CAPÍTULO 2

Acendendo as luzes

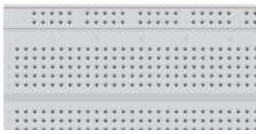
Neste capítulo, você trabalhará em seus quatro primeiros projetos. Todos utilizam luzes LED de diversas formas. Você aprenderá a controlar as saídas do Arduino e também entradas simples, como botões pressionados. No lado do hardware, você aprenderá sobre LEDs, botões e resistores, incluindo resistores pull-up e pull-down, importantes para garantir que os dispositivos de entrada sejam lidos corretamente. Nesse processo, você aprenderá conceitos de programação com a linguagem do Arduino. Vamos começar com um projeto "Olá mundo", que faz seu Arduino piscar um LED externo.

Projeto 1 – LED piscante

Para o primeiro projeto, você repetirá o sketch que utilizou no estágio de testes. Dessa vez, no entanto, você conectará um LED a um dos pinos digitais em vez de utilizar o LED 13, soldado na placa. Você também aprenderá exatamente como o hardware e o software para este projeto funcionam, aprendendo, ao mesmo tempo, um pouco sobre eletrônica e codificação na linguagem do Arduino (que é uma variante da linguagem C).

Componentes necessários

Protoboard



LED de 5 mm

Resistor de 100 ohms*

Fios jumper

* Esse valor pode ser diferente, dependendo do LED que você utilizar. O texto explicará como descobrir o valor correto.

O melhor tipo de protoboard para a maioria dos projetos deste livro é uma protoboard de 840 pontos. Protoboards desse tipo costumam ser de tamanho padrão, medindo aproximadamente 16,5 cm por 5,5 cm e apresentando 840 furos (ou pontos) na placa. Geralmente, as placas têm pequenos encaixes nas laterais que permitem conectar diversas placas, umas às outras, para criar protoboards maiores; isso é útil para projetos mais complexos. Para este projeto, entretanto, uma protoboard de tamanho normal será suficiente.

O LED deverá ser de 5 mm, de uma cor qualquer. Você também deverá saber qual a corrente e a voltagem (por vezes chamadas de corrente contínua e voltagem contínua) do LED, para poder calcular o valor do resistor necessário — isso será feito mais adiante no projeto.

Os fios jumper que utilizaremos podem ser fios encontrados comercialmente (geralmente com pontas moldadas para facilitar sua inserção na protoboard), ou você pode criar seus próprios, cortando tiras curtas de fios rígidos de núcleo único e retirando cerca de 6 mm da ponta.

Conectando os componentes

Primeiro, certifique-se de que seu Arduino esteja desligado, desconectando-o do cabo USB. Agora, pegue sua protoboard, o LED, o resistor e os fios, e conecte tudo como mostra a figura 2.1.

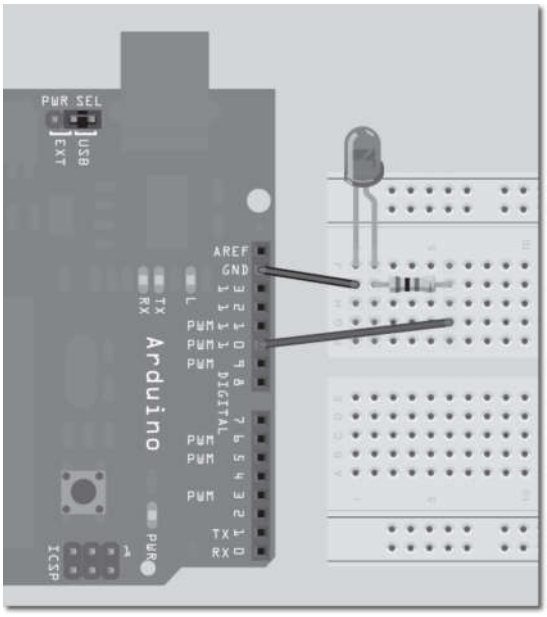


Figura 2.1 – Circuito para o Projeto 1 – LED piscante (consulte o site da Novatec para versão colorida).

