
apostila do aluno

# Sumário


**1 Conceitos básicos** **7**
- 1.1 O Projeto Arduino . . . 8
- 1.2 Instalação do *software* . . . 9
- 1.3 Primeiro projeto . . . 10
  - 1.3.1 Cálculos de resistência . . . 12
  - 1.3.2 Alimentação do circuito . . . 13
- 1.4 Bibliotecas e *shields* . . . 13
- 1.5 Integração com o PC . . . 14
- 1.6 Portas analógicas e digitais . . . 15
  - 1.6.1 Portas digitais . . . 15
  - 1.6.2 Portas analógicas . . . 18

**2 Fundamentos de Eletrônica** **21**
- 2.1 Resistores e Lei de Ohm . . . 22
  - 2.1.1 Resistores em série . . . 22
  - 2.1.2 Resistores em paralelo . . . 22
  - 2.1.3 Código de cores . . . 23
  - 2.1.4 Divisor de tensão . . . 23
- 2.2 Capacitores e indutores . . . 23
  - 2.2.1 Capacitores . . . 24
  - 2.2.2 Indutores . . . 24
- 2.3 Diodos . . . 25
- 2.4 Transistores . . . 25
  - 2.4.1 Utilização de transistores com relés . . . 26
  - 2.4.2 Ponte-H . . . 27

**3 Eletrônica Digital** **29**
- 3.1 Introdução . . . 30
- 3.2 Portas lógicas . . . 30
  - 3.2.1 Tabela-verdade . . . 30
  - 3.2.2 Representação das operações . . . 31
  - 3.2.3 Funções lógicas compostas . . . 31

**4 Fazendo barulho com o Arduino** **33**

**5 Armazenando na EEPROM** **35**

# Prefácio

Essa apostila é destinada aos alunos que realizaram o Curso de Arduino[1], tendo como premissa explicar em mais detalhes temas abordados em aula para que os alunos não se preocupem com anotações durante os experimentos.

Seu conteúdo (com exceção das fotos de terceiros, devidamente citadas) está disponível através da licença **Creative Commons Atribuição-Uso não-comercial-Compartilhamento pela mesma licença 3.0 Unported**, que está disponível nas formas compacta e completa nos seguintes endereços:

`http://creativecommons.org/licenses/by-nc-sa/3.0/deed.pt_BR`
`http://creativecommons.org/licenses/by-nc-sa/3.0/legalcode`


Caso possua correções, sugestões ou mesmo queira contribuir escrevendo essa apostila, sinta-se livre para entrar em contato – terei um imenso prazer em receber sua contribuição!

# *Download*

Essa apostila está disponível para *download* através do *site* do Curso de Arduino[1]. Acesse o site para verificar novas versões no seguinte endereço:

`http://www.CursoDeArduino.com.br/apostila`

# O Autor

Álvaro Justen, também conhecido como "Turicas", é o criador do Curso de Arduino[1] e autor dessa apostila. Fã de carteirinha de software livre (usuário há mais de 12 anos), sempre participa de eventos (palestrando ou organizando) e grupos de usuários, além de contribuir com o desenvolvimento de diversos projetos. Foi responsável pela criação do grupo de usuários de Arduino do Rio de Janeiro[2], onde são realizados encontros mensais para discutir sobre a plataforma.

Está finalizando sua graduação em Engenharia de Telecomunicações pela Universidade Federal Fluminense (Niterói/RJ), onde já desenvolveu diversas atividades de pesquisa, ensino e extensão (muitas ligadas ao Arduino); é programador Python[3], tendo criado e contribuído com diversos projetos nessa linguagem; entusiasta de metodologias ágeis e Coding Dojo[4], sendo o responsável por trazer a prática a Niterói.

Álvaro atualmente possui uma empresa que ministra cursos de Arduino por todo o Brasil e desenvolve projetos utilizando a plataforma para diversas empresas. Além disso, desenvolve bibliotecas abertas para o Arduino e publica artigos com dicas e projetos em seu blog.

# Contato


- E-mail: `alvaro@CursoDeArduino.com.br`
- Blog: `http://blog.justen.eng.br/`
- Twitter: `http://twitter.com/turicas`
- Telefone: +55 21 9898-0141


[1] `http://www.CursoDeArduino.com.br/`
[2] `http://ArduInRio.cc/`
[3] `http://www.python.org/`
[4] `http://dojorio.org/`

# Capítulo 1

# Conceitos básicos

## 1.1 O Projeto Arduino

Arduino[1] é um projeto que engloba *software* e *hardware* e tem como objetivo fornecer uma plataforma fácil para prototipação de projetos interativos, utilizando um microcontrolador. Ele faz parte do que chamamos de computação física: área da computação em que o *software* interage **diretamente** com o *hardware*, tornando possível integração fácil com sensores, motores e outros dispositivos eletrônicos.

