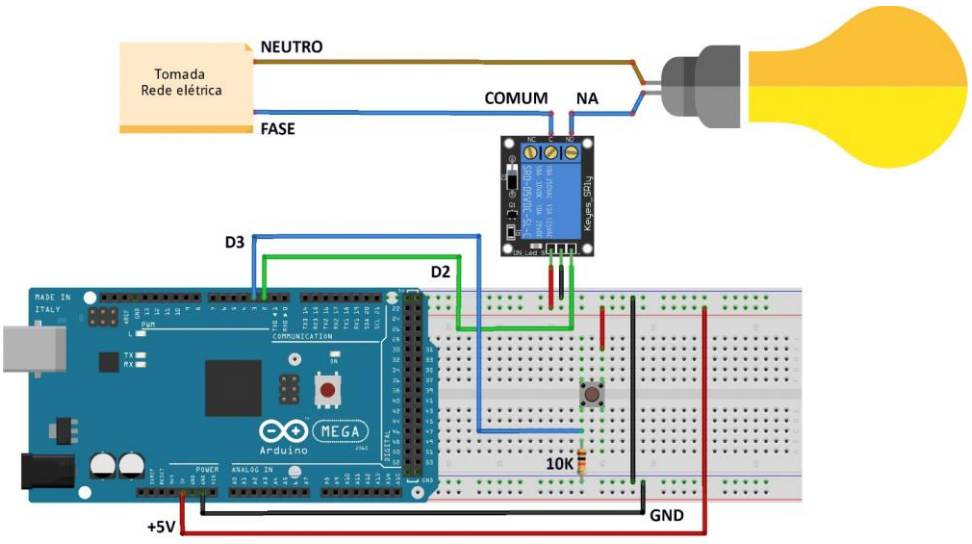
Lista de Materiais:

Para este exemplo você vai precisar:

- Protoboard,
- Arduino Mega,
- Jumpers de ligação,
- Módulo relé de 1 canal,
- Uma chave táctil (push-button),
- 1 resistor de 10K.

IMPORTANTE: se você não tem conhecimentos em montagens de circuitos elétricos CA, procure uma pessoa especializada para montar para você. Risco de choque elétrico ! Verifique todas as ligações, antes de energizar o circuito ! A Eletrogate não se responsabilizará por danos materiais ou pessoais no caso de acidentes.

Diagrama de circuito:



Código para acionar o Relé com uma chave táctil:

```
// Exemplo 8 - Módulo Relé 1 Canal
// Apostila Eletrogate - KIT ADVANCED
int buttonPin = 3;                          // pino D3 para botao
int ledPin =  13;                           // LED na placa Arduino
int IN1 = 2;                                // pino D2 para controle do rele
int ledState = LOW;                         // estado do LED = LOW
int buttonState;                            // variavel do estado do botao
int lastButtonState = LOW;                  // estado previo do botao = LOW
unsigned long lastDebounceTime = 0;         // ultimo tempo de debounce
unsigned long debounceDelay = 50;           // timer do debounce 50 ms

void setup(){
  pinMode(ledPin, OUTPUT);                  // pino do LED = saida
  pinMode(buttonPin, INPUT);                // pino do botao = entrada
  pinMode(IN1, OUTPUT);                     // pino de controle do rele = saida
  digitalWrite(ledPin, ledState);           // apaga o LED
  digitalWrite(IN1, ledState);              // desliga o rele
}
void loop(){
  int reading = digitalRead(buttonPin);     // le o estado do botao
  if (reading != lastButtonState){          // se o estado for diferente do anterior
    lastDebounceTime = millis();            // reset do timer debouncing
  }

  if ((millis() - lastDebounceTime) > debounceDelay)  {
    // se o tempo do botao pressionado for maior do que debounce
    if (reading != buttonState){             // se o estado do botao for diferente do anterior
      buttonState = reading;                // muda o estado do botao
      if (buttonState == HIGH){             // se o botao esta no nivel High
       ledState = !ledState;               // muda o estado do LED
      }
    }
  }
  digitalWrite(ledPin, ledState);           // seta o LED
  digitalWrite(IN1, ledState);              // seta o estado do rele 1
  lastButtonState = reading;                // salva a leitura do botao
}
```

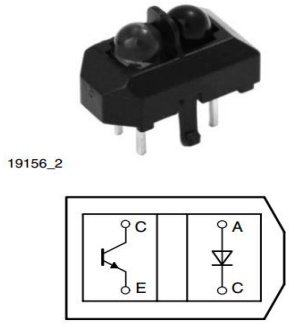
H com módulo relé 2 canais

Exemplo 9 - Acendendo um LED com Sensor Reflexivo Infravermelho

O sensor ótico reflexivo **TCRT5000** é um dos mais populares para utilização em projetos com Arduino. Trata-se de um sensor reflexivo que possui um emissor infravermelho e um fototransistor. O emissor é um led infravermelho que emite um sinal nessa faixa do espectro. Já o fototransistor é o receptor que faz a leitura do sinal refletido.

Ou seja, o led emite um feixe infravermelho que pode ou não ser refletido por um objeto. Caso o feixe seja refletido, o fototransistor identifica o sinal refletido e gera um pulso em sua saída.

A distância máxima de detecção não é grande, ficando em torno de 25 mm (dois centímetros e meio), o que pode limitar um pouco a sua aplicação. O fototransistor vem com um filtro de luz ambiente, o que maximiza a identificação do feixe infravermelho refletido.



Sensor TCRT5000 visto por cima Fonte: Vishay

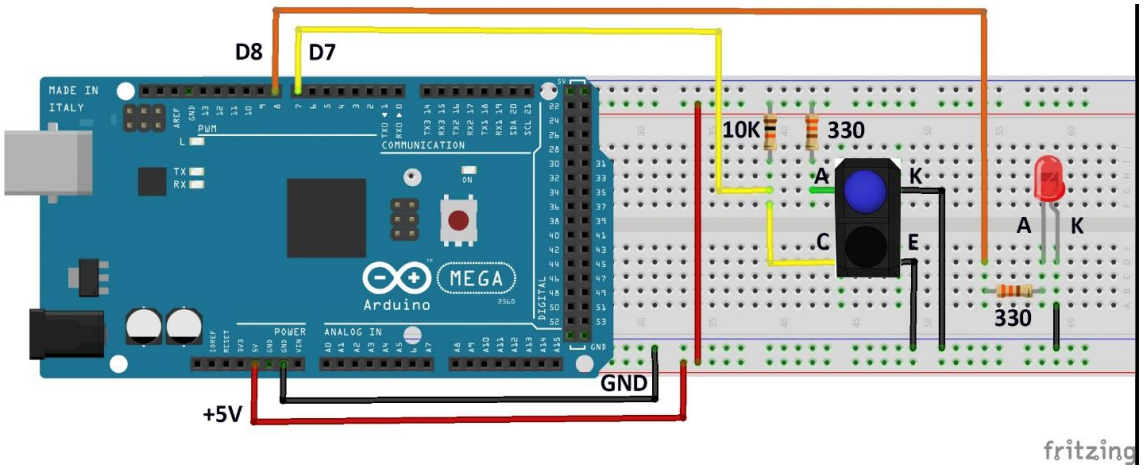
Lista de materiais:

- Arduino Mega R3;
- Protoboard;
- Sensor TCRT5000;
- Led vermelho 5mm;
- Resistor de 330;
- Resistor de 10K;
- Jumpers para ligação no protoboard;

Diagrama de Circuito:

O terminal Anodo (A) positivo do LED infravermelho está ligado por meio de um resistor de 330R ao VCC (5V) e o seu terminal catodo (K) negativo está ligado em GND. O Coletor (C) do fototransistor está ligado ao resistor de 10K (que está conectado no VCC) e também na entrada digital D7 do Arduino. Essa entrada é que vamos usar para identificar se o sensor percebeu algum objeto ou não. O Emissor (E) do fototransistor está ligado ao GND.

O LED vermelho para indicação do Sensor, tem o terminal Anodo (A) positivo conectado à um resistor de 330R que está conectado à porta digital D8 do Arduino. O terminal catodo (K) negativo está ligado ao GND.



Código:

```
// Exemplo 9  - Sensor Ótico Reflexivo TCRT5000
// Apostila Eletrogate - KIT ADVANCED

int leituraSensor;                    // variavel de leitura do sensor
int led = 8;                          // led indicador = D8 do Arduino
int fotoTransistor = 7;               // coletor do fototransistor = D7 do Arduino

void setup(){
  pinMode(led, OUTPUT);               // pino do led indicador = saida
  pinMode(fotoTransistor, INPUT);     // pino do coletor do fototransistor =
entrada
}
void loop(){
  leituraSensor = digitalRead(fotoTransistor);  // leitura do sensor TCRT5000
  if (leituraSensor == 0 ){                      // se o led refletir a luz
    digitalWrite(led, HIGH);                    // acende LED indicador
  }
  else {                                         // senão
    digitalWrite(led, LOW);                     // apaga LED indicador
  delay(500);                                   // atraso de 0,5 segundos
  }
}
```

O código para utilização do sensor TCRT5000 é bem simples. Com o circuito montado, basta monitorar o estado do pino no qual a saída do sensor (coletor do fototransistor) está ligada. Se esse pino estiver em nível baixo, é porque algum objeto está presente dentro do raio de medição do sensor (a luz infra-vermelha do LED esta sendo refletida) . Se o pino estiver em nível alto, então não há objeto nenhum no raio de medição do sensor.

Nesse exemplo, a presença do sensor é identificada pelo acionamento de um led, ou seja, sempre que um objeto for identificado pelo sensor, o led ficará aceso.

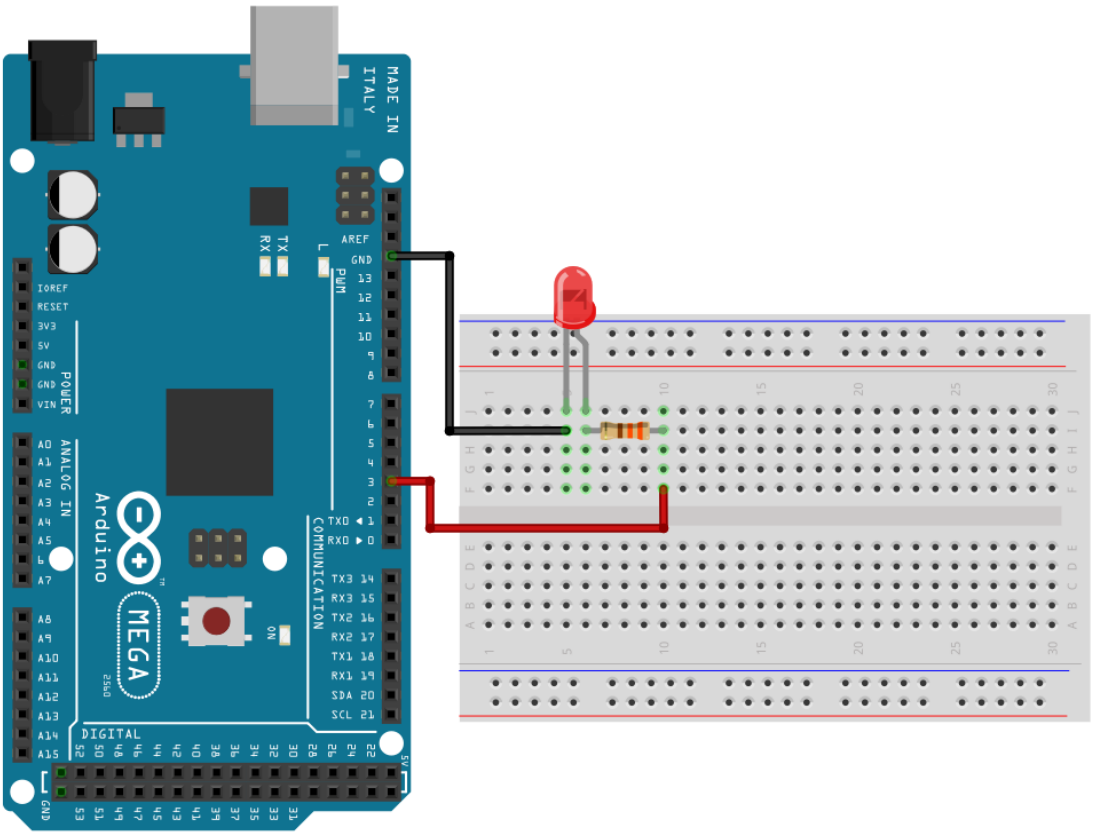
Exemplo 10 – Efeito fade

Este projeto demonstra como criar um efeito de fade no LED, onde seu brilho aumenta e diminui gradativamente e de forma automática.

Lista de materiais:

- Arduino Mega + cabo;
- LED;
- Resistor de 330Ω;
- Protoboard;
- Jumpers.

Diagrama de circuito:



Carregue o código a seguir:

```
// Exemplo 10 - Efeito fade
// Apostila Eletrogate - KIT ADVANCED

#define LED 3       //nomeia o pino 3

void setup(){
  pinMode(LED, OUTPUT);       //seta o pino 3 como saída
}

void loop(){
  for(int i=0; i<255; i++){   //incrementa os valores de 0 até 255
    analogWrite(LED, i);      //liga o LED proporcionalmente a i
    delay(10);                //aguarda 10 milissegundos
  }
  for(int i=255; i>0; i--){   //decrementa os valores de 255 até 0
    analogWrite(LED, i);      //liga o LED proporcionalmente a i
    delay(10);                //aguarda 10 milissegundos
  }
}
```

Seu funcionamento é simples e pode ser explicado usando o conceito de *iteração*. Na programação, chamamos de ***iteração*** o processo de executar um conjunto de instruções várias vezes dentro de um laço de repetição (também chamado de loop), até que a condição de verificação deste seja falsa.

Nesse exemplo, usamos o laço *for()* para iniciar uma iteração. O laço inicia a variável 'i' em zero e incrementa seu valor de 1 em 1 até que a condição "i < 255" seja falsa. A cada iteração, o brilho do LED é ajustado conforme o valor de 'i' e o programa aguarda 10 milissegundos antes de fazer o próximo incremento. Esse processo aumenta o brilho do LED de forma gradativa no primeiro *for()* e diminui, também gradativamente, no segundo *for()*.

Exemplo 11 - Controlando o brilho do LED com potenciômetro

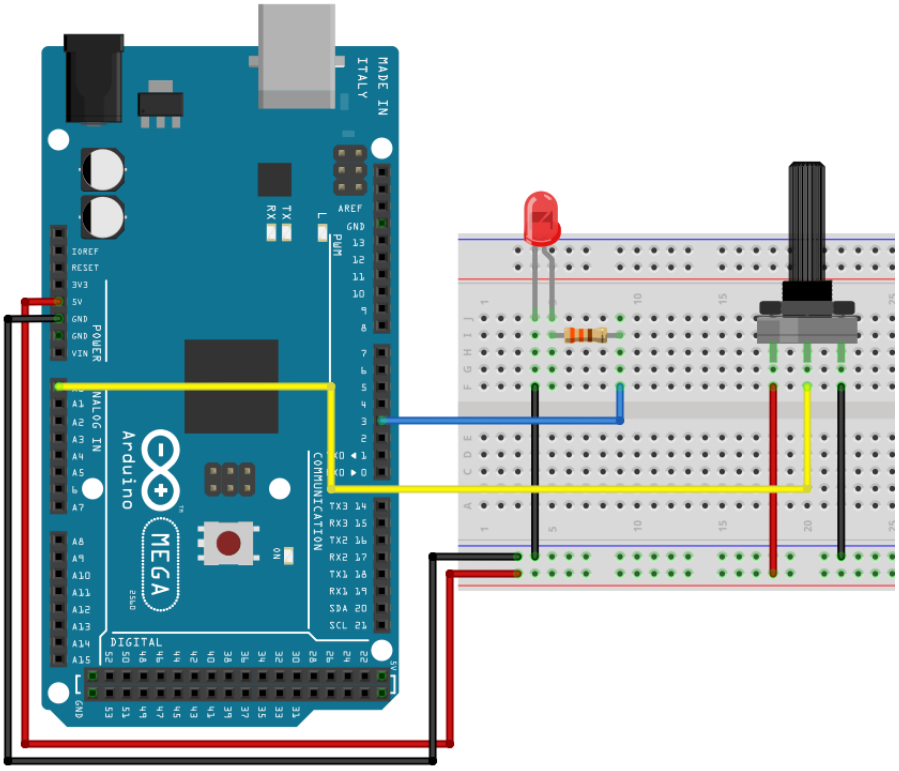
Vamos repetir os conceitos apresentados no exemplo 5 e utilizar um potenciômetro para variar o brilho de um LED. O princípio de funcionamento é o mesmo, com a diferença de que o comando map(), para o servo, vai de 0 a 180 por referenciar o ângulo de atuação, dado em graus. No caso do LED, vamos variar o valor de 0 a 255, sendo esse o intervalo válido para o controle de brilho através do sinal de PWM gerado pelo Arduino.