Não importa se você utiliza fios de cores diferentes ou furos diferentes na protoboard, desde que os componentes e os fios estejam conectados na mesma ordem da figura.

Tenha cuidado ao inserir os componentes na protoboard. Caso sua protoboard seja nova, a superfície dos furos ainda estará rígida. A não inserção cuidadosa dos componentes pode resultar em danos.

Certifique-se de que seu LED esteja conectado corretamente, com o terminal (ou perna) mais longo conectado ao pino digital 10. O terminal longo é o ânodo do LED, e deve sempre ir para a alimentação de +5 V (nesse caso, saindo do pino digital 10); o terminal curto é o cátodo e deve ir para o terra (GND).

Quando você estiver seguro de que tudo foi conectado corretamente, ligue seu Arduino e conecte o cabo USB.

Digite o código

Abra seu IDE do Arduino e digite o código da listagem 2.1.

Listagem 2.1 – Código para o projeto 1

```
// Projeto 1 - LED piscante
int ledPin = 10;
void setup() {
  pinMode(ledPin, OUTPUT);
}
void loop() {
  digitalWrite(ledPin, HIGH);
  delay(1000);
  digitalWrite(ledPin, LOW);
  delay(1000);
}
```

Pressione o botão Verify/Compile no topo do IDE para certificar-se de que não há erros em seu código. Se não houver erros, clique no botão Upload para fazer o upload do código ao seu Arduino. Caso tudo tenha sido feito corretamente, agora você deverá ver o LED vermelho, na protoboard, acendendo e apagando em intervalos de um segundo.

Vamos analisar o código e o hardware para descobrir como ambos funcionam.

Projeto 1 – LED piscante – Análise do código

A primeira linha do código do projeto é:

```
// Projeto 1 - LED piscante
```

Trata-se apenas de um *comentário* em seu código. Você pode perceber que é um comentário porque ela inicia com //, e qualquer texto que inicie dessa forma é ignorado pelo compilador. Comentários são essenciais em seu código; eles ajudam a compreender como seu código funciona. À medida que seus projetos se tornarem mais complexos, e seu código se expandir até centenas ou talvez milhares de linhas, comentários

serão vitais para facilitar a compreensão de como cada seção funciona. Você pode desenvolver um trecho incrível de código, mas não pode simplesmente supor que se lembrará de como ele funciona quando revisitá-lo muitos dias, semanas ou meses depois. Comentários, por outro lado, poderão ajudá-lo a recordar a funcionalidade do código. Da mesma forma, caso seu código seja visto por outras pessoas, comentários poderão ajudar esses indivíduos a compreender o que ocorre nele. O espírito do Arduino, e de toda a comunidade de fonte aberta, trata do compartilhamento de código e projetos. Espero que, quando você vier a criar seus próprios projetos com o Arduino, também esteja disposto a compartilhá-los com o mundo.

Há outro formato para comentários: um bloco de instrução dentro dos sinais `/*` e `*/`, da seguinte maneira:

```
/* Todo o texto dentro
das barras e dos asteriscos
é um comentário, e será
ignorado pelo compilador */
```

O IDE transformará automaticamente a cor de todos os comentários em cinza. A linha seguinte no programa é

```
int ledPin = 10;
```

Isso é que chamamos de *variável*. Uma variável é um local em que podemos armazenar dados. Nesse caso, você está definindo uma variável de tipo `int`, ou inteiro. Um *inteiro* é um número dentro do intervalo de -32.768 e 32.767. Em seguida, você atribui a esse inteiro o nome `ledPin` e dá a ele um valor de `10`. (Você não tinha de chamá-lo `ledPin`, poderia ter dado o nome que quisesse. Mas você deve sempre procurar fazer com que suas variáveis tenham nomes descritivos, por isso `ledPin`, para mostrar que esta variável define qual pino no Arduino você utilizará para conectar o LED.) Nesse caso você está utilizando o pino digital 10. Ao final da instrução há um ponto e vírgula. Esse símbolo diz ao compilador que a instrução, agora, está completa.