A parte de *hardware* do projeto, uma placa que cabe na palma da mão, é um computador como qualquer outro: possui microprocessador, memória RAM, memória *flash* (para guardar o *software*), temporizadores, contadores, dentre outras funcionalidades. Atualmente, o projeto está na versão Uno, porém muitos Arduinos encontrados hoje são da versão Duemilanove (2009, em italiano), que possui um *clock* de 16MHz, 2kB de memória RAM, 32kB de memória *flash*, 14 portas digitais e 6 entradas analógicas.

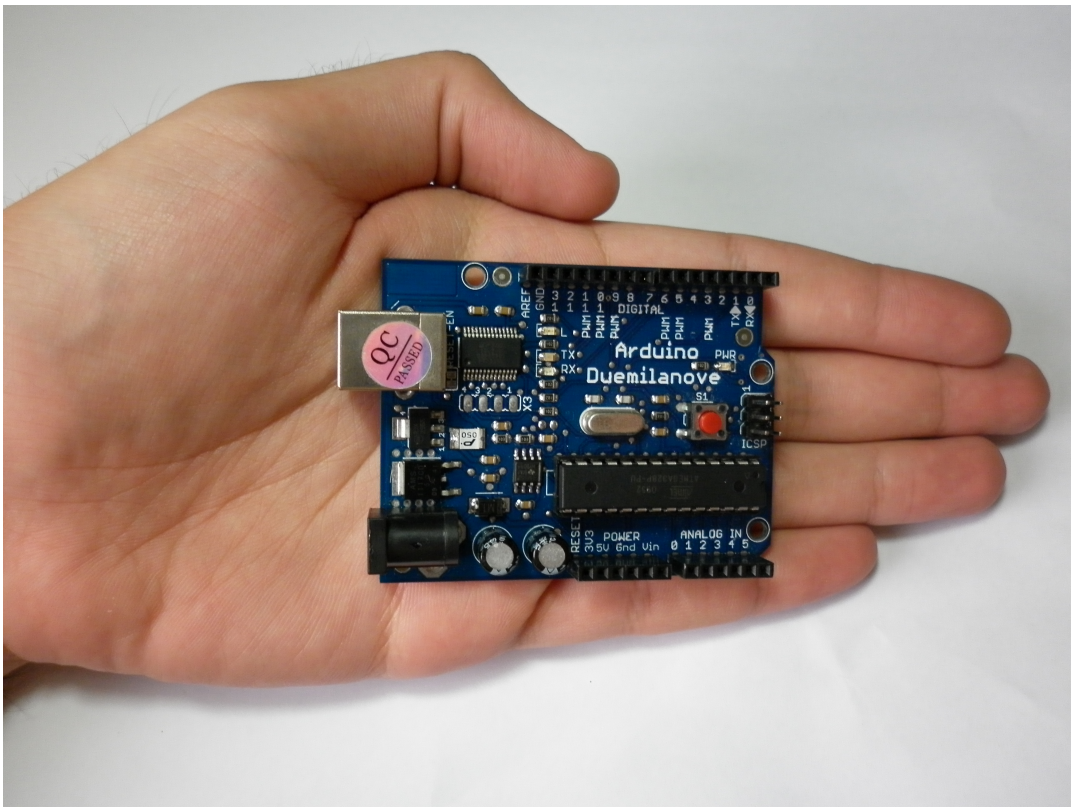



Figura 1.1: Foto do *hardware* de um *Arduino Duemilanove*

A principal diferença entre um Arduino e um computador convencional é que, além ter menor porte (tanto no tamanho quanto no poder de processamento), o Arduino utiliza dispositivos diferentes para entrada e saída em geral. Por exemplo: em um PC utilizamos teclado e mouse como dispositivos de entrada e monitores e impressoras como dispositivos de saída; já em projetos com o Arduino os dispositivos de entrada e saída são circuitos elétricos/eletrônicos.

Como a interface do Arduino com outros dispositivos está mais perto do meio físico que a de um PC, podemos ler dados de sensores (temperatura, luz, pressão etc.) e controlar outros circuitos (lâmpadas, motores, eletrodomésticos etc.), dentre outras coisas que não conseguiríamos diretamente com um PC. A grande diferença com relação ao uso desses dispositivos, no caso do Arduino, é que, na maior parte das vezes, nós mesmos construímos os circuitos que são utilizados, ou seja, não estamos limitados apenas a produtos existentes no mercado: o limite é dado por nosso conhecimento e criatividade!