Lista de materiais:

- Arduino Mega + cabo;

- Potenciômetro 10KΩ;
- 1 LED;
- 1 Resistor de 330Ω
- Protoboard;
- Jumpers.

Diagrama de circuito



O código para esse projeto é simples, como podemos conferir abaixo:

```
// Exemplo 11 - Controlando o brilho do LED com potenciômetro
// Apostila Eletrogate - KIT ADVANCED

#define LED 3    //nomeia o pino do LED
#define POT A0   //nomeia o pino do potenciômetro

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

void loop(){
  int brilho = map(analogRead(POT), 0, 1023, 0, 255); //conversão de valores
  analogWrite(LED, brilho);  //varia o brilho do LED
}
```

Faça o upload do código após as conexões e gire o eixo do potenciômetro para variar o brilho do LED.

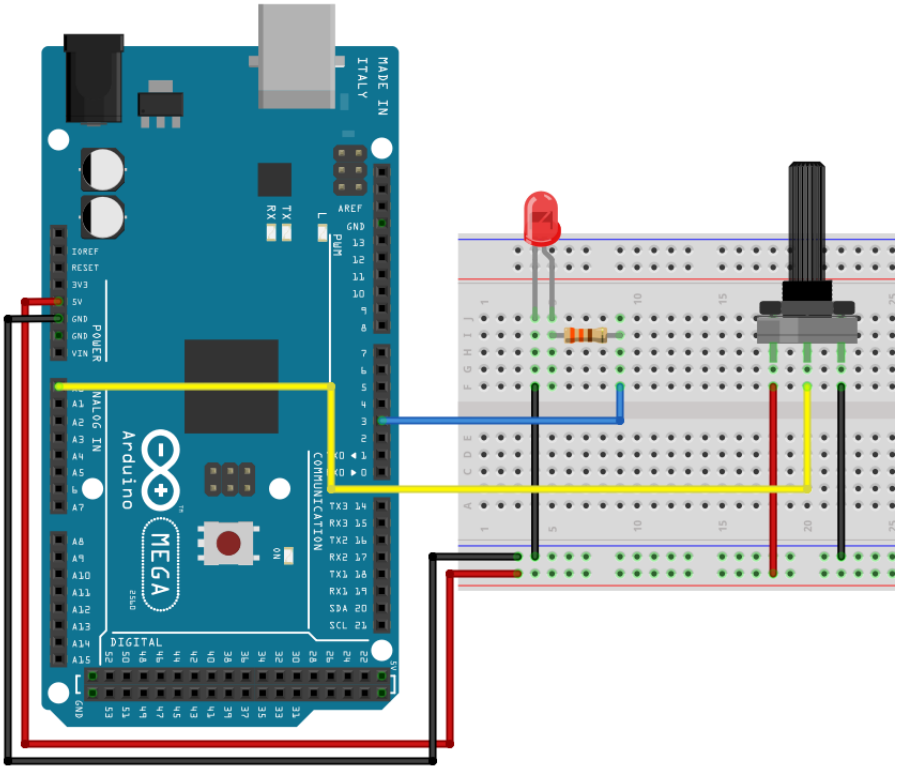
Exemplo 12 – Explorando o efeito de persistência da visão (POV) com LED

Nesse exemplo, vamos explorar o efeito POV, sigla em inglês para Persistência da Visão. Esse efeito se resume a uma ilusão de óptica, na qual nosso olho continua a ver uma imagem mesmo após a fonte de luz cessar.

Lista de materiais:

- Arduino Mega + cabo;
- Potenciômetro 10KΩ;
- 1 LED;
- 1 Resistor de 330Ω
- Protoboard;
- Jumpers.

Diagrama de circuito



Carregue o código abaixo em seu Arduino:

```
//Exemplo 12 - Controlando LEDs com funções diferentes sem usar delay()
// Apostila Eletrogate - KIT ADVANCED

//Definição dos pinos
#define ledPin 3 // Pino do LED

int intervalo; // Intervalo em milissegundos para o piscar do LED

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

void loop() {
  int leituraPotenciometro = analogRead(A0); // Lê o valor do potenciômetro

  // Mapeia a leitura do potenciômetro para um intervalo de 10 ms a 1000 ms
  intervalo = map(leituraPotenciometro, 0, 1023, 10, 1000);

  // Alterna o estado do LED com base no intervalo
  digitalWrite(ledPin, HIGH);
  delay(intervalo); // Espera o intervalo em milissegundos
  digitalWrite(ledPin, LOW);
  delay(intervalo); // Espera o intervalo em milissegundos
}
```

Repare que, conforme você gira o potenciômetro, a frequência em que o LED pisca aumenta ou diminui. Configuramos para que consiga variar de 1Hz (1x por segundo) até 100Hz (100x por segundo), de modo que, quanto mais próximo de 100Hz menos percepção do LED piscando você irá ter, graças ao efeito de persistência da visão. Legal, né?

Exemplo 13 - Controlando LEDs com funções diferentes sem usar delay()

Em projetos anteriores, utilizamos a função delay() para pausar a execução do código por um tempo determinado. No entanto, essa abordagem interrompe a execução do programa, o que pode não ser ideal para projetos onde várias ações precisam acontecer simultaneamente. Neste exemplo, vamos ensinar como controlar o brilho de um LED usando um potenciômetro ao mesmo tempo em que um segundo LED pisca de forma intermitente, sem interferir no primeiro.

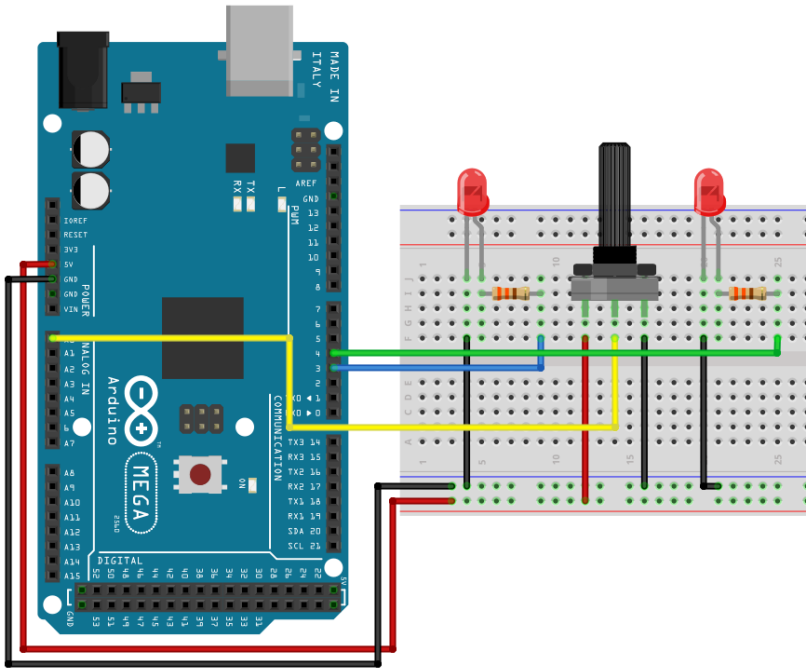
Para isso, utilizaremos a função ***millis()***, que atua como um contador interno do Arduino e começa a ser incrementado a partir do momento em que o microcontrolador é energizado ou resetado, funcionando independentemente do código em execução.

Essa função nos permite medir o tempo decorrido e gerenciar essas ações de forma eficiente, sem interromper o fluxo do programa.

Lista de materiais:

- Arduino Mega;
- Protoboard;
- Potenciômetro 10KΩ;
- 2 LEDs;
- 2 Resistores de 330Ω;
- Jumpers.

Diagrama de circuito



A lógica de funcionamento do código é bem simples. Primeiro, vamos atribuir os valores enviados pelo potenciômetro e já convertidos num intervalo entre 0 e 255 à variável brilho, em seguida usamos o comando analogWrite() para acionar o LED conforme atribuímos a função *millis()* à variável *millisPisca*, que é declarada como *unsigned long* porque precisamos armazenar valores muito grandes (já que a

contagem ocorre em milissegundos) e sabemos que não usaremos números negativos — daí o uso do termo **unsigned**, que significa 'sem sinal'.

No loop, fazemos duas verificações para saber quanto tempo passou. Primeiro, verificamos se a diferença entre o valor atual de *millis()* e o valor armazenado em *millisPisca* é menor que o intervalo definido para o LED piscar. Se essa condição for verdadeira, o LED é ligado. Se for falsa, o LED será desligado e o valor de *millisPisca* é atualizado, recebendo a contagem atual de *millis()*, reiniciando o ciclo.

Carregue o código abaixo em seu Arduino e manipule o potenciômetro para observar o funcionamento descrito:

```
//Exemplo 13 - Controlando LEDs com funções diferentes sem usar delay()
// Apostila Eletrogate - KIT ADVANCED

//Definição dos pinos
#define LED_BRILHO 3
#define LED_PISCA 4
#define POT A0

#define tempoLED_PISCA 250  //Intervalo desejado para o LED piscar

unsigned long  millisPisca = millis();  //Recebe o valor inicial de millis()

void setup(){
  //Definindo ambos os pinos dos LEDs como saída
  pinMode(LED_BRILHO, OUTPUT);
  pinMode(LED_PISCA, OUTPUT);

}

void loop (){
  int brilho = map(analogRead(POT), 0, 1023, 0, 255); //conversão de valores
  analogWrite(LED_BRILHO, brilho);  //varia o brilho do LED

  //Se a diferença entre o valor atual de millis e o valor armazenado for menor que
o intervalo desejado:
  if((millis() - millisPisca) < tempoLED_PISCA){
        digitalWrite(LED_PISCA, HIGH);   //Ligue o LED
  } else {
    digitalWrite(LED_PISCA, LOW);    //Senão, desligue o LED
    if ((millis() - millisPisca) >= 2 * tempoLED_PISCA) {  // Se um ciclo foi
completo:
      millisPisca = millis(); //atualiza millisPisca com o valor atual de millis()
    }
  }
}
```

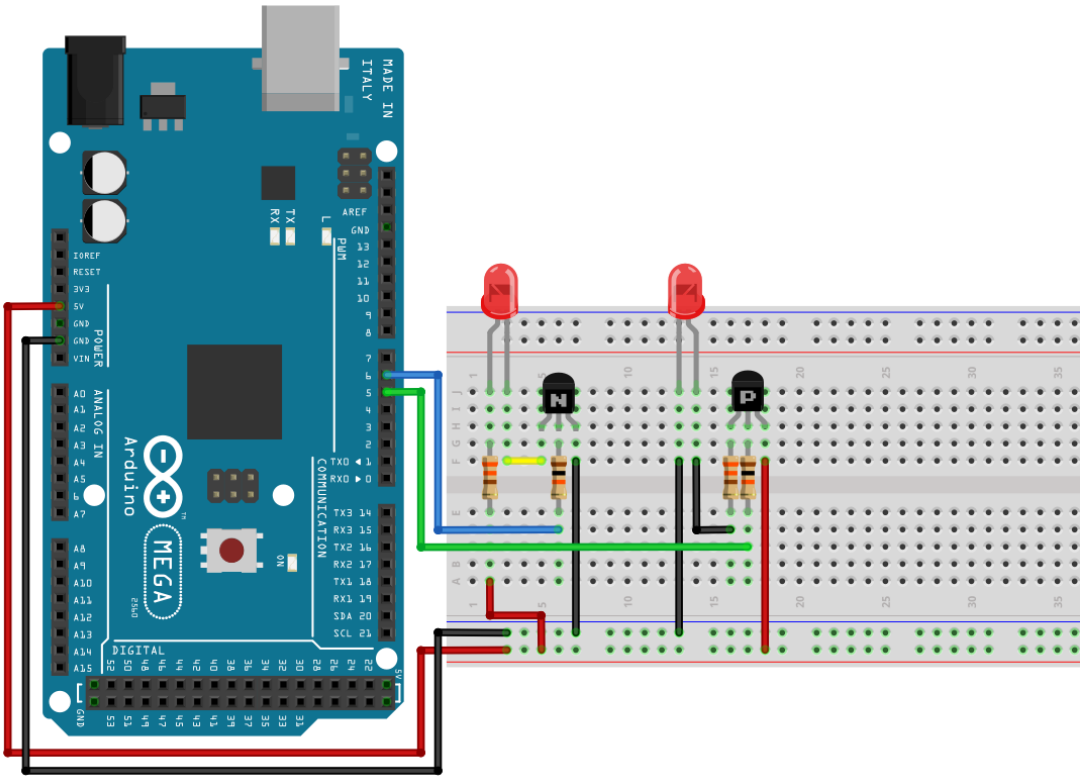
Exemplo 14 – Diferença entre transistores NPN e PNP na prática

Como vimos anteriormente, existem dois tipos de transistores, os NPN e os PNP. Nesse exemplo, vamos mostrar na prática as diferenças na conexão com a carga (em nosso caso, um LED) e as mudanças no código para o acionamento de cada um.

Lista de materiais:

- Arduino Mega;
- Protoboard;
- 2 LEDs;
- 2 Resistores de 330Ω;
- 2 resistores de 10KΩ;
- Transistor BC548;
- Transistor BC558;
- Jumpers.

Diagrama de circuito



No diagrama acima, podemos observar a diferença na conexão dos LEDs dependendo do tipo de transistor usado. Embora os transistores NPN (BC548) e PNP (BC558)

tenham a mesma disposição de terminais (Coletor – Base – Emissor), a forma como a corrente flui através deles é diferente - reveja o esquemático dos transistores, lá na página 17.

No transistor NPN (BC548), o emissor está conectado ao negativo (GND) do Arduino e o coletor ao terminal negativo do LED. Nesse caso, **considerando o sentido convencional da corrente elétrica**, os elétrons fluem do positivo (coletor) para o negativo, que está conectado ao emissor. Assim, a corrente elétrica passa primeiro pelo LED e seu resistor e depois pelo transistor, antes de chegar ao polo negativo.

No transistor PNP (BC558), o emissor está conectado ao positivo do Arduino (+5V) e o coletor ao terminal positivo do LED através do resistor. Nesse caso, os elétrons fluem do LED para o coletor do transistor e, em seguida, do emissor para o polo positivo. Para que o transistor PNP conduza, o emissor precisa estar mais positivo que a base — por isso, utilizamos o comando **digitalWrite(PNP, LOW)** para ativá-lo. Quando aplicamos um sinal negativo à base, a corrente flui do emissor para o coletor, ativando o LED.

Em ambos os casos, o fluxo da corrente é controlado pela base do transistor, que recebe um sinal do Arduino. No transistor NPN, aplicamos um sinal positivo à base (**digitalWrite(NPN, HIGH)**) para que os elétrons possam fluir do coletor para o emissor. No transistor PNP, aplicamos um sinal negativo à base para permitir que os elétrons fluam do emissor para o coletor. Isso faz com que os transistores NPN e PNP trabalhem de forma invertida no circuito.