Ainda que você possa dar qualquer nome às suas variáveis, todas as variáveis em C devem iniciar com uma letra; o restante do nome pode ser formado por letras, números e underscores. Note que a linguagem C diferencia caracteres maiúsculos e minúsculos. Por fim, você não pode utilizar nenhuma das palavras-chave da linguagem, como `main`, `while`, `switch` etc. como nomes de variáveis. Palavras-chave são nomes de constantes: variáveis e funções definidas como parte da linguagem do Arduino. Para evitar que você nomeie uma variável como uma palavra-chave, todas as palavras-chave no sketch serão mostradas em vermelho.

Imagine uma variável como uma pequena caixa, na qual você pode guardar objetos. Assim, nesse sketch, você preparou uma área na memória para armazenar um número de valor inteiro, e armazenou nessa área o número 10.

Por fim, uma variável é chamada de "variável" porque você pode modificá-la. Futuramente, você realizará cálculos matemáticos em variáveis, para fazer com que seu programa realize funções mais avançadas.

Em seguida, vemos a função `setup()`:

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

Um sketch do Arduino deve ter uma função `setup()` e uma função `loop()`, do contrário, não funcionará. A função `setup()` é executada somente uma vez no início do programa, e é nela que você emitirá instruções gerais para preparar o programa antes que o loop principal seja executado, como a definição dos modos dos pinos, das taxas de transmissão serial etc. Basicamente, uma função é uma porção de código agrupada em um bloco conveniente. Por exemplo, se você tivesse criado sua própria função para realizar uma série de complicadas operações matemáticas, e que tivesse muitas linhas de código, poderia executar esse código quantas vezes quisesse simplesmente chamando o nome da função, em vez de ter de reescrever o código cada vez que fosse usá-lo. Você verá funções em maiores detalhes no futuro, quando começar a criar suas próprias. No caso desse programa, todavia, a função `setup()` tem somente uma instrução para executar. A função inicia com

```
void setup()
```

Isso diz ao compilador que sua função é chamada `setup`, que ela não retorna nenhum dado (`void`) e que você não passa nenhum parâmetro a ela (parênteses vazios). Se sua função retornasse um valor inteiro e você também tivesse valores inteiros a serem passados a ela (por exemplo, para serem processados pela função), o resultado seria algo como:

```
int myFunc(int x, int y)
```

Aqui você criou uma função (ou um bloco de código) de nome `myFunc`. A ela foram passados dois inteiros, `x` e `y`. Assim que tiver concluído, ela retornará um valor inteiro, no ponto de execução imediatamente depois de onde havia sido chamada no programa (portanto, utilizamos `int` antes do nome da função).

Todo o código dentro da função está contido entre chaves. Um símbolo `{` inicia o bloco de código, e um símbolo `}` termina o bloco. Tudo que existir entre esses dois símbolos, no código, fará parte da função. (Falarei mais detalhadamente sobre funções no futuro, por isso não se preocupe com elas, por enquanto.)

Nesse programa, você tem duas funções; a primeira é chamada `setup`, e seu propósito é preparar tudo que é necessário para que seu programa funcione antes da execução do loop principal do programa:

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

Sua função `setup` tem apenas uma instrução, `pinMode`, que diz ao Arduino que você deseja definir o modo de um de seus pinos como Saída (Output), e não Entrada (Input). Dentro dos parênteses, você coloca o número do pino e o modo (`OUTPUT` ou `INPUT`). O número de seu pino é `ledPin`, previamente definido com o valor `10`. Dessa forma, essa instrução está simplesmente dizendo ao Arduino que o pino digital 10 deve ser definido como modo `OUTPUT`. Como a função `setup()` executa apenas uma vez, agora você avança para o loop principal da função:

```
void loop() {
  digitalWrite(ledPin, HIGH);
  delay(1000);
  digitalWrite(ledPin, LOW);
  delay(1000);
}
```

A função `loop()` é a função principal do programa e executa continuamente enquanto o Arduino estiver ligado. Todas as declarações dentro da função `loop()` (dentro das chaves) são executadas uma de cada vez, passo a passo, até que se alcance o fim da função; nesse ponto, o loop reinicia desde o princípio e assim infinitamente, ou até que o Arduino seja desligado ou o botão Reset pressionado.