O melhor de tudo nesse projeto é que seu *software*, *hardware* e documentação são abertos. O *software* é livre (GNU GPL[2]), o *hardware* é totalmente especificado (basta entrar no site e baixar os esquemas) e a documentação está disponível em Creative Commons[3] – os usuários podem colaborar (seja escrevendo documentação, seja traduzindo) através da wiki!

[1] `http://www.arduino.cc/`
[2] `http://www.gnu.org/licenses/gpl.html`
[3] `http://creativecommons.org/licenses/`

## 1.2 Instalação do *software*

Para criar um projeto com o Arduino, basta comprar uma placa Arduino (utilizaremos o Arduino Duemilanove como exmplo) – que custa em torno de US$30 no exterior e por volta de R$100 no Brasil –, fazer download da interface integrada de desenvolvimento (IDE)[4] e ligar a placa à porta USB do PC.

Como qualquer computador, o Arduino precisa de um *software* para executar comandos. Esse *software* será desenvolvido na Arduino IDE em nosso PC, utilizando a linguagem C++. Após escrever o código, o compilaremos e então faremos o envio da versão compilada à memória flash do Arduino, através da porta USB. A partir do momento que o *software* é gravado no Arduino não precisamos mais do PC: o Arduino, como é um computador independente, conseguirá sozinho executar o *software* que criamos, desde que seja ligado a uma fonte de energia.

Antes de iniciar nosso projeto precisamos então instalar a IDE. Vamos lá:

- **Ubuntu GNU/Linux 10.10**: Basta executar em um terminal:

```
sudo aptitude install arduino
```

  ou procurar pelo pacote "arduino" no Synaptic (menu Sistema → Administração → Gerenciador de pacotes Synaptic).

- **Ubuntu GNU/Linux** (anterior a 10.10): Consulte a página de instalação do Arduino em Ubuntu[5].

- **Outras distribuições GNU/Linux**: Consulte a página de instalação em outras distribuições GNU/Linux[6].

- **Microsoft Windows**: Consulte a página de instalação para as variadas versões do Microsoft Windows[7].

- **Apple Mac OS X**: Consulte a página de instalação para o Mac OS X[8].

Após a instalação, abra a IDE (no Ubuntu GNU/Linux ela estará disponível no menu *Aplicativos* → *Eletrônica* → *Arduino IDE*). A seguinte tela será mostrada:

[4] http://arduino.cc/en/Main/Software
[5] http://www.arduino.cc/playground/Linux/Ubuntu
[6] http://www.arduino.cc/playground/Learning/Linux
[7] http://www.arduino.cc/en/Guide/Windows
[8] http://www.arduino.cc/en/Guide/MacOSX