Uma forma bem simples de entender como a corrente flui pelo transistor é observar a setinha presente no emissor, na simbologia do componente. Note que, no transistor NPN, essa seta está saindo do transistor, indicando que a corrente entra no transistor pelo coletor e sai pelo emissor.

Já no PNP, a seta está entrando no transistor, indicando que a corrente entra no transistor pelo emissor e sai pelo coletor.

Com esses conceitos em mente, carregue o código no Arduino e veja o funcionamento do circuito:

```
//Exemplo 14 - Diferença entre transistores NPN e PNP na prática
//Apostila Eletrogate - KIT ADVANCED

//Definição dos pinos
#define PNP 5
#define NPN 6

void setup() {
  //Configura os pinos como saída
  pinMode(NPN, OUTPUT);
  pinMode(PNP, OUTPUT);
}

void loop() {
  //Envia um sinal de nível lógico alto aos transistores
  digitalWrite(NPN, HIGH);
  digitalWrite(PNP, HIGH);

  delay(1000);     // Aguarda 1 segundo

  // Envia um sinal de nível lógico baixo aos transistores
  digitalWrite(NPN, LOW);
  digitalWrite(PNP, LOW);

  delay(1000);     //Aguarda 1 segundo
}
```

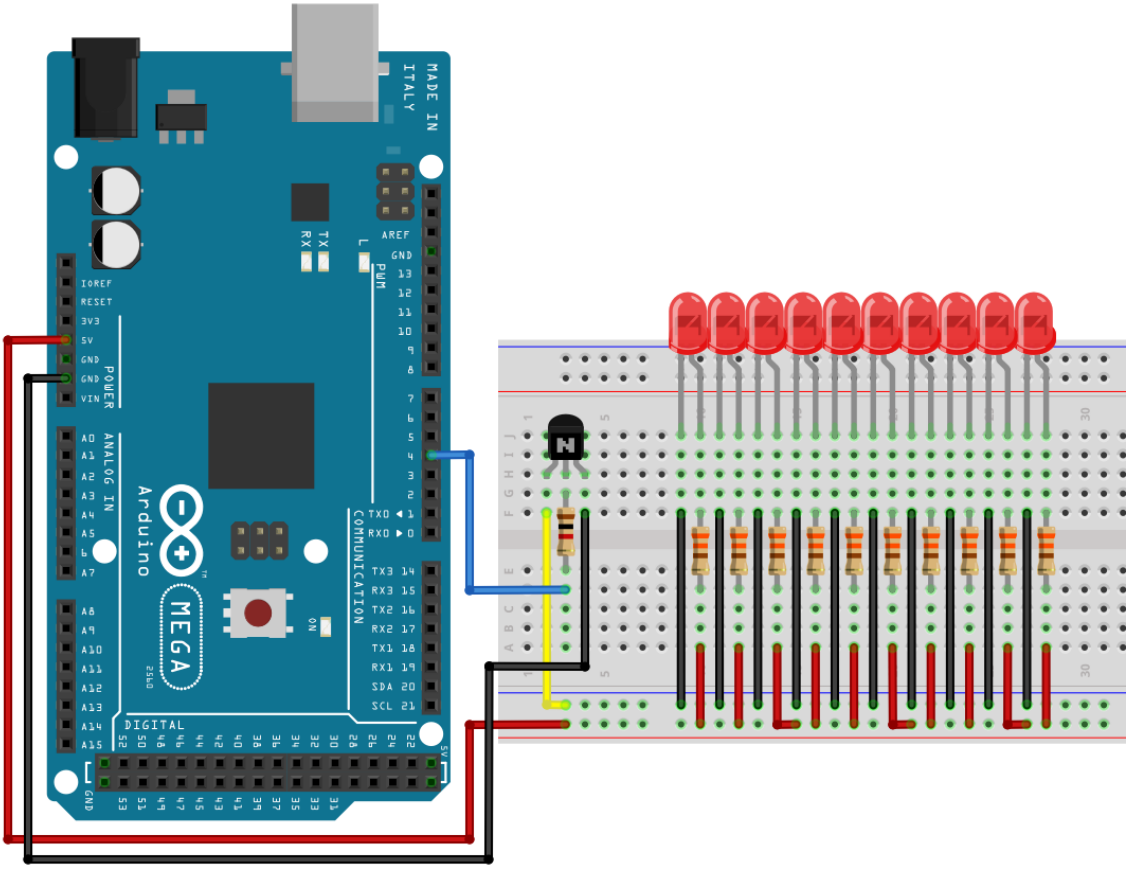
Exemplo 15 – Acionamento de cargas usando transistores

Nesse exemplo, vamos abordar o uso dos transistores como drivers, utilizando o BC548. A principal vantagem dessa configuração é permitir ao Arduino controlar cargas de maior corrente sem sobrecarregar seus pinos digitais (que fornecem somente 40mA cada). Para ilustrar isso, vamos usar o transistor para acionar 10 LEDs ao mesmo tempo, totalizando cerca de 75mA (conforme nossas medições).

Lista de materiais:

- Arduino Mega;
- Protoboard;
- 10 LEDs;
- 10 Resistores de 330Ω;
- 1 Resistor de 10KΩ;
- Transistor BC548;
- Jumpers.

Diagrama de circuito:



Carregue o código abaixo em seu Arduino:

```
//Exemplo 15 - Acionamento de cargas usando transistores
//Apostila Eletrogate - KIT ADVANCED

//Definição dos pinos
#define NPN 4

void setup() {
  //Configura os pinos como saída
  pinMode(NPN, OUTPUT);
}

void loop() {
  //Envia um sinal de nível lógico alto aos transistores
  digitalWrite(NPN, HIGH);
  delay(1000);     // Aguarda 1 segundo
  // Envia um sinal de nível lógico baixo aos transistores
  digitalWrite(NPN, LOW);
  delay(1000);     //Aguarda 1 segundo
}
```

Por ser uma pessoa interessada em aprender, você pode estar se perguntando:

“Mas como chegaram a esses valores para o resistor de base nos exemplos 11 e 12?”

Ao final da apostila, deixamos uma explicação completa de como calcular esse resistor, juntamente com exemplos de cálculo para que você aprenda a dimensioná-lo corretamente.

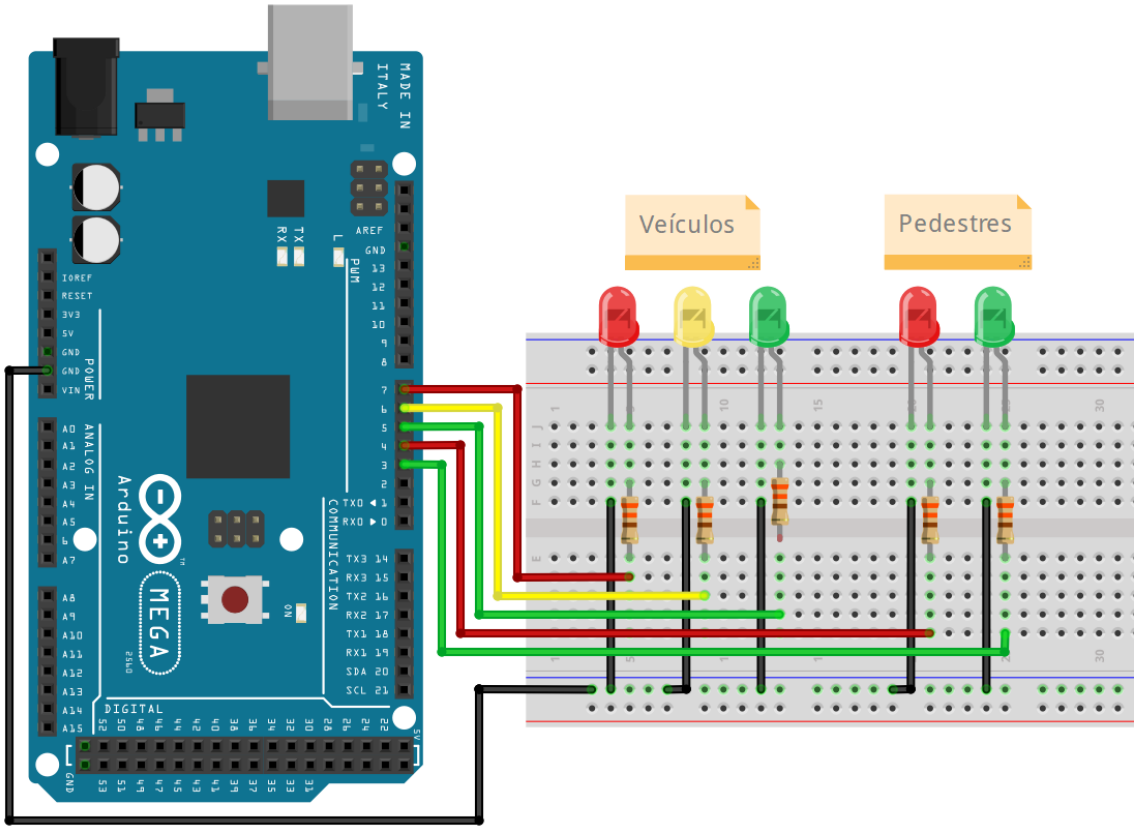
Exemplo 16 – Semáforo com Arduino

Nesse projeto, vamos definir as regras que queremos que nosso código siga. Dessa forma, você pode associar o que a descrição da regra pede que seja feito com os comandos, ajudando na compreensão do funcionamento do código. Acompanhe:

Lista de materiais:

- Arduino Mega;
- Protoboard;
- 2 LEDs vermelhos;
- 2 LEDs verdes;
- 1 LED amarelo;
- 5 Resistores de 330Ω;
- Jumpers.

Diagrama de circuito



O código deverá satisfazer as seguintes situações:

1- O semáforo de veículos deve permanecer verde por 8s, desligando em seguida;
2- O LED amarelo acende por 3s, apaga e depois o LED vermelho dos veículos e o LED verde dos pedestres acendem;
3- O semáforo de pedestres fica aberto por 7s;
4- Após 4s, o LED vermelho pisca e permanece aceso enquanto o semáforo de veículos está verde.

O resultado é o código abaixo. Carregue-o em seu Arduino e confirme o funcionamento descrito.

```
//Exemplo 16 - Semáforo com Arduino
// Apostila Eletrogate - KIT ADVANCED

//Definições dos pinos onde os LEDs dos pedestres e veículos estão
conectados
#define VERDE_PED 3
#define VERMELHO_PED 4
#define VERDE_VEI 5
#define AMARELO_VEI 6
#define VERMELHO_VEI 7

void setup() {
 pinMode(VERDE_PED, OUTPUT);
 pinMode(VERMELHO_PED, OUTPUT);
 pinMode(VERDE_VEI, OUTPUT);
 pinMode(AMARELO_VEI, OUTPUT);
 pinMode(VERMELHO_VEI, OUTPUT);
}

void loop() {
 digitalWrite(VERMELHO_VEI, LOW);     //Desliga o LED vermelho dos veículos
 digitalWrite(VERDE_VEI, HIGH);       //Liga o LED verde dos veículos
 digitalWrite(VERMELHO_PED, HIGH);    //Liga o LED vermelho dos pedestres
 delay(8000);
 digitalWrite(VERDE_VEI, LOW);        //Desliga o LED verde dos veículos
 digitalWrite(AMARELO_VEI, HIGH);     //Liga o LED amarelo do veículos
 delay(3000);
 digitalWrite(AMARELO_VEI, LOW);      //Desliga o LED amarelo dos veículos
 digitalWrite(VERMELHO_PED, LOW);     //Desliga o LED vermelho dos pedestres
 digitalWrite(VERMELHO_VEI, HIGH);    //Liga o LED vermelho dos veículos
 digitalWrite(VERDE_PED, HIGH);       //Liga o LED verde dos pedestres
 delay(5000);
 digitalWrite(VERDE_PED, LOW);        //Desliga o LED verde dos pedestres

 //Laço para piscar o LED vermelho dos pedestres
 for (int i = 0; i < 4; i++) {
 digitalWrite(VERMELHO_PED, HIGH);
 delay(250);
 digitalWrite(VERMELHO_PED, LOW);
 delay(250);
 }
}
```

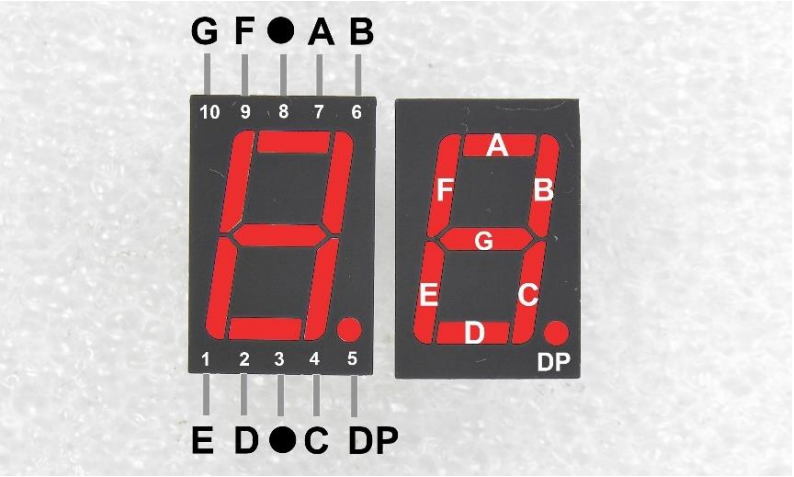
Exemplo 17- Acionando displays de 7 segmentos

Os displays de 7 segmentos surgiram a partir da evolução dos LEDs. Um LED é um diodo semicondutor que emite luz quando uma corrente elétrica o atravessa. Para evitar que ele queime, é essencial limitar a corrente, por isso um resistor em série deve ser utilizado ao energizá-lo.

Cada segmento do display é, na verdade, um LED individual. Ao acionar os segmentos, podemos formar números e letras (com algumas limitações). Os segmentos são identificados por letras de A a G, além do ponto decimal, rotulado como DP.

Existem dois tipos de displays: catodo comum e anodo comum. No display de catodo comum, os negativos de todos os segmentos estão conectados a um único pino. No display de anodo comum, os positivos são os pinos interligados entre si, resultando em um acionamento invertido em relação ao catodo comum.

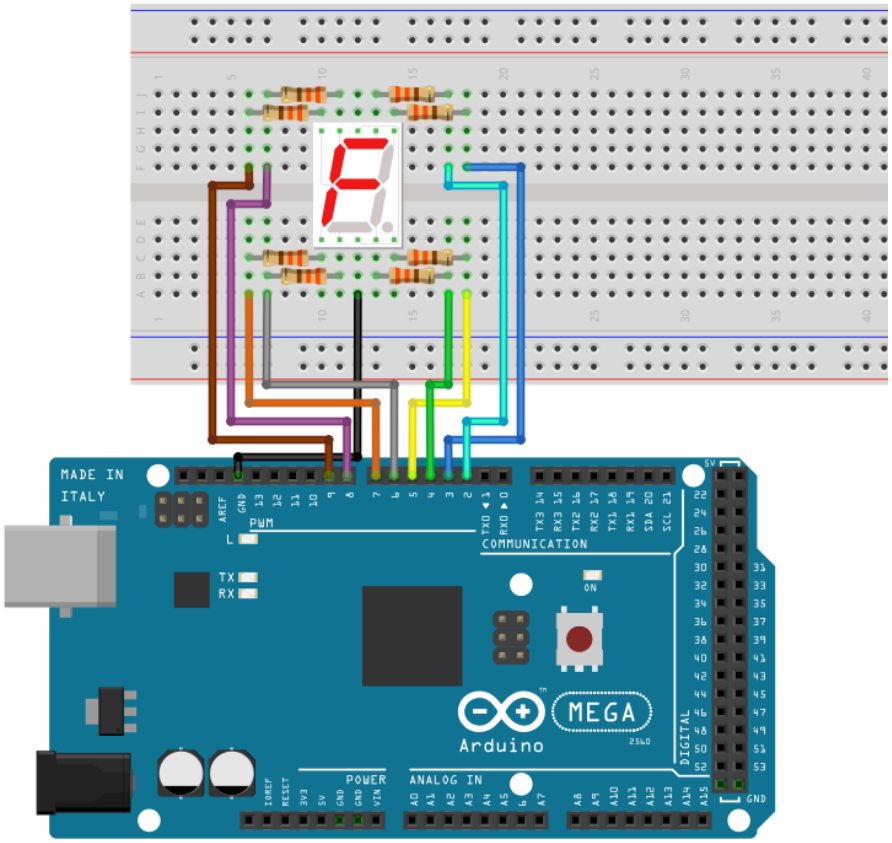
Neste projeto, utilizamos um display de 7 segmentos de catodo comum. Os pinos marcados com bolinhas pretas representam os pinos comuns dos catodos. Utilize apenas um desses pinos, pois ambos estão interligados internamente. Os demais devem ser conectados ao Arduino através de um resistor para limitar a corrente.