Neste projeto, você deseja que o LED acenda, fique aceso por um segundo, apague, permaneça apagado por um segundo e então repita o processo. Os comandos para dizer ao Arduino como fazer essas operações estão dentro da função `loop()`, pois você deseja que sejam repetidos seguidas vezes. A primeira instrução é:

```
digitalWrite(ledPin, HIGH);
```

Ela escreve um valor `HIGH` ou `LOW` para o pino dentro da instrução (nesse caso `ledPin`, que é o pino digital 10). Quando você define um pino como `HIGH`, está enviando 5 volts para ele. Quando define como `LOW`, o pino se torna 0 volt, ou terra. Essa instrução, portanto, envia 5 V para o pino 10 e acende o LED. Depois dela, temos:

```
delay(1000);
```

Essa instrução simplesmente diz ao Arduino para esperar 1.000 milissegundos (há 1.000 milissegundos em um segundo) antes de executar a instrução seguinte:

```
digitalWrite(ledPin, LOW);
```

O que desliga a força que vai para o pino digital 10 e apaga o LED. Então, há outra instrução de espera por mais 1.000 milissegundos, e depois a função termina. Entretanto, como essa é sua função `loop()` principal, a função reiniciará desde o princípio.

Seguindo a estrutura do programa passo a passo novamente, você pode ver que tudo é muito simples:

```
// Projeto 1 - LED piscante
int ledPin = 10;
void setup() {
  pinMode(ledPin, OUTPUT);
}
void loop() {
  digitalWrite(ledPin, HIGH);
  delay(1000);
  digitalWrite(ledPin, LOW);
  delay(1000);
}
```

Você inicia atribuindo uma variável `ledPin`, e dando a ela um valor de `10`. Depois, avança para a função `setup()`, na qual você define o modo para o pino digital 10 como saída. No loop principal do programa, você define o pino digital 10 como `HIGH`, enviando a ele 5 V. Então, espera por um segundo e desliga os 5 V, antes de esperar mais um segundo. O loop então reinicia desde o princípio: o LED acenderá e apagará continuamente, enquanto o Arduino tiver energia.

Agora que você sabe de tudo isso, pode modificar o código para acender o LED por um intervalo diferente de tempo e desligá-lo por outro intervalo. Por exemplo, se você quisesse que o LED ficasse aceso por dois segundos e, depois, apagado por meio segundo, poderia fazer o seguinte:

```
void loop() {
  digitalWrite(ledPin, HIGH);
  delay(2000);
  digitalWrite(ledPin, LOW);
  delay(500);
}
```

Se quisesse que o LED ficasse apagado por cinco segundos e, depois, piscasse brevemente (250ms), como o indicador LED de um alarme de carro, poderia fazer o seguinte:

```
void loop() {
  digitalWrite(ledPin, HIGH);
  delay(250);
  digitalWrite(ledPin, LOW);
  delay(5000);
}
```

Para que o LED pisque, acendendo a apagando rapidamente, tente:

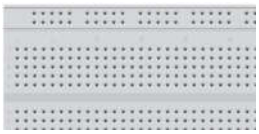
```
void loop() {
  digitalWrite(ledPin, HIGH);
```

```
    delay(50);
    digitalWrite(ledPin, LOW);
    delay(50);
}
```

Alternando o tempo em que o LED fica aceso e apagado, você pode criar o efeito que quiser (dentro do que pode ser feito com um único LED). Antes de avançarmos para algo mais empolgante, vamos analisar o hardware e ver como ele funciona.

Projeto 1 – LED piscante – Análise do hardware

O hardware utilizado no projeto 1:

Protoboard 

LED de 5 mm 

Resistor de 100 ohms* 

Fios jumper 

A protoboard é um dispositivo reutilizável, sem solda, utilizado para prototipar um circuito eletrônico ou para experimentar projetos de circuitos. A placa consiste em uma série de furos em uma grade; sob a placa, esses furos são conectados por uma tira de metal condutivo. A forma como essas tiras são dispostas é tipicamente a que vemos na figura 2.2.