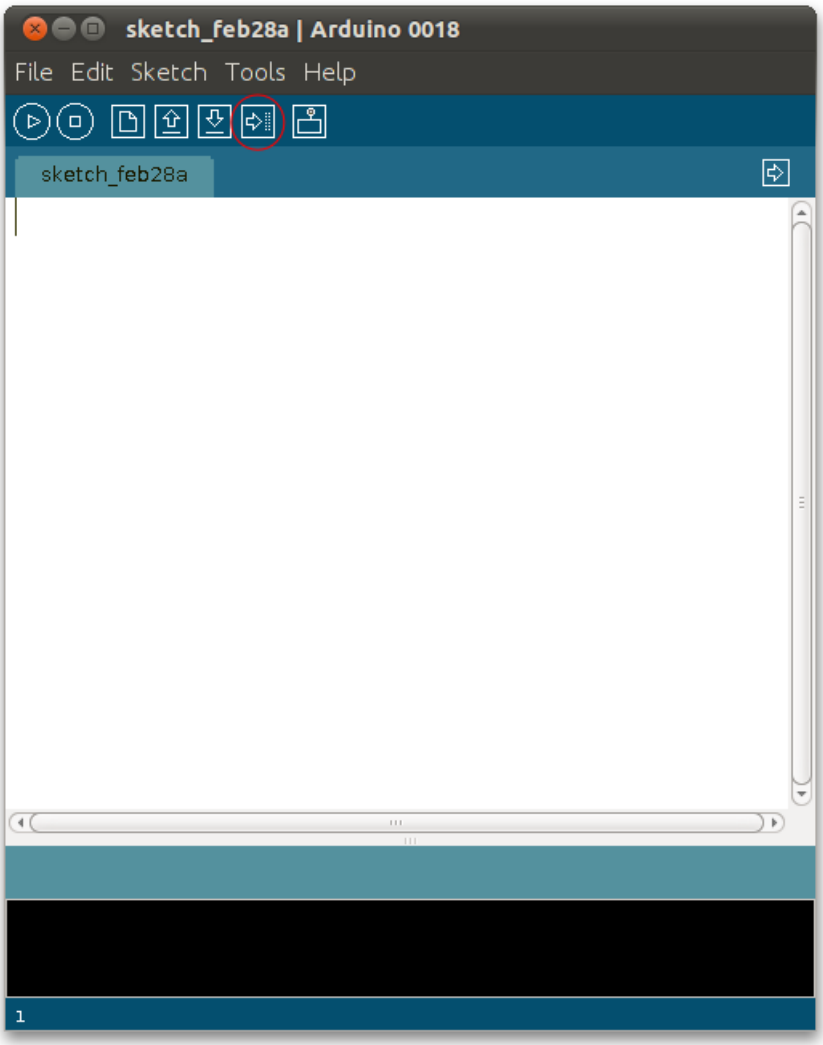



Figura 1.2: Arduino IDE versão 0018 rodando no Ubuntu GNU/Linux 10.10

## 1.3 Primeiro projeto

Quando ensinamos linguagens de programação novas, geralmente o primeiro exemplo é um *hello world*. Como o Arduino não vem por padrão com um display, nosso primeiro exemplo será fazer um LED piscar – e por isso será chamado *blink*. Nosso LED ficará aceso durante um segundo e apagado durante outro segundo e então recomeçará o ciclo. Abra a IDE e digite o seguinte código:

```
void setup() {
    pinMode(13, OUTPUT);
}
void loop() {
    digitalWrite(13, HIGH);
    delay(1000);
    digitalWrite(13, LOW);
    delay(1000);
}
```

Chamamos um código feito para Arduino de *sketch* e o salvamos em um arquivo com a extensão `.pde`. Com nosso *sketch* pronto, bastará conectar o Arduino na porta USB e clicar no botão *upload* (segundo, da direita para a esquerda – destacado na figura acima). Após o processo, será vista a mensagem *Done uploading* na IDE e então o *sketch* estará rodando no Arduino, ou seja, o LED acenderá e apagará, de 1 em 1 segundo. Vamos agora à explicação do processo:

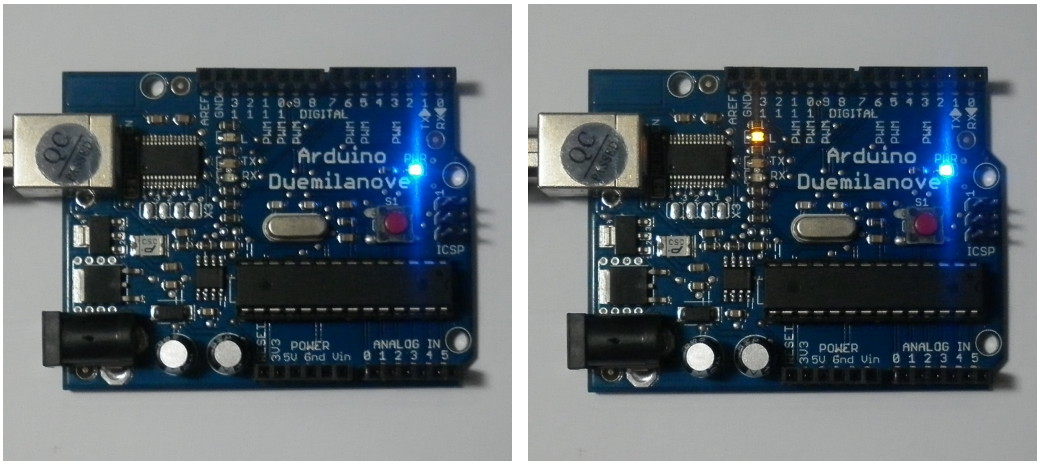
O Arduino possui 14 portas digitais, que podemos utilizar como entrada ou saída. Nesse caso, vamos utilizar a porta de número 13 como saída, dessa forma, podemos controlar quando a porta ficará com 5V ou quando ficará com 0V – internamente o Arduino possui um LED conectado à porta 13 e, por isso, teremos como "ver" nosso *software* funcionando.