Lista de componentes

- Arduino;
- Protoboard;
- 8 resistores de 330Ω;
- Display de 7 segmentos;
- Jumpers.

Diagrama de montagem



Em nosso código, vamos explorar dois novos recursos da programação: arrays e funções. Começando com os arrays, para ilustrar, imagine uma gaveta que guarda um único parafuso – esta representa uma variável. Agora, pense em uma segunda gaveta com divisórias, onde cada compartimento contém um parafuso de tamanho diferente. Isso é o que chamamos de array.

Por exemplo, um array chamado *tamanhoParafuso[3]* pode armazenar três valores: *tamanhoParafuso[0] = 25, tamanhoParafuso[1] = 30, tamanhoParafuso[2] = 20*. Cada tamanho ocupa uma posição específica, chamada índice, que começa em 0, permitindo acesso individual sem afetar os demais.

Quanto às funções, elas nos permitem nomear um conjunto de instruções que podem ser reutilizadas em outras partes do código. Por exemplo, ao fazer um LED piscar com o Arduino, em vez de repetir os comandos de ligar, aguardar e desligar, você pode criar uma função chamada piscaLED(). Dentro dessa função, reunimos todos os comandos necessários para realizar a tarefa de piscar o LED. Assim, sempre que precisar que o LED pisque, em vez de repetir os quatro comandos, você simplesmente chama piscaLED(), trazendo organização, praticidade e legibilidade ao seu código, facilitando seu entendimento e manutenção.

Tendo isso em mente, carregue o código abaixo em seu Arduino:

```
//Exemplo 17 - Acionando displays de 7 segmentos
//Apostila Eletrogate - KIT ADVANCED

int pinos[8] = {2, 3, 4, 5, 6, 7, 8, 9};
int    N0[8] = {0, 0, 0, 1, 0, 0, 0, 1};
int    N1[8] = {1, 0, 0, 1, 1, 1, 1, 1};
int    N2[8] = {0, 0, 1, 1, 0, 0, 1, 0};
int    N3[8] = {0, 0, 0, 1, 0, 1, 1, 0};
int    N4[8] = {1, 0, 0, 1, 1, 1, 0, 0};
int    N5[8] = {0, 1, 0, 1, 0, 1, 0, 0};
int    N6[8] = {0, 1, 0, 1, 0, 0, 0, 0};
int    N7[8] = {0, 0, 0, 1, 1, 1, 1, 1};
int    N8[8] = {0, 0, 0, 1, 0, 0, 0, 0};
int    N9[8] = {0, 0, 0, 1, 0, 1, 0, 0};

void reset7Seg() {
  for (int i = 0; i < 8; i++) {
    digitalWrite(pinos[i], HIGH);
  }
}

void exibir(int caractere) {
  reset7Seg();  //Limpar o display
     switch (caractere) {
       case 0:
          for (int i = 0; i < 8; i++) {
            if (N0[i] == 1) {
              digitalWrite(pinos[i], HIGH);
            } else {
              digitalWrite(pinos[i], LOW);
            }
          }
       break;
       case 1:
          for (int i = 0; i < 8; i++) {
            if (N1[i] == 1) {
              digitalWrite(pinos[i], HIGH);
            } else {
              digitalWrite(pinos[i], LOW);
            }
          }
       break;
       case 2:
          for (int i = 0; i < 8; i++) {
            if (N2[i] == 1) {
              digitalWrite(pinos[i], HIGH);
            } else {
              digitalWrite(pinos[i], LOW);
            }
          }
       break;
    }
}

void setup() {
  for (int i = 0; i < 8; i++) {
    pinMode(pinos[i], OUTPUT);
   }
}

void loop() {
   for (int i = 0; i < 3; i++) { // Mude o limite para o número total de caracteres
configurados no switch + 1
     exibir(i);
     delay(1000); // Esperar 1 segundo entre os caracteres
   }
}
```

Seu funcionamento é bem simples: criamos um array com 8 posições para armazenar os pinos do Arduino onde os LEDs do display estão conectados. Também criamos arrays de mesmo tamanho contendo os pinos que devem estar ligados ou desligados para formar cada caractere. No setup, configuramos os pinos utilizando um laço for e o comando *pinMode(i, OUTPUT)*, que interpreta o valor em cada índice do array como sendo o número do pino e o configura como saída.

Para manter a organização e evitar repetições, implementamos as funções *reset7Seg()*, que desliga todos os LEDs antes de exibir um novo caractere, e *exibir()*, que faz o acionamento do display. Para isso, informamos como parâmetro um número que pode ir de 0 até a quantidade de arrays de caracteres, utilizando o comando switch para atribuir cada número à leitura do respectivo array (veja a parte V, onde explicamos cada comando do Arduino). No loop, criamos um laço for que conta de zero até 2, fornecendo o parâmetro necessário para a função exibir() e definindo um tempo de 1 segundo entre a exibição de cada caractere.

Na prática, nosso código conta de 0 a 2, somente, mas isso tem um objetivo: desafiarmos **você** a implementar a contagem de 0 a 9 e criar arrays para exibir também as letras “a”, “b”, “c”, “d”, “e”, “f”, “g”, “h”, “L”, “p” e “u”, que são as possíveis de serem formadas/exibidas num display de 7 segmentos.

Fazendo isso, você colocará à prova todo o seu aprendizado até aqui ao mesmo tempo que o coloca em prática, desenvolvendo habilidades essenciais para a programação do Arduino.

Exemplo 18 - Cronômetro Digital com Display LCD

O display 16×2 é um dispositivo que possui interface para comunicação e controle padronizada. Esses displays são fáceis de serem controlados e graças à essa padronização é possível trocar um display 16×2 de um fabricante por outro diferente sem maiores problemas.

A tecnologia utilizada nos displays tradicionais é LCD (Liquid Crystal Display). Mais recentemente, modelos mais modernos com tecnologia O-Leds , os LEDS orgânicos, já estão sendo encontrados também.

A grande vantagem e razão pela qual até hoje é componente popular, é o seu preço e a facilidade de implementação em projetos eletrônicos. Antes dos displays LCD, a interface gráfica usada para mostrar caracteres alfa numéricos era os displays a LED de x segmentos (os mais comuns eram de 7, 14 ou 16 segmentos).

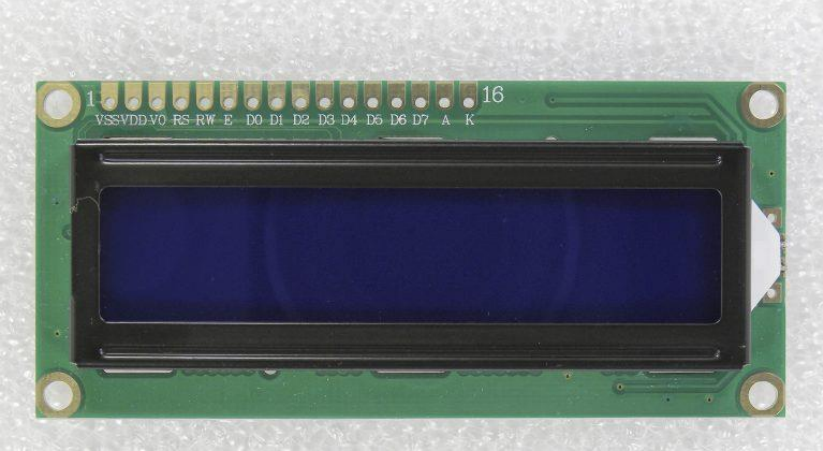
Esses componentes mais antigos utilizavam vários leds (7,14 ou 16) dispostos em segmentos, de forma que dependendo de qual conjunto de leds fosse aceso, um determinado caractere seria mostrado.

LCD 16x2. Créditos: Eletrogate

Na hora de comprar um display LCD, você vai encontrar diversas opções, além do mais tradicional 16×2. A especificação é dada em relação ao número de caracteres por linha e o número de linhas. Assim, 16×2 significa que o display pode exibir 16 caracteres em cada linha, e possui 2 linhas. Alguns valores típicos para essas especificações são:

- Quantidade de caracteres padrão de mercado: 8, 12, 16, 20, 24 e 40;
- Quantidade de linhas mais comuns: 2 e 4;
- Cores de fundo (backlight): Azul ou verde;

O display que vamos trabalhar é de 16 colunas e 2 linhas (16×2), backlight (luz de fundo) azul e caracteres brancos. A interface com Arduino é feita por meio de um barramento de 16 pinos. Em geral apenas 12 são utilizados de fato. Os pinos do LCD são:



LCD 16x2 pinagem – foto Gustavo Murta

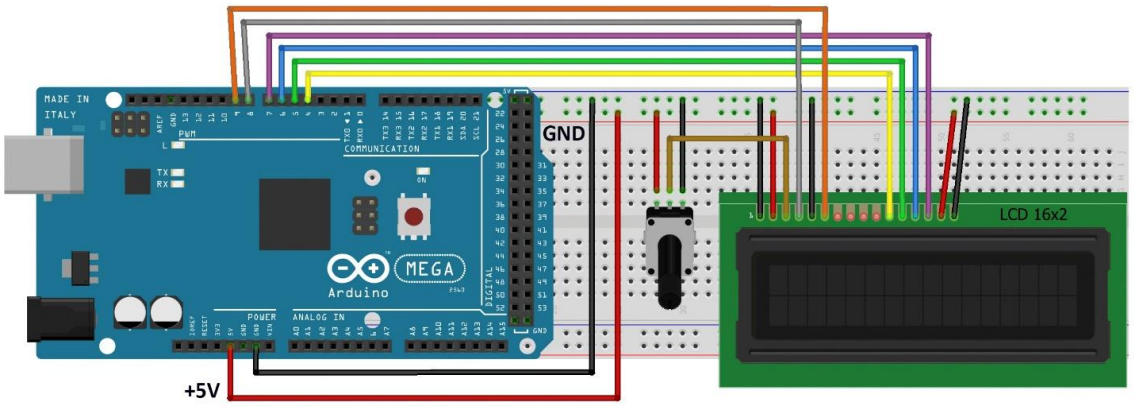
- Pino de **register select (RS):** Controle em que parte da memória (do LCD) o usuário está escrevendo os dados. São duas opções, você pode escrever um dado para ser mostrado no display, ou uma instrução no MCU do display;
- Pino de **Read/Write (R/W**): Seleciona se está em modo de leitura ou de escrita;
- Pino de **Enable**: Habilita a escrita de dados nos registradores;
- 8 pinos de dados **(D0 -D7).** Os estados de cada pino desses corresponde aos bits que você está enviando para display ou os valores lidos, quando está no modo de leitura.
- Pinos de alimentação **VCC e GND** do LCD e do LED Backlight (A-anodo/K-catodo);
- Pino **V0** de ajuste de contraste;

Vamos conferir em nosso exemplo como usar o LCD 16x2:

Lista de materiais:

- 1 Arduino Mega;
- 1 Protoboard;
- Jumpers para ligação no protoboard;
- Display LCD 16x2;
- Potenciômetro 10K;

Diagrama de circuito:



No diagrama acima, o potenciômetro é usado para controlar o contraste do LCD, e deve ser ajustado sempre que você ligar o display para poder visualizar bem cada caractere. As demais ligações são padrão.

No código abaixo, os pinos do Arduino são configurados de acordo com a ligação acima.

```
// Exemplo 18 - Display LCD 16x2
// Apostila Eletrogate - KIT ADVANCED
#include <LiquidCrystal.h>              // biblioteca Liquid Crystal
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);    // pinos do LCD - RS E D4 D5 D6 D7
int contador = 0;                       // variável contador
void setup(){
  lcd.begin(16, 2);                     // inicializa LCD 16x2
  delay(500);                           // atraso de 0,5 segundos
}

void loop(){
  lcd.clear();                          // limpa tela do LCD
  lcd.setCursor(0, 0);                  // selecionando coluna 0 e linha 0
  lcd.print("Exemplo LCD !");           // mostra no LCD
  lcd.setCursor(1, 1);                  // selecionando coluna 1 e linha 1
  lcd.print(contador);                  // mostra no LCD a contagem
  contador++;                           // incrementa contador
  if (contador == 60)                   // se contador = 60
    contador = 0;                       // zera o contador
  delay(1000);                          // atraso de 1 segundo
}
```

No código, é mostrado na primeira linha (0) do display, a mensagem “Exemplo LCD !“. Na segunda linha (1) será mostrado um contador que é incrementado de 1 em 1 segundo. Para usar o LCD é preciso incluir a biblioteca **LiquidCrystal.h**, que é nativa da IDE arduino. Essa biblioteca possui as seguintes funções:

- lcd.clear() - Limpa a tela do display
- lcd.SetCursor(0,0) - Posiciona o cursor a partir de onde os caracteres serão escritos
- lcd.print() - Imprime caracteres no display

Exemplo 19 - Trena Eletrônica com Sensor Ultrassônico

O princípio de funcionamento do Sensor HC-SR04 consiste na emissão de sinais ultrassônicos e na leitura do sinal de retorno (reflexo/eco) desse mesmo sinal. A distância entre o sensor e o objeto que refletiu o sinal é calculada com base no tempo entre o envio e leitura de retorno.

“Sinais Ultrassônicos são ondas mecânicas (ondas de som) com frequência acima de 40 KHz, imperceptíveis ao ouvido humano.“

Como ouvido humano só consegue identificar ondas mecânicas até a frequência de 20KHz, os sinais emitidos pelo sensor Ultrassônico não podem ser escutados por nós.