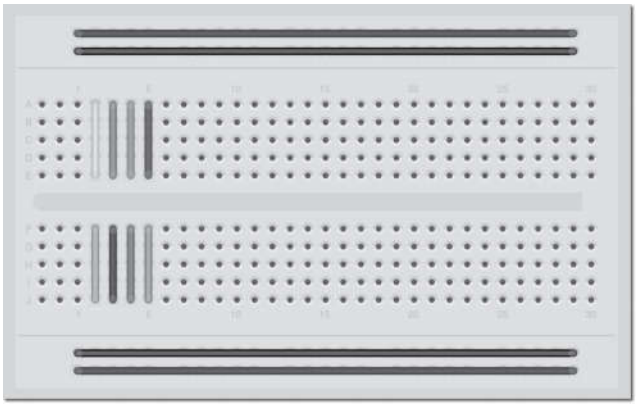


Figura 2.2 – Como são dispostas as tiras de metal em uma protoboard.

As tiras ao longo do topo e da base correm em paralelo à placa, e são projetadas para carregar o barramento de alimentação e o barramento do terra. Os componentes no meio da placa convenientemente conectam com os 5 V (ou a voltagem que você estiver

utilizando) ou com o terra. Algumas protoboards têm uma linha vermelha e outra preta correndo paralelas a esses furos, para mostrar qual é a alimentação (vermelho) e qual é o terra (preto). Em protoboards maiores, o barramento de alimentação às vezes tem uma divisão, indicada por uma quebra na linha vermelha. Isso torna possível enviar voltagens diferentes para partes distintas de sua placa. Caso você esteja utilizando apenas uma voltagem, um pequeno pedaço de fio jumper pode ser colocado de um lado a outro desse espaço, para garantir que a mesma voltagem seja aplicada em todo o barramento.

As tiras no centro correm a 90 graus dos barramentos de alimentação e do terra, e há um espaço vazio no meio para que você possa colocar Circuitos Integrados, de modo que cada pino do chip vá para um conjunto diferente de furos e, portanto, para um barramento diferente (Figura 2.3).

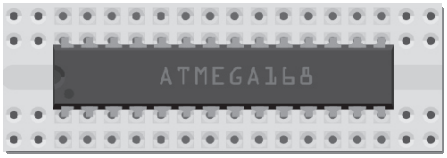


Figura 2.3 – Circuito Integrado (ou chip) conectado no espaço de uma protoboard.

O próximo componente é um *resistor*, um dispositivo projetado para provocar resistência a uma corrente elétrica, causando uma queda na voltagem em seus terminais. Você pode pensar em um resistor como um cano de água muito mais fino do que o cano conectado a ele. Conforme a água (ou a corrente elétrica) entra no resistor, o cano se torna mais fino e o volume da água (corrente) saindo na outra ponta é, dessa forma, reduzido. Você utiliza resistores para diminuir a voltagem ou a corrente para outros dispositivos.

O valor de resistência é conhecido como ohm, e seu símbolo é o ômega grego, Ω. Nesse caso, o pino digital 10 está emitindo 5 V de corrente contínua a 40 mA (miliampères; amperagem de acordo com o datasheet[1] da Atmega), e seu LED requer (de acordo com o datasheet) uma voltagem de 2 V e uma corrente de 35 mA. Portanto, você necessita de um resistor que reduza os 5 V para 2 V, e a corrente de 40 mA para 35 mA, caso queira exibir o LED com brilho máximo. Se você deseja um LED de menor luminosidade, pode utilizar um valor mais alto de resistência.

> **Nota**: NUNCA utilize um valor de resistor mais BAIXO que o necessário. Você colocará corrente demais no LED, danificando-o permanentemente. Você também poderia danificar outros componentes de seu circuito.

A fórmula para calcular o resistor necessário é

$$R = (V_S - V_L) / I$$

1 N.T.: Datasheet, ou folha de dados, é um termo técnico usado para identificar um documento relativo a um determinado produto, contendo suas especificações técnicas (fonte: Wikipédia).