Para que nosso *software* funcione corretamente no Arduino, precisamos criar duas funções específicas: `setup` e `loop`. A função `setup` é executada assim que o Arduino dá boot, já a função `loop` fica sendo executada continuamente (em *loop*) até que o Arduino seja desligado. Como as portas digitais são de entrada ou saída, definimos então dentro da função `setup` que a nossa porta 13 é uma porta de saída – fazemos isso através da chamada à função `pinMode`, que já vem na biblioteca padrão do Arduino.

Depois de configurarmos corretamente a porta 13 como saída, precisamos acender e apagar o LED que está conectado a ela. Para alterar a tensão na porta, utilizamos a função `digitalWrite` (que também está na biblioteca padrão do Arduino); passamos para essa função a porta que queremos alterar a tensão e o novo valor de tensão (`HIGH` = 5V, `LOW` = 0V). Depois das chamadas para acender e apagar o LED, chamamos a função `delay` passando o parâmetro `1000` – o que essa função faz é esperar um tempo em milisegundos para então executar a próxima instrução.

Deve-se ressaltar que a IDE Arduino inclui automaticamente todas as bibliotecas que utilizamos. Se você está acostumado com C/C++, note que não precisamos digitar as diretivas `include` para arquivos como o `stdlib.h`, por exemplo. Tudo é feito de forma automática para facilitar o desenvolvimento do projeto!

Como o Arduino já vem com um LED internamente conectado à porta 13, não precisaremos de circuitos externos para que esse projeto funcione, ou seja, bastará fazer *upload* daquele código e já teremos o resultado esperado:




(a) LED apagado (b) LED aceso

Figura 1.3: Arduino Duemilanove rodando o exemplo *Blink*

Porém, se quisermos acender um LED externo à placa, podemos conectá-lo diretamente à porta 13. Podemos utilizar um LED de 5mm que acende com 2,5V. O problema, nesse caso, se dá por conta da porta digital: ela assume ou a tensão 0V, ou a tensão 5V – e caso coloquemos 5V no LED ele irá queimar.

Para solucionar esse problema precisamos ligar algum outro componente que seja responsável por dividir parte dessa tensão com o LED, para que ele não queime, então utilizaremos um resistor. Portanto, ligamos um resistor de 120Ω em série com o LED, o resistor à porta 13 e o LED à porta GND – *ground* ou terra –, como na Figura 1.4.

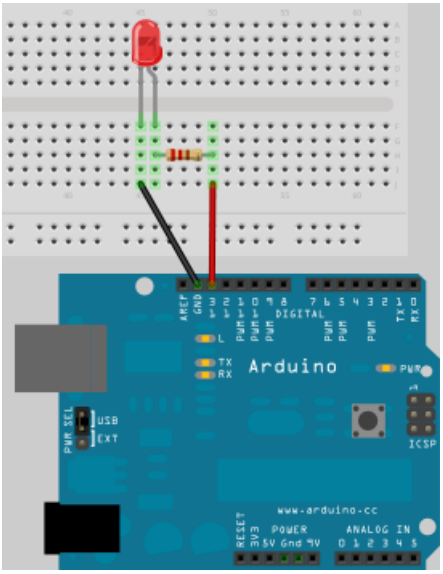



Figura 1.4: Utilizando um LED externo para o exemplo *Blink*

Não precisamos fazer nenhuma alteração no *software* para que esse circuito funcione: basta ligar o Arduino na porta USB do computador, para que o computador dê energia ao circuito, e então veremos o LED externo piscar juntamente com o LED interno. Em vez do LED, poderíamos controlar outros componentes, como motores, eletrodomésticos etc.

### 1.3.1 Cálculos de resistência

Para chegar ao valor de 120Ω acima, precisei fazer algumas contas (e arredondamentos). Vamos aprender agora a calcular o valor dos resistores que precisamos utilizar. Se precisarmos acender um LED verde, que é alimentado com tensão de 2,2V e corrente de 20mA através do Arduino, precisaremos de um resistor, como já vimos, já que o Arduino só consegue fornecer ou 0V ou 5V. Colocaremos o resistor em série com o LED, e com isso podemos concluir que:

- A tensão total (soma das tensões no resistor e no LED) será de 5V, ou seja: $V_{LED} + V_R = 5V$
- A corrente total que passa pelo resistor e pelo LED é igual, ou seja, 20mA, ou seja: $I_{LED} = I_R = 20mA$
- Precisamos colocar uma tensão de 2,2V no LED, ou seja: $V_{LED} = 2,2V$

Sabendo desses detalhes, podemos concluir que a tensão no resistor será de: $V_R = 5V - V_{LED} \therefore V_R = 5V - 2,2V \Rightarrow V_R = 2,8V$. Como $I_R = 20mA$ e $V_R = 2,8V$, podemos calcular o valor da resistência $R$ do resistor que iremos utilizar através da Lei de Ohm:

$$V = RI$$

Assim, temos: $2,8V = R0,020A \therefore R = \frac{2,8V}{0,020A} \Rightarrow R = 140\Omega$