O sensor HC-SR04 é composto de três partes principais:

- Transmissor Ultrassônico – Emite as ondas ultrassônicas que serão refletidas pelos obstáculos;
- Um receptor – Identifica o eco do sinal emitido pelo transmissor;
- Circuito de controle – Controla o conjunto transmissor/receptor, calcula o tempo entre a emissão e recepção do sinal;

Seu funcionamento consiste em três etapas:

1. O circuito externo (Arduino) envia um pulso de trigger de pelo menos 10us em nível alto;
2. Ao receber o sinal de trigger, o sensor envia 8 pulsos de 40khz e detecta se há algum sinal de retorno ou não;
3. Se algum sinal de retorno for identificado pelo receptor, o sensor gera um sinal de nível alto no pino de saída cujo tempo de duração é igual ao tempo calculado entre o envio e o retorno do sinal ultrassônico;

Por meio desse pulso de saída cujo tempo é igual a duração entre emissão e recepção, nós calculamos a distância entre o sensor e o objeto/obstáculo, por meio da seguinte equação:

$$Distancia = \frac{(Tempo\_de\_duração\_do\_sinal\_de\_saída * velocidade\_do\_som)}{2}$$

Nela, o tempo de duração do sinal de saída é o tempo no qual o sinal de output permanece em nível alto e a velocidade do som = 340 m/s;

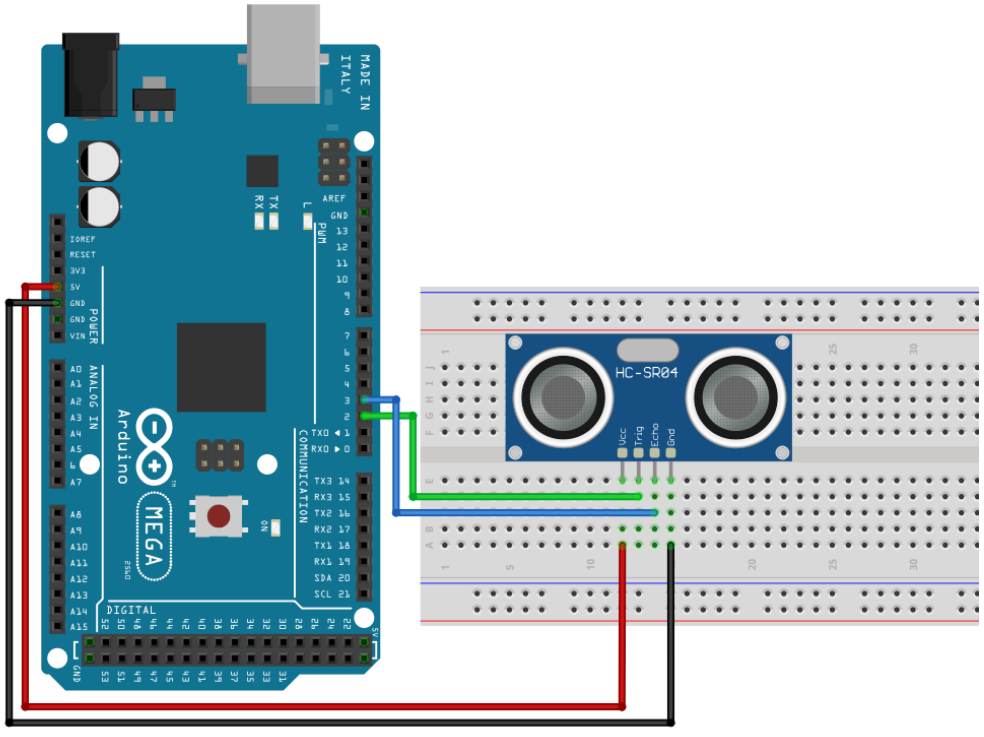
Repare que as unidades devem estar coerentes para o resultado da conta ser correto. Assim, se a velocidade do som é dada em metros/segundo, o tempo de duração do sinal de saída deve estar em segundos para que possamos encontrar a distância em metros.

Lista de Materiais:

Para este exemplo você vai usar os seguintes componentes:

- Protoboard;
- Jumpers de ligação;
- Sensor ultrassônico HCSR-04.

Diagrama de circuito:



Copie e cole o código a seguir em sua IDE:

```
// Exemplo 19 - Sensor de Distância Ultrassônico HC-SR04
// Apostila Eletrogate - KIT ADVANCED

int PinTrigger = 2;                         // pino usado para disparar os pulsos do sensor
int PinEcho = 3;                            // pino usado para ler a saida do sensor
float TempoEcho = 0;                        // variável tempo do eco
const float velocidadeSom_mps = 340;        // em metros por segundo
const float velocidadeSom_mpus = 0.000340;  // em metros por microsegundo

void setup(){
  pinMode(PinTrigger, OUTPUT);              // configura pino Trigger como saída
  digitalWrite(PinTrigger, LOW);            // pino trigger - nível baixo
  pinMode(PinEcho, INPUT);                  // configura pino ECHO como entrada
  Serial.begin(9600);                       // inicializa monitor serial 9600 Bps
  delay(100);                               // atraso de 100 milisegundos
}

void loop(){
  DisparaPulsoUltrassonico();                       // dispara pulso ultrassonico
  TempoEcho = pulseIn(PinEcho, HIGH);               // mede duração do pulso HIGH de eco em
microsegundos
  Serial.print("Distancia em metros: ");            // mostra no monitor serial
  Serial.println(CalculaDistancia(TempoEcho));      // mostra o calculo de distancia em metros
  Serial.print("Distancia em centimentros: ");      // mostra no monitor serial
  Serial.println(CalculaDistancia(TempoEcho) * 100); // mostra o calculo de distancia em cm
  delay(2000);                                      // atraso de 2 segundos
}
void DisparaPulsoUltrassonico(){
  digitalWrite(PinTrigger, HIGH);                   // pulso alto de Trigger
  delayMicroseconds(10);                            // atraso de 10 milissegundos
  digitalWrite(PinTrigger, LOW);                    // pulso baixo de Trigger
}

float CalculaDistancia(float tempo_us){
  return ((tempo_us * velocidadeSom_mpus) / 2 );    // calcula distancia em metros
}
```

Explicando, as variáveis declaradas são para determinar os pinos de trigger (pino 2) e de leitura do sensor (pino 3). Temos três variáveis do tipo float utilizadas para medir o tempo do pulso no output do sensor e calcular a distância. Na função void setup(), inicializamos o pino 2 como saída e o 3 com entrada. Além disso, configuramos a comunicação serial para 9600 bits/segundo.

Na função void loop(), onde o programa será executado continuamente, executa-se três passos básicos:

1. Enviar pulso de 10us (microssegundos) para o pino de trigger do sensor. Isto é feito com a função DisparaPulsoUltrassonico();

2. Ler o pino de saída do sensor e medir o tempo de duração do pulso utilizando a função pulseIn. Esta função retorna o tempo de duração do pulso em microssegundos para que ele seja armazenado em uma variável;

3. Por fim, chamamos a função CalculaDistancia (tempo) passando como parâmetro o tempo lido com a função pulseIn. Esta função calcula a distância entre o sensor e o obstáculo. Nós chamamos esta função de dentro da função Serial.println(), de forma que o resultado já é impresso diretamente no terminal serial;

A função **DisparaPulsoUltrassonico()** apenas ativa o pino 2 em nível alto, espera 10 microsegundos e volta a setar o pino para nível baixo. Lembre-se que 10 us é o tempo mínimo que o pulso deve perdurar para disparar o sensor HC-SR04.

A função **CalculaDistancia()** recebe como parâmetro o tempo do pulso no pino Echo do sensor. Com esse tempo nós usamos a equação, para calcular a distância entre o sensor e o obstáculo.

Exemplo 20 – Controlando um Motor de Passo com Potenciômetro

O Motor de passo é um motor elétrico que não possui escovas ou comutadores, permitindo assim uma vida longa sem tantos desgastes. O rotor (parte móvel) constitui-se de um ou mais imãs permanentes. No estator (parte fixa) encontram-se duas ou mais bobinas, dependendo do tipo de motor. O controle do motor é feito por um circuito eletrônico que aciona repetidamente as bobinas numa sequência que permite o giro do rotor. Cada pulso enviado para o circuito, faz com que o motor avance um passo. O sentido de rotação do motor é controlado também pela sequência e pela polarização

das bobinas. A velocidade que o rotor gira é determinada pela frequência dos pulsos do circuito de controle. Quanto maior a frequência, maior será o RPM.

Informações sobre o Motor de Passo 28BYJ-48:

Esse motor é Unipolar pois possui 4 enrolamentos que chamamos de fases. Em uma das pontas das fases, todas estão conectadas juntas (fio comum). Portanto, esse

motor não pode ser usado em Drivers para motores Bipolares (com duas fases somente). Interessante, que ele pode ser alimentado com 5V e consome baixa corrente, o que facilita a montagem.

Algumas características do motor:

- Tensão de operação : 5V CC
- Número de fases : 4
- Razão da variação de velocidade : 1/64 (mecanismo de redução)
- Angulo do passo : 5,625 graus => 64 passos/volta (360/64=5,625)
- Resistência CC : 23 ohms
- Frequência : 100 Hz
- Torque de tração: > 34,3 mN.m

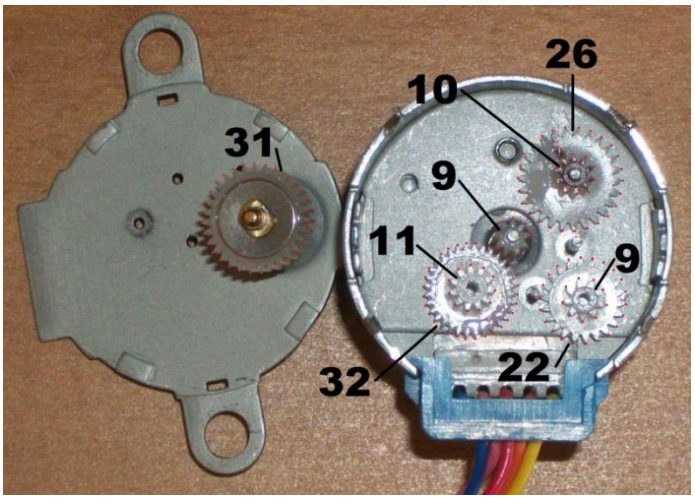
OBS: a resistência medida nas bobinas do motor foi de 23 ohms. Assim podemos deduzir o consumo estático de corrente em cada bobina :

I = V / R -> 5V / 23 ohms -> 217 mA

Nesse tipo de motor, cada fase é energizada por um circuito driver num único sentido de corrente. Isto é, uma extremidade do enrolamento será sempre positiva e a outra sempre negativa. A desvantagem do motor unipolar é que tem menos torque do que o bipolar similar, pois sempre terá no máximo, a metade das fases energizadas. Daí pode-se concluir que tem 50% da eficiência em relação ao bipolar.

Esse motor tem uma caixa de engrenagens para redução da rotação do eixo, mas que permite um aumento no torque. Veja abaixo a caixa de engrenagens desmontada. A relação de redução é de 63,68:1 , isto é, para cada giro do eixo de saída, são necessárias 63,68 voltas do eixo interno do motor. Como esse motor de passo precisa

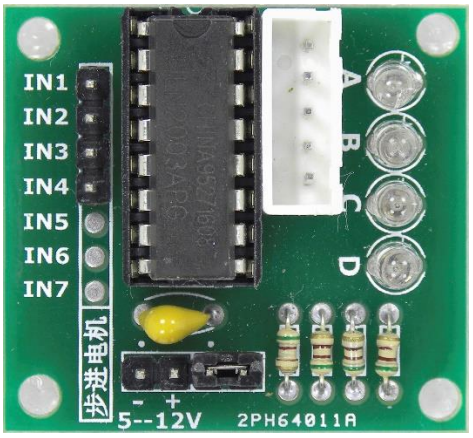
de 64 passos para dar uma volta, são necessários 64 x 63,68 = 4075 passos (pulsos) aproximadamente para um giro do eixo de saída.



Informações sobre o Módulo Driver ULN2003:

O único chip no módulo é o ULN2003. Esse chip possui um conjunto de sete drivers de transistores Darlington que permitem o acionamento de cargas indutivas. Todas as saídas tem o coletor aberto e diodos de supressão (Clamp) . Os transistores suportam tensões de até 50V e correntes de até 500 mA. Todas as entradas IN1, IN2,IN3 e IN4 são compatíveis com sinais limitados a tensões de até 5V.

Pinos do Módulo ULN2003 :



O pino mais à esquerda (-) no módulo é o terra (veja foto acima). Esse terra tem que ser conectado ao terra da fonte e ao terra do Arduino. O pino (+) no caso do Motor 28BYJ-48, tem que ser conectado ao positivo de uma fonte de 5V (preferencialmente de 1 Ampére). Não recomendo que conecte no 5V do Arduino, pois poderá sobrecarregar o

regulador de tensão do mesmo. O jumper Power ON/OFF é usado para ligar ou desligar o motor. Os quatro leds vermelhos (A,B,C e D) são usados para indicar o acionamento de cada um dos drivers (fases do motor).

IMPORTANTE: Muita atenção ao conectar a alimentação nos pinos. Uma ligação incorreta poderá danificar o driver ou o motor.

Modos de operação:

Para um motor de Passo Unipolar, temos alguns modos de operação. Temos o **modo Passo completo com alto torque** (Full step), o **modo Passo completo com baixo torque** (Wave Step), o **modo Meio Passo** (half step) e **Micro-passo** (Micro stepping).

No caso do Micro passo, a corrente nos enrolamentos deve ser alterada. Como o nosso circuito não permite o controle da corrente, esse modo de operação não se aplica para esse driver.

Passo completo com alto torque (Full step): - Duas Fases são acionadas ao mesmo tempo, nessa ordem:

Step	ϕ4	ϕ3	ϕ2	ϕ1
0	0	0	1	1
1	0	1	1	0
2	1	1	0	0
3	1	0	0	1

Passo completo com baixo torque (Wave Step): - Somente uma Fase acionada de cada vez, nessa ordem:

Step	ϕ4	ϕ3	ϕ2	ϕ1
0	0	0	0	1
1	0	0	1	0
2	0	1	0	0
3	1	0	0	0

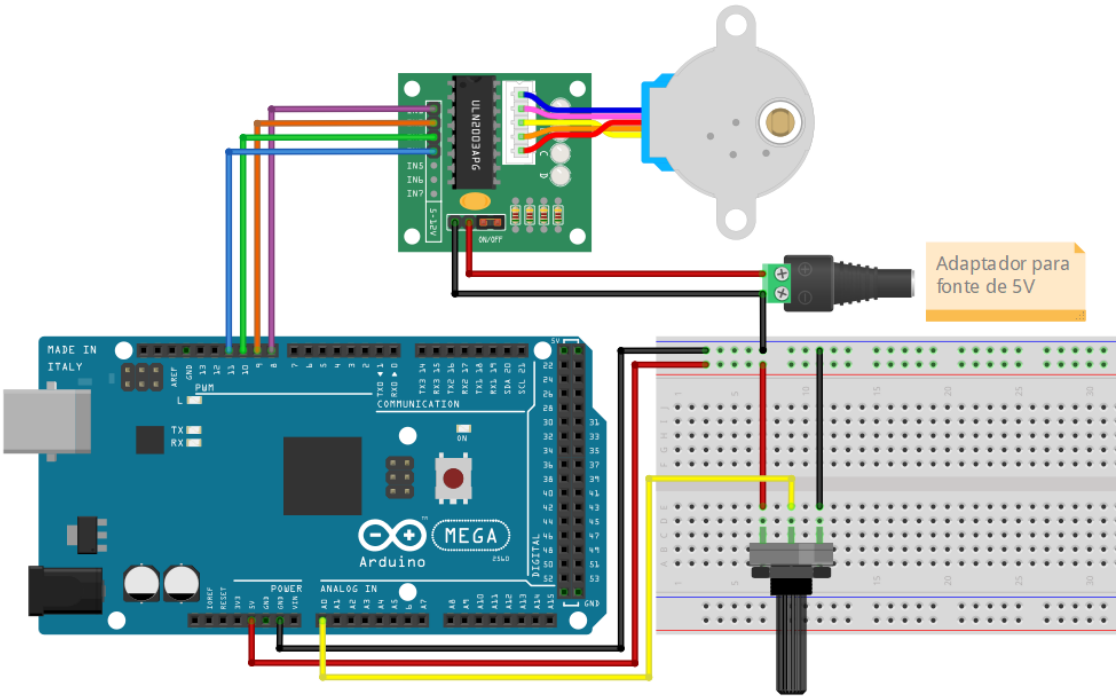
Meio Passo (half step): - Na sequência de oito passos, em alguns passos temos somente uma fase acionada e em outros passos, temos duas fases acionadas, nessa ordem:

Step	ϕ4	ϕ3	ϕ2	ϕ1
0	0	0	0	1
1	0	0	1	1
2	0	0	1	0
3	0	1	1	0
4	0	1	0	0
5	1	1	0	0
6	1	0	0	0
7	1	0	0	1

Lista de materiais:

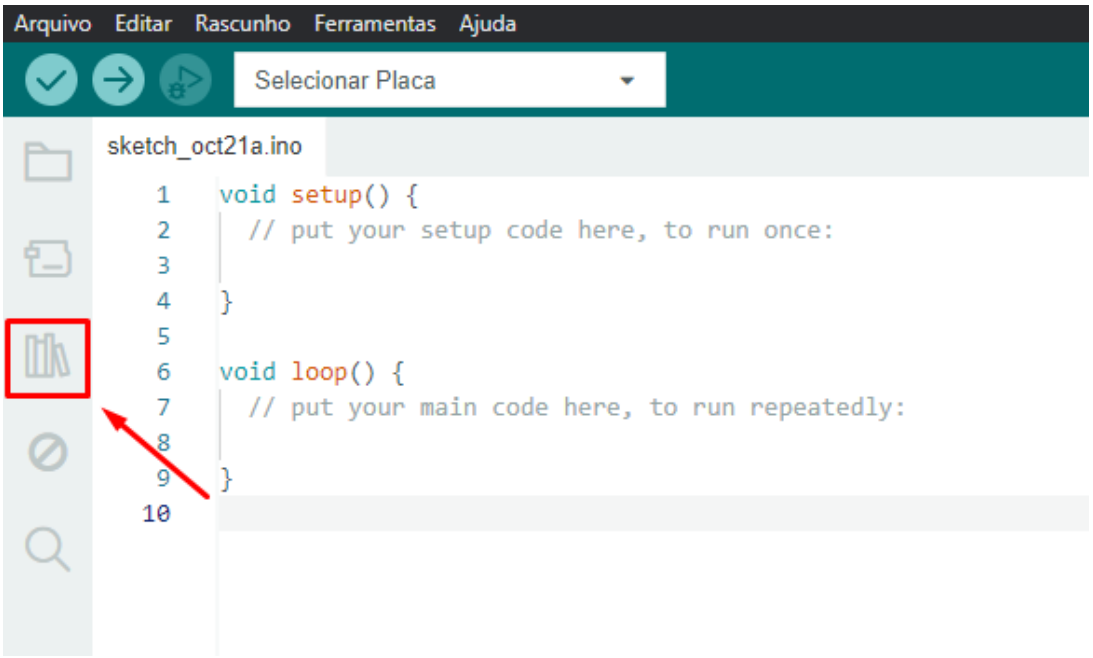
- Motor de passo 28BYJ-48 + Driver ULN2003;
- Arduino Mega R3 + Cabo;
- Protoboard;
- Potenciômetro;
- Adaptador para fonte;
- Jumpers.

Diagrama do circuito :

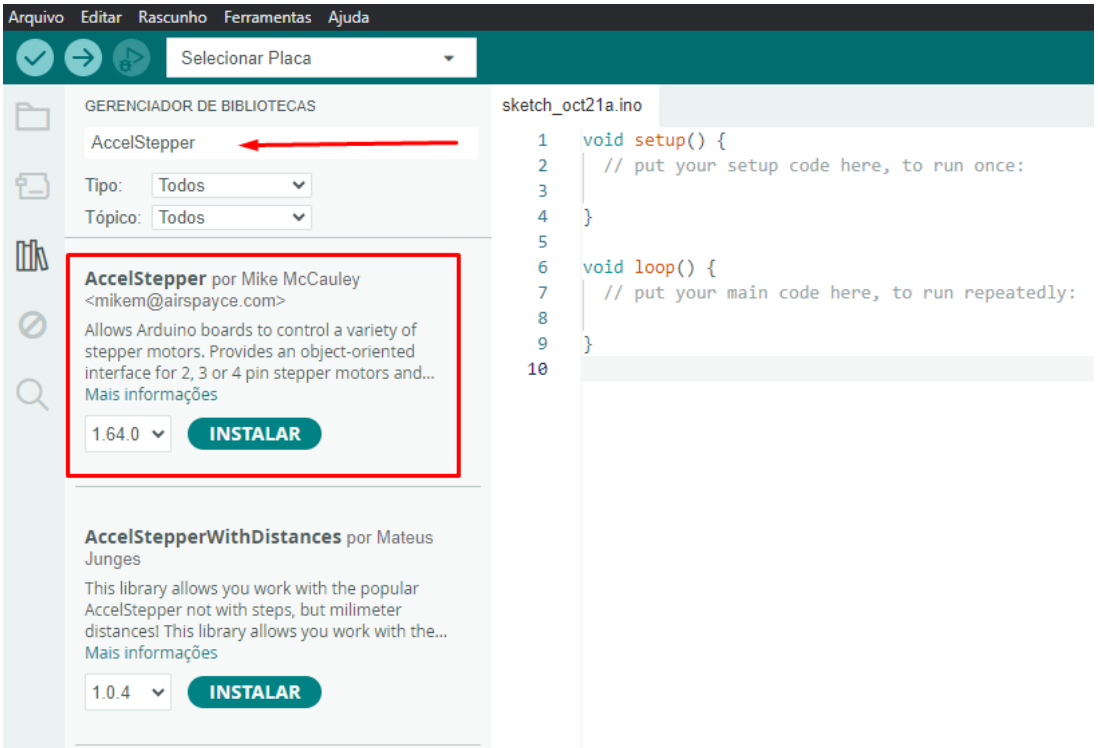


Para esse projeto, nosso código fará uso de uma biblioteca chamada **AccelStepper.** Diferente das outras que utilizamos ao longo dessa apostila, a AccelStepper não é uma biblioteca nativa da Arduino IDE, portanto, precisamos instalá-la manualmente. Para isso, siga os passos abaixo:

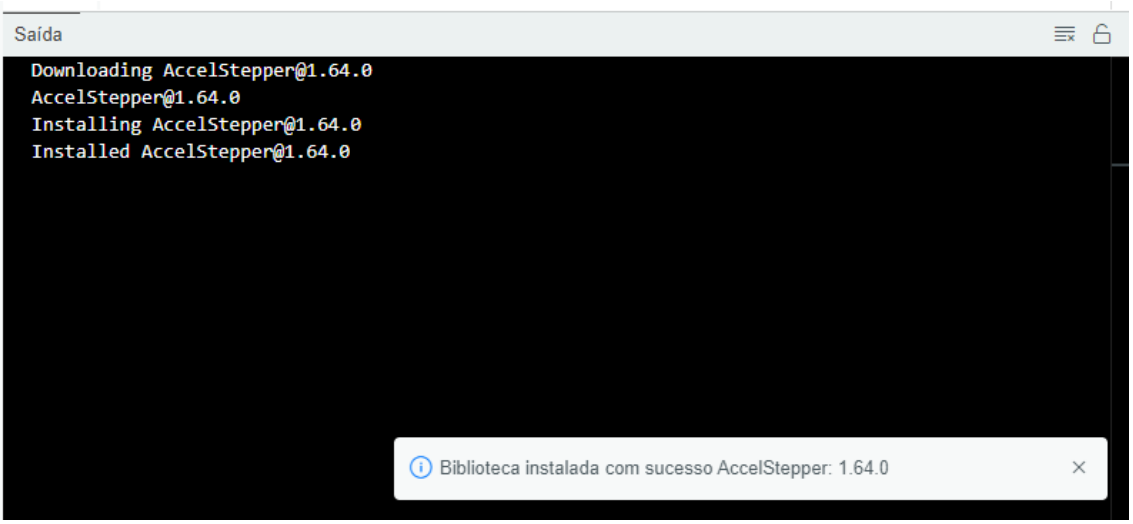
1º - Abra o Gerenciador de Bibliotecas da Arduino IDE clicando no botão mostrado abaixo:



2º - Com o gerenciador aberto, digite o nome da biblioteca e tecle Enter:



3º - Vamos instalar a versão destacada acima, desenvolvida por Mike McCauley. Basta clicar em “Instalar” e aguardar a mensagem abaixo:



Com a biblioteca instalada e o circuito montado, basta carregar o código abaixo em seu Arduino e girar o potenciômetro para observar o funcionamento do motor:

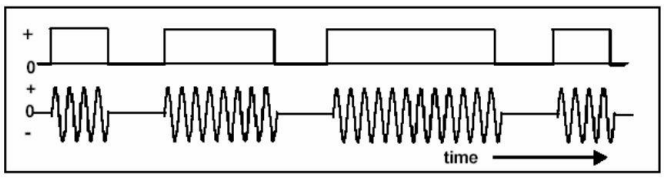
Esse é o Sketch da aplicação:

```
// Exemplo 20 - Motor de passo unipolar 28BYJ-48 + Driver ULN2003
// Apostila Eletrogate - KIT ADVANCED
// Baseado em AccelStepper/ProportionalControl 8pde-example


#include <AccelStepper.h>          // biblioteca AccelStepper

#define ANALOG_PIN A0              // pino A0 para leitura da tensão do Potenciometro

AccelStepper motorPasso (AccelStepper::FULL4WIRE, 8, 10, 9, 11);  // Passo completo

void setup()
{
  motorPasso.setMaxSpeed(500);              // maxima velocidade = 500 pulsos/seg
}

void loop()
{
  int analog_in = analogRead(ANALOG_PIN);   // lendo a tensão do pino A0 do Arduino
  motorPasso.moveTo(analog_in);             // gira o eixo do motor X pulsos
  motorPasso.setSpeed(100);                 // velocidade = 100 pulsos por segundo
  motorPasso.runSpeedToPosition();          // gira o eixo para a posição definida
}
```

Exemplo 21 – Comunicação sem Fio com Transmissor e Receptor RF

Esse conjunto de Módulos de transmissão(TX) e recepção(RX) de RF 433 MHz serve para o envio de dados digitais entre dois Arduinos (ou outro tipo de Microcontrolador). A comunicação é unidirecional , isto é, os dados são enviados pelo transmissor e recebidos pelo receptor.

A tipo de modulação da portadora de Rádio Frequência é o **ASK (amplitude shift keying)** - modulação por chaveamento de amplitude. Isto é, quando existe o Bit 1 a portadora transmite o sinal de 433 MHz. Quando o Bit é zero, nenhum sinal é transmitido.

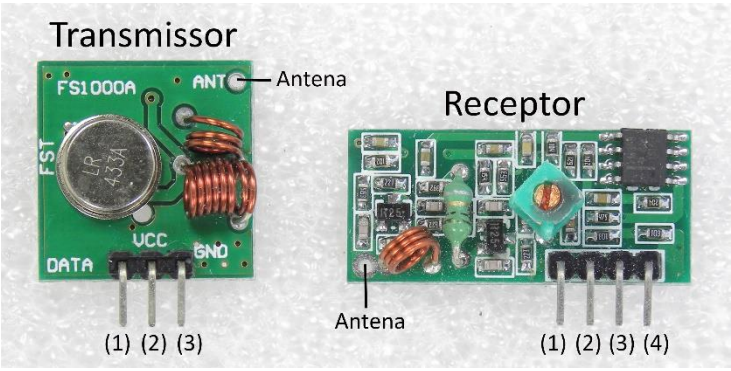


Referência : *http://blog.eletrogate.com/wp-content/uploads/2018/09/ASKnFSK.pdf*

Essas são algumas aplicações para esses módulos RF 433 MHz :

- transmissão de dados à curta distância,
- controle remoto,
- automação residencial,
- sistema de segurança - Alarme,
- Registro remoto de dados (Log).

Pinagem dos Módulos TX /RX de 433 MHz :



Modulo Transmissor - pinagem:

1. Data - pino de transmissão de dados

2. VCC - pino de alimentação : 3 a 12 V

3. GND - terra

Módulo Receptor - pinagem :

1. VCC - pino de alimentação : 5V

2. Data - pino de recepção de dados

3. Data - idem - conectado ao pino 2

4. GND – terra

Especificações dos Módulos TX /RX de 433 MHz :

Módulo Transmissor 433 MHz (FS1000A) :

Corrente de operação : 20 a 28 mA

Frequencia de operação : 433,92 MHz

Potência de saída: 10mW

Taxa de transferência < 4Kbps

Tensão de operação : 3 a 12 V

Alcance da transmissão < 100 m (com antena, sem obstáculos)

Antena externa : fio com 17,3 cm

Módulo Receptor 433 MHz (XY-MK-5V) :

Tensão de operação : 5 V (somente)

Frequencia de operação : 433,92 MHz

Corrente de operação : 4 mA

Sensibilidade de recepção : -105DB

Antena externa : fio com 17,3 cm

Sobre as Antenas dos Módulos TX e RX :

Para um perfeito funcionamento dos módulos de RF 433 MHz, é necessário a instalação das antenas tanto no transmissor quanto no receptor. Sabendo-se que a frequência da portadora é de 433,92 MHz e a velocidade da onda eletromagnética no espaço é de 3 x 10^8 m/s , o comprimento de onda é :

$$\lambda = v/f = 300.000.000 / 433.920.000 = 0,69 \text{ metros}$$

Usando uma antena com 1/4 do comprimento de onda :

$$D = 0,69 \text{ m} / 4 = 17,28 \text{ cm}$$

Portanto solde fios rígidos de aproximadamente 17,3 cm nos locais indicados na foto dos módulos. Uma antena para o transmissor e outra para o receptor, ambas com o mesmo tamanho.

Transmitindo dados entre dois Arduinos :

Os primeiros tutoriais que descobri na WEB sobre o uso desses módulos TX/RX RF 433 MHz , usavam a biblioteca **VirtualWire**. Mas no site do idealizador dessa Biblioteca, ele indica outra biblioteca mais recente - a **RadioHead,** que permite o uso de vários tipos de módulos de comunicação.

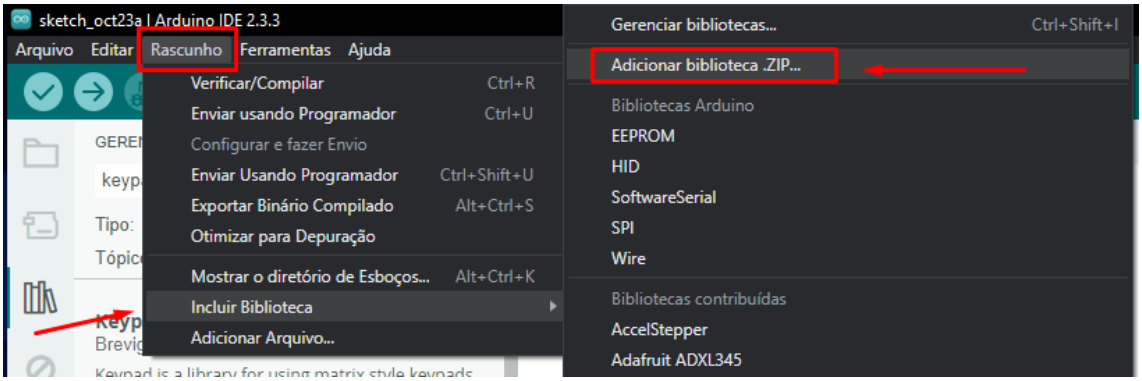
https://github.com/adafruit/RadioHead

https://github.com/adafruit/RadioHead

Para esse projeto, faremos uso de uma biblioteca chamada RadioHead, que será responsável por simplificar o envio dos dados. Como ela não foi publicada na Arduino IDE, precisamos fazer a instalação através de um arquivo compactado (.zip). Comece baixando-a através do link abaixo:

http://blog.eletrogate.com/wp-content/uploads/2018/09/RadioHead-master.zip

Em seguida, na Arduino IDE, clique em **Sketch > Incluir Biblioteca > Adicionar biblioteca.zip** e localize o arquivo RadioHead-master.zip que você acabou de baixar, clicando em abrir na sequência.