Em que V_S é a voltagem fornecida, V_L é a voltagem do LED e I é a corrente do LED. Nosso LED de exemplo tem uma voltagem de 2 V e uma corrente de 35 mA, conectado a um pino digital do Arduino, de 5 V, assim o valor necessário para o resistor seria de

R = (5 – 2) / 0.035

o que dá um valor de 85,71.

Resistores vêm com valores-padrão e o valor comum mais próximo nesse caso seria de 100 Ω. Sempre escolha o resistor com valor mais próximo MAIOR do que o valor necessário. Se você escolher um valor menor, muita corrente atravessará o resistor, danificando-o.

Mas como encontrar um resistor de 100 Ω? Um resistor é pequeno demais para conter informações de fácil leitura, por isso, resistores utilizam um código de cores. Ao redor do resistor você tipicamente encontrará quatro faixas de cores; utilizando o código da tabela 2.1 você pode descobrir o valor de um resistor. Da mesma forma, você pode descobrir o código de cores de uma determinada resistência.

Tabela 2.1 – Código de cores dos resistores

Cor	Primeira faixa	Segunda faixa	Terceira faixa (multiplicador)	Quarta faixa (tolerância)
Preto	0	0	$x10^0$	
Marrom	1	1	$x10^1$	±1%
Vermelho	2	2	$x10^2$	±2%
Laranja	3	3	$x10^3$	
Amarelo	4	4	$x10^4$	
Verde	5	5	$x10^5$	±0,5%
Azul	6	6	$x10^6$	±0,25%
Violeta	7	7	$x10^7$	±0,1%
Cinza	8	8	$x10^8$	±0,05%
Branco	9	9	$x10^9$	
Dourado			$x10^{-1}$	±5%
Prata			$x10^{-2}$	±10%
Nenhuma				±20%

De acordo com a tabela, para um resistor de 100 Ω você necessita de 1 na primeira faixa, que é marrom, seguido por um 0 na faixa seguinte, que é preta. Então, deve multiplicar isso por 10^1 (em outras palavras, adicionar um zero), o que resulta em marrom na terceira faixa. A faixa final indica a tolerância do resistor. Caso seu resistor tenha uma faixa dourada, ele tem uma tolerância de ±5%; isso significa que o valor, de fato, do resistor varia entre 95 Ω e 105 Ω. Dessa forma, se você tem um LED que requer 2 V e 35 mA, necessitará de um resistor com uma combinação de faixas Marrom, Preto, Marrom.

Caso queira um resistor de 10 kΩ (ou 10 quilo-ohms), necessitará de uma combinação Marrom, Preto, Laranja (1, 0, +3 zeros). Se necessitar de um resistor de 570 kΩ, as cores serão Verde, Violeta e Amarelo.

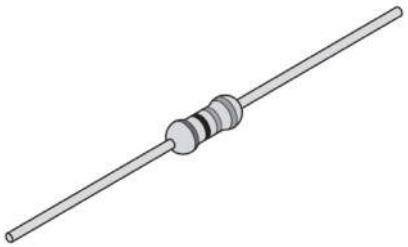


Figura 2.4 – Resistor de 10 kΩ com tolerância de 5%.

Da mesma forma, se você encontrasse um resistor e quisesse saber seu valor, poderia fazer o mesmo processo em ordem inversa. Assim, se encontrasse o resistor da figura 2.4 e quisesse descobrir seu valor para que pudesse guardá-lo em uma caixa devidamente marcada, poderia consultar a tabela e ver que ele tem um valor de 220 Ω.

Agora que você sabe como funciona a codificação por cores, escolha o valor de resistência correto para o LED que você comprou, para completar este projeto.

O componente final (fora os fios jumper, mas estou certo de que você pode descobrir sozinho o que eles fazem) é o LED, que significa Light Emitting Diode (Diodo Emissor de Luz). Um diodo é um dispositivo que permite o fluxo de corrente em apenas uma direção; é como uma válvula em um sistema de distribuição de água, mas nesse caso ele permite o fluxo da corrente elétrica em uma direção. Caso a corrente tente reverter e retornar na direção oposta, o diodo impede que ela o faça. Diodos podem ser úteis para prevenir que alguém conecte acidentalmente a alimentação e o terra aos terminais errados em um circuito, danificando os componentes.