Depois de feito o cálculo, podemos generalizar com a seguinte fórmula:

$$R = \frac{V_{fonte} - V_{LED}}{I_R}$$

Para o LED verde, precisamos de um resistor de 140Ω, porém não existem resistores com esse valor para venda – os valores são pré-definidos[9]. Dada essa situação, temos duas alternativas:

- Utilizar um resistor de maior resistência e limitar mais a corrente (que fará com que o LED brilhe menos); ou
- Associar dois ou mais resistores em série ou paralelo para conseguir o valor.

Geralmente escolhemos um resistor de valor próximo, já que uma alteração pequena de corrente não causará danos ao dispositivo, porém em alguns casos precisaremos combinar resistores de valores diferentes para conseguir o valor equivalente – esse tema será explicado em mais detalhes no próximo capítulo.

[9]Saiba mais em `http://www2.eletronica.org/hack-s-dicas/valores-comerciais-para-resistores-capacitores-e-indutores/`

### 1.3.2 Alimentação do circuito

Internamente, o circuito do Arduino é alimentado com uma tensão de 5V. Quando ligamos o Arduino em uma porta USB do PC, o próprio PC, através do cabo USB, alimenta o Arduino. Porém nem sempre temos um PC por perto; para esses casos, podemos utilizar uma outra fonte de energia de 5V (a fonte deve ser ligada diretamente nos pinos 5V e GND do Arduino).

Como não possuímos pilhas/baterias em abundância no mercado com tensão de 5V, fica complicado alimentar um Arduino dessa forma alternativa – se tivermos uma tomada de 127/220VAC por perto, poderíamos ligar uma fonte AC/DC (essas sim, existem aos montes). Para resolver esse problema, o Arduino possui um regulador de tensão que aceita tensões de 7 a 12V (na verdade, ele consegue funcionar com tensões entre 6 e 20V, apesar de não ser recomendado). Com o regulador de tensão podemos combinar pilhas em série, utilizar uma bateria de 9V ou mesmo baterias de carros, motos e no-breaks (12V).

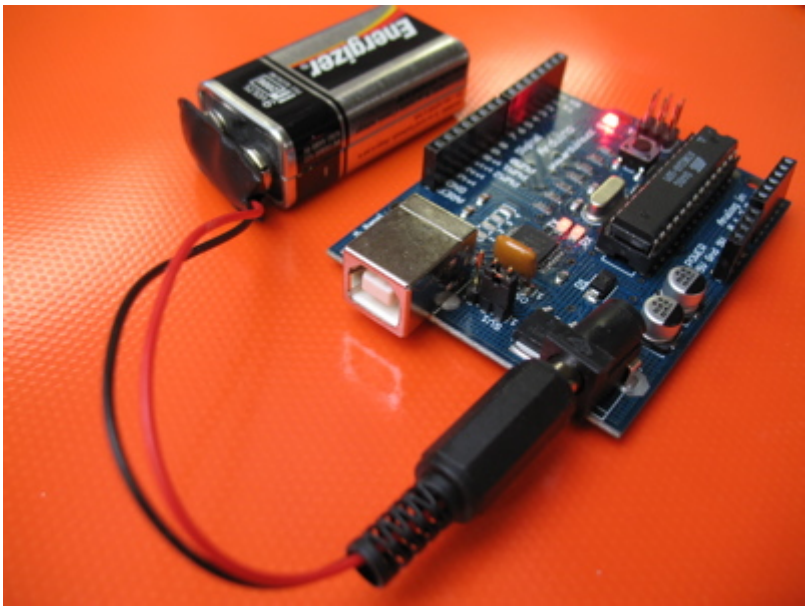



Figura 1.5: Arduino alimentado por uma bateria de 9V
Retirado de `http://www.arduino.cc/playground/Learning/9VBatteryAdapter`

## 1.4 Bibliotecas e *shields*

Assim como a IDE já vem com diversas funções pré-definidas, o Arduino possui outras bibliotecas para controle de servomotores, displays LCD, geração de áudio, recepção de sinais de sensores e outros dispositivos (como teclado PS/2), dentre muitas outras coisas! E quem pensa que essa estensibilidade toda se restringe à parte de *software* está muito enganado: o Arduino possui o que chamamos de *shields*, que são placas que se acoplam à placa original, agregando funcionalidades à mesma.