Bem interessante essa biblioteca, e funcionou perfeitamente nos meus testes. Ela poderá ser adaptada para a sua aplicação, enviando dados entre dois Arduinos. Usando os módulos com antenas, o alcance poderá ser expressivo. Lembrando que quanto maior a tensão de alimentação do módulo transmissor, maior será a potência de RF a ser transmitida e portanto maior alcance (limite : 12V).

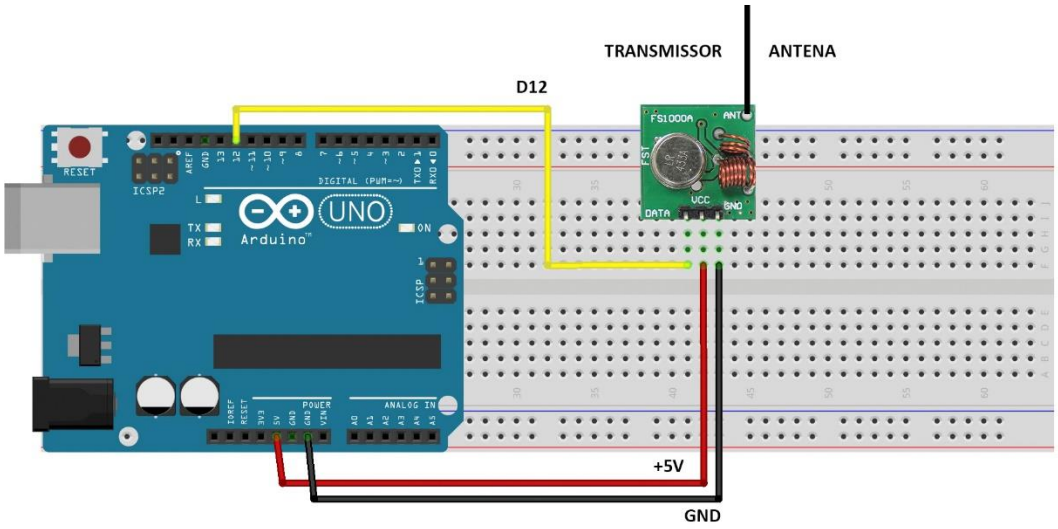
Como visto nas especificações do módulo TX, a taxa de transmissão de dados é relativamente baixa (< 4000 bps). Mas para aplicações como o envio de dados de um sensor, por exemplo, essa biblioteca poderá ser usada.

Montagem do Módulo Transmissor RF 433 MHz:

O Módulo Transmissor poderá ser montado com um Arduino Mega ou Mega. A alimentação do módulo Transmissor poderá ser de até 12V, para um maior alcance.

Esse é o Sketch para ser gravado no Arduino conectado ao Módulo Transmissor. Uma mensagem de teste será enviada à cada meio segundo.

http://blog.eletrogate.com/wp-content/uploads/2018/09/RF433askTX.ino



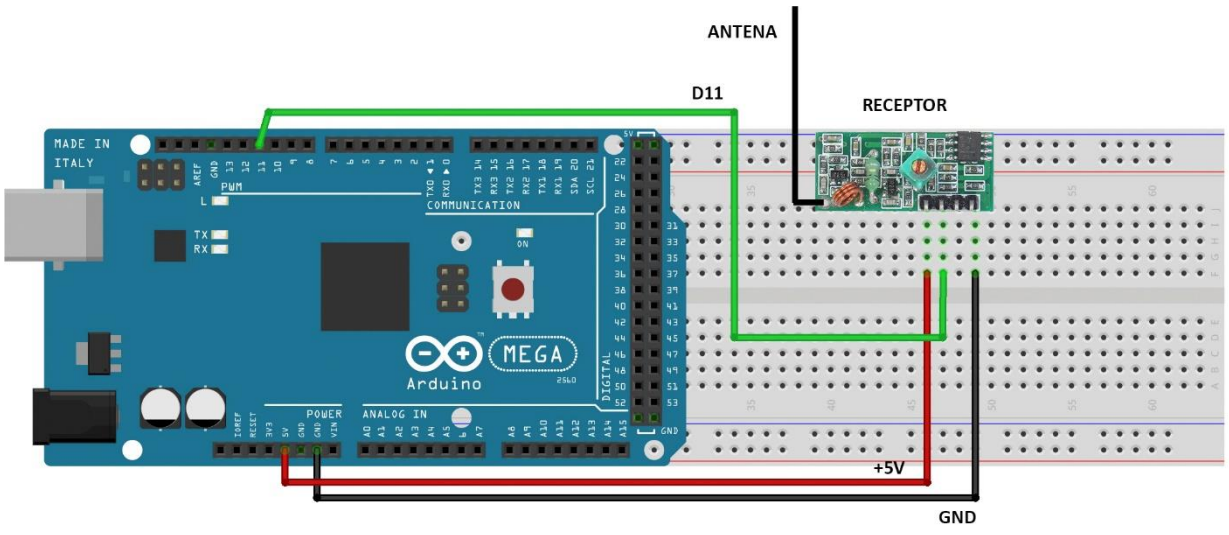
Módulo Transmissor RF 433 Mhz

Sketch do Módulo Transmissor :

```
// Exemplo 21 – Módulos RF – Transmissor e Receptor – 433 MHz
// Apostila Eletrogate - KIT ADVANCED
// Arduino Transmissor

#include <RHReliableDatagram.h>                    // biblioteca Radiohead reliableDatagram
#include <RH_ASK.h>                                // biblioteca Radiohead ASK
#include <SPI.h>                                   // biblioteca SPI

#define TX_ADDRESS 1                               // endereço do transmissor
#define RX_ADDRESS 2                               // endereço do recept

RH_ASK driver;                                     // instância RH ASK
RHReliableDatagram gerente(driver, TX_ADDRESS);    // configurando o gerenciador

uint8_t count = 1;                                 // contador
uint8_t data[] = "Mensagem de teste";              // mensagem a ser enviada
uint8_t buf[RH_ASK_MAX_MESSAGE_LEN];               // buffer da mensagem

void setup()
{
  Serial.begin(9600);                              // inicializa console serial 9600 bps
  if (!gerente.init())                             // se a inicialização do gerenciador falhar
    Serial.println("Falha na inicializacao");      // print na console serial
}

void loop()
{
  Serial.print("Transmitindo mensagem n. ");                // print na console serial
  Serial.println(count);                                    // print do contador
  if (!gerente.sendtoWait(data, sizeof(data), RX_ADDRESS))  // se gerenciador enviar mensagem
  {
    count++;                                                // incrementa contador
  }
  delay(500);                                               // atraso 0,5 segundos
}
```

Montagem do Módulo Receptor RF 433 MHz:

O Módulo Receptor pode ser montado com um Arduino Mega ou Mega.



Esse é o Sketch para ser gravado no Arduino conectado ao Módulo Receptor. Uma mensagem de teste será recebida à cada meio segundo.

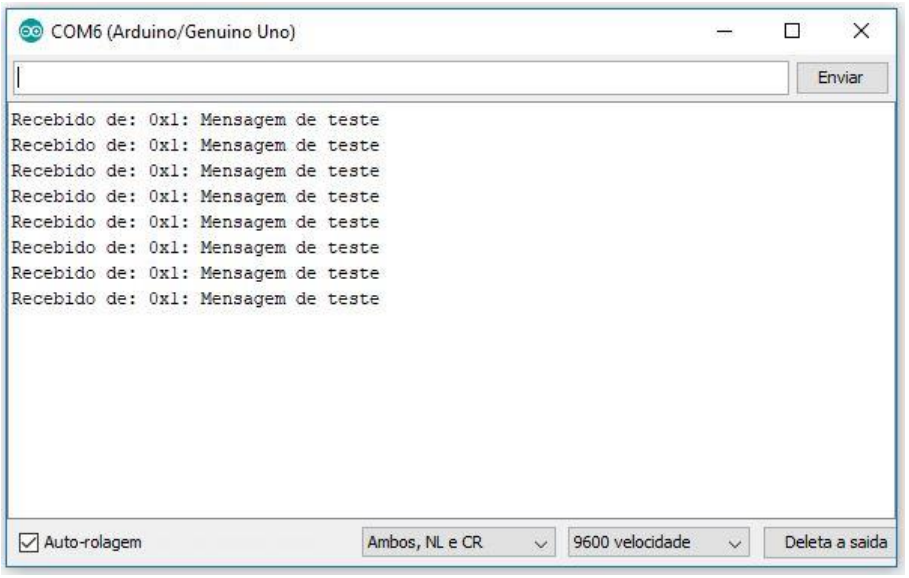
Durante os testes iniciais, coloque o transmissor perto do Receptor. Depois faça o teste distante um do outro. Obstáculos como paredes maciças, portões metálicos poderão prejudicar a recepção. Uma transmissão ao ar livre poderá ter um alcance bem maior.

http://blog.eletrogate.com/wp-content/uploads/2018/09/RF433askRX.ino.ino

Sketch do Modulo Receptor:

```
// Exemplo 12 – Módulos RF – Transmissor e Receptor – 433 MHz
// Apostila Eletrogate - KIT ADVANCED
// Arduino Receptor

#include <RHReliableDatagram.h>                    // biblioteca Radiohead reliableDatagram
#include <RH_ASK.h>                                // biblioteca Radiohead ASK
#include <SPI.h>                                   // biblioteca SPI

#define TX_ADDRESS 1                               // endereço do transmissor
#define RX_ADDRESS 2                               // endereço do receptor

uint8_t count = 0;                                 // contador
uint8_t buf[RH_ASK_MAX_MESSAGE_LEN];               // buffer da mensagem
uint8_t tamanho;                                   // tamanho da mensagem
uint8_t from;                                      // endereço de quem transmite

RH_ASK driver;                                     // instância RH ASK
RHReliableDatagram gerente(driver, RX_ADDRESS);    // configurando o gerenciador

void setup()
{
  Serial.begin(9600);                              // inicializa console serial 9600 bps
  if (!gerente.init())                             // se a inicialização do gerenciador falhar
    Serial.println("Falha na inicializacao");      // print na console serial
}

void loop()
{
  if (gerente.available())                         // se gerenciador estiver ativo
  {
    tamanho = sizeof(buf);                         // determina o tamanho do buffer
    if (gerente.recvfromAck(buf, &tamanho, &from))   // se o gerenciador receber mensagem
    {
      Serial.print("Recebido de: 0x");             // print na console serial
      Serial.print(from, HEX);                     // print do endereço do transmissor em
Hexadecimal
      Serial.print(": ");                          // print na console serial
      Serial.println((char*)buf);                  // print da mensagem recebida
    }
  }
}
```

Essa é a tela da Console Serial da IDE do Arduino Receptor (9600 bps), com as mensagens recebidas:



Referências de projetos interessantes com os Módulos de RF 433 MHz :

- *http://labdegaragem.com/forum/topics/desvendando-controle-remoto-rf*
- *http://blog.eletrogate.com/guia-basico-dos-modulos-tx-rx-rf-433mhz/*
- *https://acturcato.wordpress.com/*

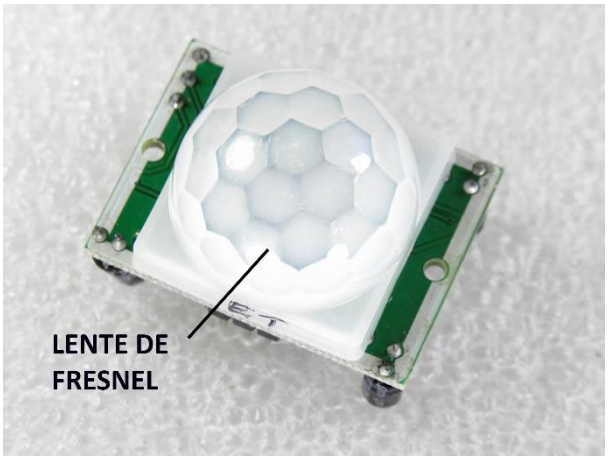
Exemplo 22 – Alarme com Buzzer e Sensor de Presença

O Módulo Sensor de presença usa um componente eletrônico piroelétrico passivo, sensível às variações de luz de raios infravermelhos – PIR (passive infrared). Sabemos que todos seres vivos emitem radiações de infravermelho, imperceptíveis ao olho humano. Dessa forma, esse sensor é capaz de detectar a presença de pessoas ou animais dentro de um ambiente.

https://en.wikipedia.org/wiki/Passive_infrared_sensor

Sob o sensor PIR existe uma lente de Fresnel que amplia a luz infravermelha recebida. Quando o ser vivo monitorado está em movimento, algumas variações na intensidade da luz são percebidas pelo circuito do módulo. Ao atingirem um determinado nível pré-configurado, o circuito dispara o sensor.

https://pt.wikipedia.org/wiki/Lente_de_Fresnel



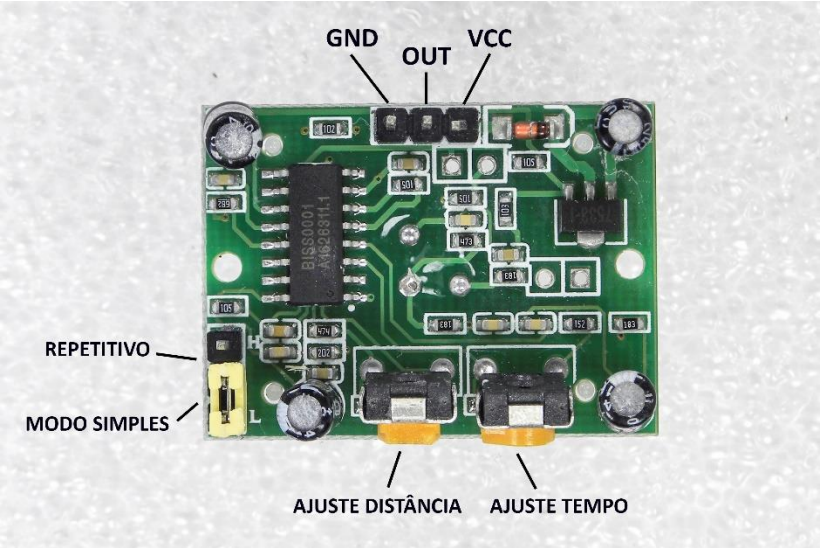
Especificações do Módulo PIR DYP-ME003:

- Sensibilidade e tempo ajustáveis
- Ângulo máximo de detecção = 110 °
- Tensão de Operação **VCC** : 4,5-20V
- Corrente estática: 50 uA
- Tensão de dados **OUT**: 3,3V (Alto) e 0V (Baixo)
- Distância detectável: 3-7m (Ajustável)
- Tempo de Atraso: 5 a 300seg (Default: 5seg)
- Tempo de Bloqueio: 2,5seg (Default)
- Dimensões: 3,2 x 2,4 x 1,8cm

https://www.elecfreaks.com/wiki/index.php?title=PIR_Motion_Sensor_Module:DYP-ME003

Aplicações sugeridas para o Módulo PIR:

- Sistemas de alarme de presença ou movimento,
- Disparo automáticos de câmera fotográfica – Camera Trap,
- Automação residencial – acionamento de lâmpadas,



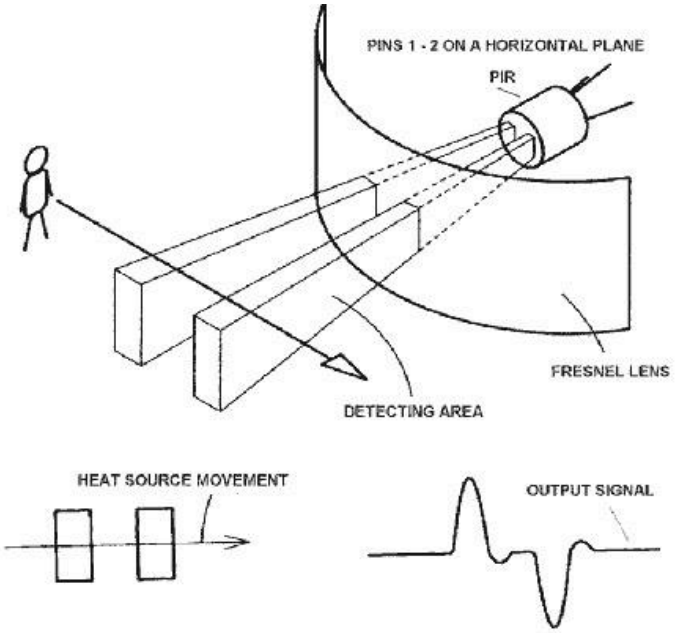
Modulo PIR DYP-ME003

OBS: alguns fabricantes podem inverter os pinos VCC com GND. Veja a identificação na placa.