Um LED é a mesma coisa, mas ele também emite luz. LEDs vêm de todos os tipos de cores e níveis de luminosidade, incluindo a parte ultravioleta e infravermelha do espectro (como nos LEDs do controle remoto de sua TV).

Caso examine cuidadosamente seu LED, você perceberá dois detalhes: os terminais têm comprimentos diferentes, e um lado do LED é chanfrado, em vez de cilíndrico (Figura 2.5). Essas são pistas que indicam qual terminal é o ânodo (positivo) e qual é o cátodo (negativo): o terminal mais comprido (ânodo) é conectado à alimentação de energia positiva (3,3 V) e o terminal com o lado chanfrado (cátodo) vai para o terra.

Se você conectar seu LED da forma errada, isso não o danificará (a menos que você coloque correntes muito elevadas nele). Entretanto, é essencial que você sempre coloque um resistor em série com o LED, para garantir que a corrente certa chegue ao LED. Você pode danificar permanentemente o LED se não o fizer.

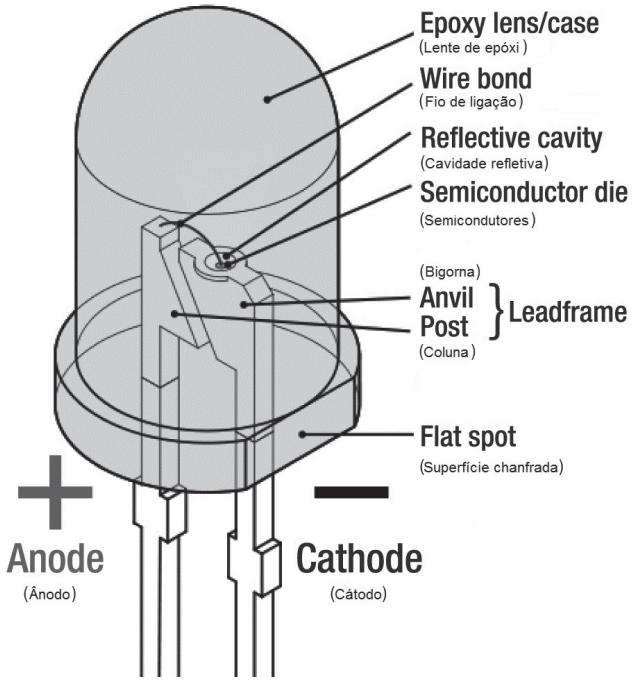


Figura 2.5 – Componentes de um LED (imagem cortesia de Inductiveload do Wikimedia Commons).

Note que você também pode obter LEDs bicolores ou tricolores, os quais têm diversos terminais saindo deles. Um LED RGB oferece um LED vermelho, verde e azul (daí RGB, de Red, Green e Blue) em um único pacote. Esse LED tem quatro terminais; um será um ânodo ou cátodo usual (comum a todos os três LEDs), e os outros terminais irão para o ânodo ou cátodo de um LED individual. Ajustando os valores de brilho dos canais R, G e B do LED RGB você pode obter a cor que quiser (o mesmo efeito pode ser obtido se você utilizar três LEDs separados, sendo um vermelho, um verde e outro azul).

Agora que você sabe como funcionam os componentes e o código deste projeto, vamos experimentar algo um pouco mais interessante.

Projeto 2 – Sinalizador de código Morse S.O.S.

Para este projeto, você reutilizará o circuito que preparamos para o projeto 1 (por isso não será necessário analisar o hardware), mas você utilizará um código diferente para fazer com que o LED sinalize as letras S.O.S., sinal de socorro internacional em código Morse. O código Morse é um tipo de codificação de caracteres que transmite letras e números utilizando padrões de ligado e desligado. Portanto, ele é muito adequado ao seu sistema digital, uma vez que você pode acender e apagar o LED no padrão necessário para soletrar uma palavra ou série de caracteres. Nesse caso, o padrão S.O.S. é formado de três pontos (sinais curtos), seguidos por três traços (sinais longos), seguidos por três pontos novamente.