Existem *shields* dos mais variados tipos, para as mais diversas funções. Alguns servem como entrada, outros como saída, e ainda outros como entrada e saída. Com os *shields* conseguimos, por exemplo, fazer o Arduino se comunicar numa rede Ethernet, ou ainda transmitir dados para qualquer dispositivo via Bluetooth, Wi-Fi ou Zigbee. Existem *shields* com circuitos integrados prontos para controlarmos motores sem que precisemos nos preocupar com complicações eletrônicas envolvidas, outros possuem leitor de cartão SD, acelerômetro, GPS e diversos outros sensores que podem gerar dados importantes para o *software* que está rodando no microcontrolador.

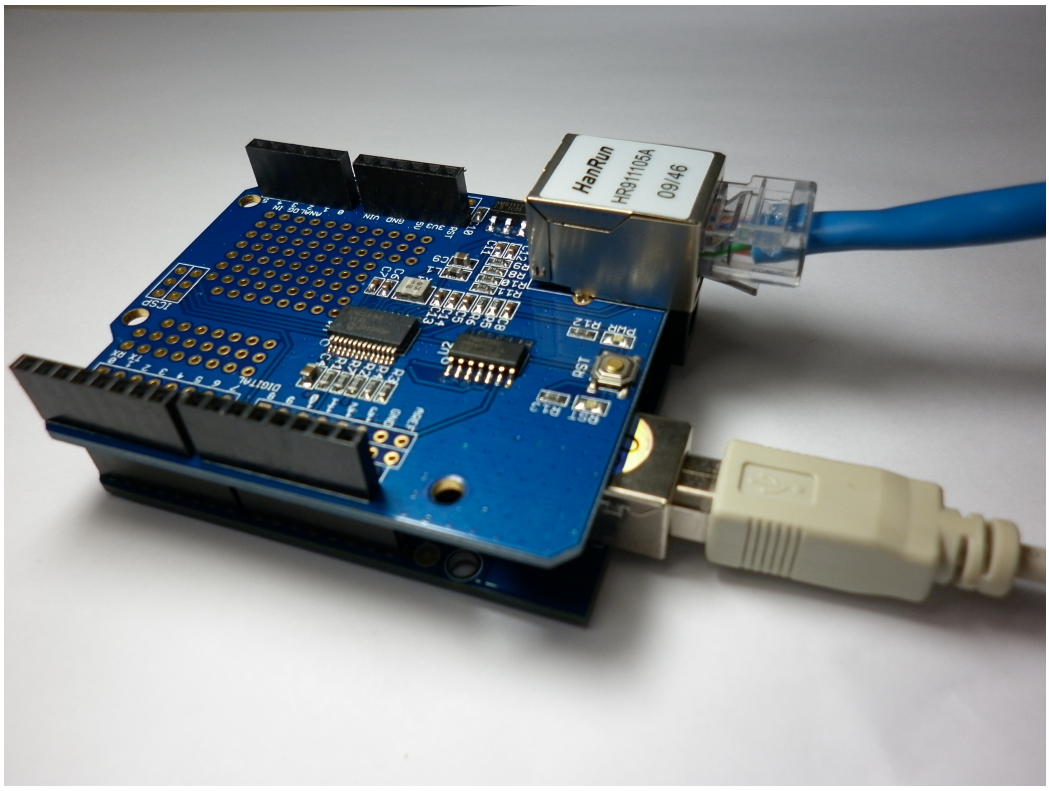



Figura 1.6: Arduino Duemilanove com *shield* Ethernet

Existem também outros *shields* mais elaborados, que são sistemas completos. Um deles, por exemplo, cria uma plataforma para desenvolvimento de jogos no Arduino: o Video Game Shield[10], que possui uma saída RCA e duas entradas para controles Numchuck do Nintendo Wii. Além do *hardware*, existe uma biblioteca para ser utilizada em conjunto, que já possui várias funções pré-programadas para fazermos desenhos na televisão e capturar os dados dos movimentos nos controles.

## 1.5 Integração com o PC

Apesar de o Arduino ser um computador independente, em alguns casos podemos nos aproveitar de um PC por perto e explorar outra funcionalidade muito boa do projeto: o Arduino consegue conversar com o computador através da porta USB. Isso nos permite desenvolver um *software* que roda no PC e se comunica com o *software* que roda no Arduino, o que nos abre um mar de possibilidades! Podemos, por exemplo, criar um *software* em Python[11] que recebe os dados de um sensor, via USB (através do Arduino), e envia para algum banco de dados na Internet – assim teremos, de certa forma, nosso Arduino online, enviando dados para o mundo, através de um PC!