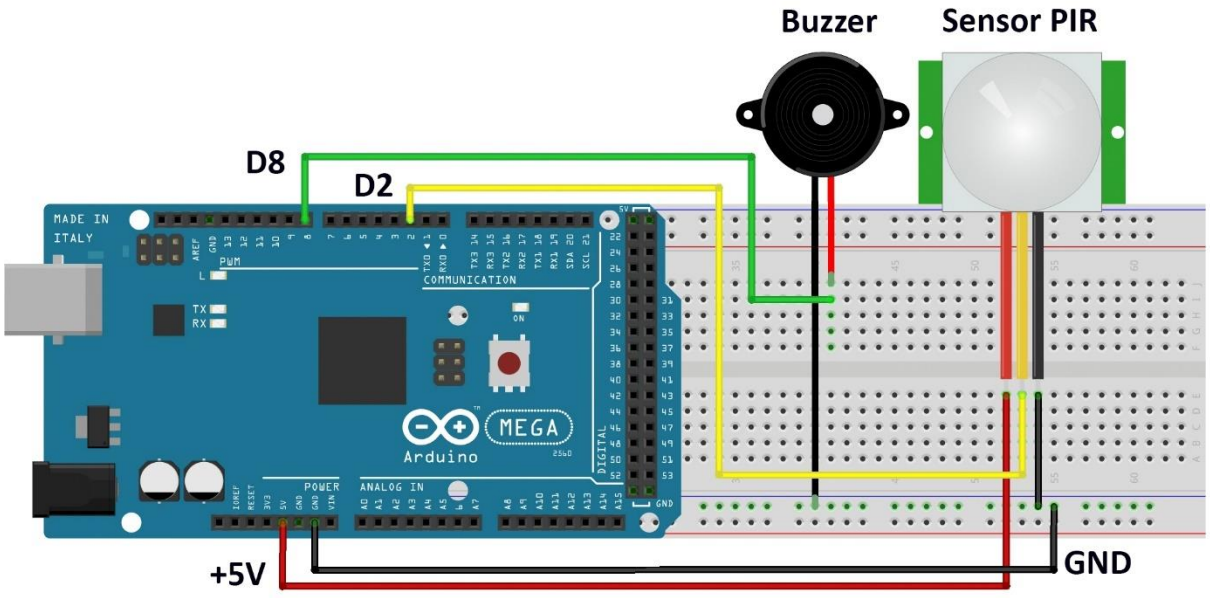
- Potenciômetro para ajuste do Tempo de atraso do disparo
  - Tempo de atraso mínimo de 5 segundos e máximo de 300 segundos
- Potenciômetro para ajuste de sensibilidade (distância)
  - A distância poderá ser ajustada entre 3 e 7 metros
- Jumper modo de disparo:
  - modo de disparo simples (posição L)
  - modo de disparo repetitivo (posição H = default)

Como funciona o Módulo PIR:

O sensor Piroelétrico possui duas áreas de "visão", sensíveis às radiações infravermelhas. As lentes de Fresnel focalizam as ondas de radiações para sobre o sensor. Quando o sensor está ocioso, ambas áreas sensíveis detectam a mesma quantidade de radiação. Quando um corpo quente como um ser humano ou animal passa na frente do sensor, variações nas quantidades de energia são percebidas pelo mesmo. Essas variações fazem o circuito disparar a saída OUT com um nível de tensão de 3,3V. A duração do pulso positivo nessa saída pode ser ajustada no potenciômetro TEMPO, entre 5 e 300 segundos. No potenciômetro de DISTÂNCIA ou sensibilidade, pode-se ajustar a distância entre o sensor e a pessoa, entre 3 e 7 metros. No modo simples, a partir de um disparo a saída se mantem alta durante o tempo ajustado. No modo repetitivo, os disparos se sobrepõem e assim a duração do pulso de saída será maior.



Montagem do Módulo PIR com Arduino:



```
// Exemplo 22 – Modulo Sensor de Presença PIR
// Apostila Eletrogate - KIT ADVANCED


int sensorPIR = 2;           // Pino ligado ao sensor PIR
int buzzer = 8;              // Pino ligado a sirene Buzzer
int disparo;                 // Variavel de disparo do sensor
int frequencia = 4000;       // Frequencia do apito em Hertz
int duracao = 500;           // Duração do apito em milisegundos


void setup()
{
  pinMode(LED_BUILTIN, OUTPUT);        // Define LED embutido D13 como saída
  pinMode(sensorPIR, INPUT);           // Define pino do sensor PIR D2 como entrada
  pinMode(buzzer, OUTPUT);             // define pino da sirene buzzer D8 como saída
  delay(5000);                         // tempo necessário para o módulo PIR se inicializar 5 segundos
}

void loop()
{
  disparo = digitalRead(sensorPIR);          // Le o valor da saída do sensor PIR
  if (disparo == LOW)                        // Se saída tem nível baixo
  {
    digitalWrite(LED_BUILTIN, LOW);          // LED embutido permanece apagado
  }
  else                                       // Se a saída tem nível alto
  {
    digitalWrite(LED_BUILTIN, HIGH);         // LED embutido acende
    tone(buzzer, frequencia, duracao);       // Buzzer apita
  }
}
```

Para o funcionamento do circuito, ajuste os potenciômetros de Tempo e Distancia para a posição mínima (gire tudo para o sentido anti-horário). Dessa forma o tempo de duração será de aproximadamente 5 segundos e a distância de percepção será de 3 metros. Insira o jumper de modo para a posição H, para que o sensor fique mais perceptível. Despois dos testes, faça as suas modificações.

Quando o sensor perceber a presença de alguém, o LED embutido no Arduino (D13) acenderá e a sirene Buzzer apitará na frequência de 4000 Hz.

Com essa montagem, usando o relé do exemplo 8 poderá implementar um sistema de alarme ou automação residencial.

Parte V - Cálculo do resistor de base dos transistores

Vamos abordar aqui como o cálculo do resistor de base deve ser feito. É de suma importância dimensioná-lo corretamente pois, como visto anteriormente, esse resistor é o responsável por controlar a corrente que fluirá pelo transistor.

O primeiro passo é calcular a corrente de base usando a fórmula

$Ib = \frac{Ic}{\beta}$, onde:

Ib representa o valor da corrente de base, Ic é o valor da corrente que fluirá pelo coletor (dada em Ampères) e β é o ganho do transistor.

Com o valor de Ib em mãos, vamos utilizar a lei de Ohm para encontrar o valor da resistência através da seguinte fórmula:

$Rb = \frac{Vb - Vbe}{Ib}$, onde:

Rb representa o valor do resistor de base em ohms, Vb é valor da tensão aplicada na base (no caso de uma aplicação com o Arduino, esse valor será de 5V) e Vbe é o valor da tensão entre base e emissor do transistor quando este está ligado (considere como sendo tipicamente 0,7V). Ib, como dito anteriormente, é o valor da corrente de base.

***IMPORTANTE**: O valor de ganho é encontrado no datasheet do componente (você encontra facilmente na internet). Abaixo, retiramos um trecho do datasheet do transistor BC548B para exemplificarmos essa questão:*

h_{FE} Classification

Classification	A	B	C
h_{FE}	110 ~ 220	200 ~ 450	420 ~ 800

Se utilizarmos qualquer valor de ganho dentro da faixa especificada, o transistor funcionará na região ativa. Para que o transistor em opere em modo saturação, o ganho utilizado no cálculo deve ser muito inferior ao ganho mínimo (nesse caso, a 200), sendo comum utilizar apenas 10% desse valor.

Outros parâmetros também são importantes ao dimensionar um transistor para seu circuito, como corrente de coletor máxima suportada, tensão máxima etc., ambos presentes no datasheet. Não abordaremos esses detalhes nessa apostila.

Dito isso, vamos considerar o exemplo 12 da apostila como modelo de cálculo. Nele, nós temos que:

- O transistor opera como chave (em modo saturação - lembre-se, utilize 10% do ganho mínimo especificado nessa situação);
- A corrente de coletor é 75mA (equivalente a 0,075A - para converter mA em A, basta dividir por 1000);
- A tensão de base é 5V;
- O modelo do transistor é o BC548B.

Com esses dados em mãos, vamos aos cálculos. Primeiramente, encontramos o valor da corrente de base:

$$Ib = \frac{Ic}{\beta} \rightarrow Ib = \frac{0{,}075}{20} \rightarrow Ib = 0{,}00375$$

Com o valor de Ib encontrado, vamos para a segunda fórmula. Nela, temos que:

- Vb equivale a 5V;
- Vbe equivale a 0,7V;
- Ib equivale a 0,00375A

Substituindo:

$$Rb = \frac{Vb - Vbe}{Ib} \rightarrow Rb = \frac{5-0{,}7}{0{,}00375} \rightarrow Rb = \frac{4{,}3}{0{,}00375} \rightarrow Rb = 1.146{,}66$$

Veja que o valor encontrado para Rb é de 1.146Ω, porém não encontramos esse resistor comercialmente. O valor mais próximo disponível no Kit Advanced é de 1KΩ (1000Ω), portanto esse deve ser o resistor utilizado na base do transistor no exemplo 12.

E aí, já consegue confirmar o valor dos resistores de base para o exemplo 11? Deixaremos essa tarefa como exercício para você!

Parte VI – Principais comandos do Arduino

Nesta seção, trazemos um resumo dos comandos da Arduino IDE utilizados ao longo dos exemplos. Essas referências são ferramentas valiosas para esclarecer suas dúvidas sobre a sintaxe (a estrutura dos comandos) e, mais importante, para desvendar o funcionamento de cada um deles. Aproveite esta oportunidade para aprofundar seu conhecimento e dominar a programação no Arduino!

- ***#include <biblioteca>***: Tem por função permitir a importação de bibliotecas externas, como a Servo.h.
- ***#define***: Define constantes que serão usadas ao longo do código, como um pino do Arduino.
- ***pinMode(pino, modo)***: Comando usado para configurar um pino como entrada (INPUT), saída (OUTPUT) ou ainda como um tipo especial de entrada que dispensa o uso de resistores para acionar botões (INPUT_PULLUP).
- ***digitalWrite(pino, valor)***: Define se o pino configurado estará ligado (HIGH) ou desligado (LOW).
- ***digitalRead(pino)***: Faz a leitura de um pino digital, verificando se está ligado ou desligado.
- ***analogRead(pino)***: Faz a leitura de um pino analógico, retornando valores entre 0 e 1023.
- ***analogWrite(pino, valor)***: Envia um sinal PWM para os pinos digitais compatíveis.
- ***delay(x)***: Pausa a execução do código por x milissegundos.
- ***millis()***: Retorna o número de milissegundos desde que o Arduino começou a rodar o código.
- ***map(fonte, in_min, in_max, out_min, out_max)***: Converte um intervalo inicial proveniente de uma (como uma entrada analógica) para outro, de maneira proporcional.
- ***Serial.begin(velocidade)***: Inicializa a comunicação serial com uma velocidade específica em bits por segundo.
- ***Serial.print()*** e ***Serial.println()***: Imprimem dados no monitor serial. O segundo, acrescido de uma quebra de linha.
- ***if (condição)*** / ***else***: Laço condicional (Se condição, execute y, senão, execute z).
- ***for (inicialização; condição; incremento)***: Laço de repetição que itera sobre um bloco de código um número definido de vezes (x recebe y, enquanto x for menor que z, incremente x).
- ***switch (variável)***/ ***case possível valor*** / ***break***: estrutura usada para selecionar um bloco de código a ser executado com base no valor da variável. As

possibilidades são listadas com o comando “case” e encerradas com o comando “break”.

- ***int:*** Usado para iniciar uma variável ou array do tipo inteiro, que trabalha somente com números inteiros (sem casas decimais).
- ***float:*** Usada para criar variáveis e arrays que precisam receber números com ponto flutuante (números com casas decimais).
- ***unsigned long:*** Usado para criar variáveis que precisam armazenar grandes números positivos, como os fornecidos pela função millis().
- ***void:*** Usado para criar funções que não retornam nenhum tipo de valor.
- ***Servo / attach / write:*** comandos dedicados da biblioteca Servo.h, usados para criar o objeto que receberá as funções da biblioteca, configurar o pino em que o servo está conectado e enviar os pulsos de forma correta.

Sugerimos que, além de consultar esse pequeno resumo, acesse a documentação oficial do Arduino e estude os demais comandos e sua sintaxe através do link a seguir:

- https://www.arduino.cc/reference/pt/

Considerações finais

Essa apostila tem por objetivo apresentar alguns exemplos básicos sobre como utilizar os componentes do Kit Advanced para Arduino, a partir dos quais você pode combinar e fazer projetos mais elaborados por sua própria conta.

Nas referências finais, também tentamos indicar boas fontes de conteúdo objetivo e com projetos interessantes. Sobre esse ponto, que consideramos fundamental, gostaríamos de destacar algumas fontes de conhecimento que se destacam por sua qualidade.

O fórum oficial Arduino possui muitas discussões e exemplos muito bons. A comunidade de desenvolvedores é bastante ativa e certamente pode te ajudar em seus projetos. No Project Hub poderá encontrar milhares de projetos com Arduino.

- Fórum oficial Arduino: https://forum.arduino.cc/
- Project Hub Arduino: https://create.arduino.cc/projecthub

O Instructables é a ótima referência do mundo maker atual. Pessoas que buscam construir suas próprias coisas e projetos encontram referências e compartilham suas experiências no site.

- Instructables: https://www.instructables.com/

O Maker Pro é outro site referência no mundo em relação aos projetos com Arduino. Há uma infinidade de projetos, todos bem explicados e com bom conteúdo.

- Maker pro: https://maker.pro/projects/arduino

Em relação à eletrônica, teoria de circuitos e componentes eletrônicos em geral, deixamos alguns livros essenciais na seção finais de referências. O leitor que quer se aprofundar no mundo da eletrônica certamente precisará de um livro basilar e de bons conhecimentos em circuitos eletroeletrônicos.

No mais, esperamos que essa apostila seja apenas o início de vários outros projetos e, quem sabe, a adoção de Kits mais avançados, como os de **Robótica** e **Arduino Advanced**. Qualquer dúvida, sugestão, correção ou crítica a esse material, fique à vontade para relatar em nosso blog oficial: http://blog.eletrogate.com/

Referências gerais

1. Fundamentos de Circuitos Elétricos. Charles K. Alexander; Matthew N. O. Sadiku. Editora McGraw-Hill.
2. Circuitos elétricos. James W. Nilsson, Susan A. Riedel. Editora: Pearson; Edição: 8.
3. Microeletrônica - 5ª Ed. - Volume Único (Cód.: 1970232). Sedra, Adel S. Editora Pearson.
4. Fundamentals of Microelectronics. Behzad Razavi. Editora John Wiley & Sons; Edição: 2nd Edition (24 de dezembro de 2012).

Abraços e até a próxima!

APOSTILA KIT

ARDUINO ADVANCED

Esta apostila acompanha o **Kit ARDUINO ADVANCED** da Eletrogate, e contém conteúdos relacionados a todos os componentes do Kit.

WWW.ELETROGATE.COM

ELETROGATE