Existem inúmeros projetos interessantes que fazem interface entre linguagens de programação e o Arduino – existem implementações para Python, Ruby, Java, C, dentre outras linguagens. E não para por aí: além de o *software* que roda no PC receber dados, ele pode também enviar dados, controlando o Arduino! Dessa forma, podemos, por exemplo, receber dados da Web e enviar comandos ao Arduino, baseados nesses dados.

Um exemplo de aplicação que utiliza a porta USB para comunicação do Arduino com o PC é o projeto Arduinoscope[12], que tem como finalidade criar um osciloscópio, onde podemos ver em tempo real, no PC, gráficos das tensões que estão ligadas às portas analógicas do Arduino. Outro exemplo é um projeto que criei, onde controlo um carrinho através de Wi-Fi, o Turiquinhas[13].

Fica a dica para quem quer começar um projeto Arduino assistido por computador de maneira fácil: vale a pena estudar um protocolo de comunicação e controle chamado Firmata[14], cuja implementação está disponível

[10] `http://www.wayneandlayne.com/projects/video-game-shield/`
[11] `http://www.python.org/`
[12] `http://code.google.com/p/arduinoscope/`
[13] `http://www.justen.eng.br/Turiquinhas`
[14] `http://www.firmata.org/`

para várias linguagens (e já vem por padrão um exemplo na IDE do Arduino). Ele facilita o processo de aquisição de dados e controle da placa.

Além das opções citadas acima, existe uma linguagem chamada Processing[15], parecidíssima com a linguagem que utilizamos no Arduino, que consegue se comunicar com o Arduino via USB e é utilizada para criar imagens, animações e interações no PC, a partir dos dados vindos da comunicação com o Arduino.

## 1.6 Portas analógicas e digitais

O Arduino possui dois tipos de portas de entrada: analógicas e digitais. Além disso, as portas digitais também servem como portas de saída, funcionando com dois tipos básicos de saída: saída digital comum e saída PWM – o PWM pode ser utilizado para simular uma saída analógica, dentre outras coisas.

### 1.6.1 Portas digitais

Utilizamos as portas digitais quando precisamos trabalhar com valores bem definidos de tensão. Apesar de nem sempre ser verdade, geralmente trabalhamos com valores digitais binários, ou seja, projetamos sistemas que utilizam apenas dois valores bem definidos de tensão. Existem sistemas ternários, quaternários, mas focaremos no binário, já que é esse o utilizado pelo Arduino.

Como o sistema é binário, temos que ter apenas duas tensões. São elas: 0V e 5V. Dessa forma, as portas digitais do Arduino podem trabalhar apenas com essas duas tensões – e o *software* que desenvolveremos poderá requisitar ao microcontrolador do Arduino que:

- Coloque uma determinada porta em 0V;
- Coloque uma determinada porta em 5V;
- Leia o valor de uma determinada porta (terá 0V ou 5V como resposta).

O Arduino Duemilanove possui 14 portas digitais que estão destacadas na figura a seguir:

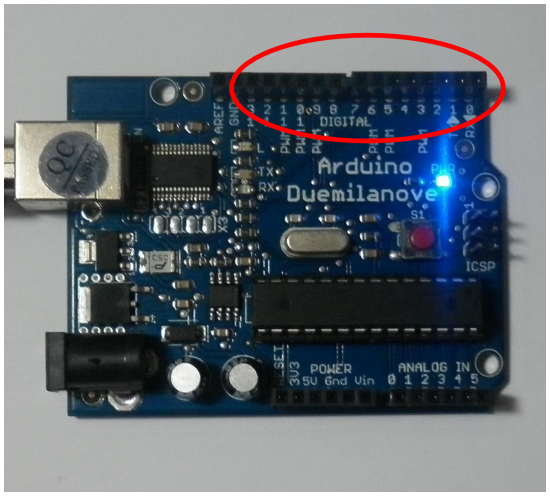



Figura 1.7: Portas digitais do Arduino, de 13 a 0

Apesar de ser possível, não é recomendável utilizar as portas 0 e 1 pois elas estão diretamente ligadas ao sistema de comunicação do Arduino (pinos RX e TX – recepção e transmissão, respectivamente) e, por isso, seu uso pode conflitar com o *upload* do *software*. Caso queira utilizá-las, certifique-se de desconectar quaisquer circuitos conectados a ela no momento do *upload*.

Utilizaremos as funções `digitalRead` e `digitalWrite` para ler e escrever, respectivamente, nas portas digitais. A função `digitalWrite` já foi exemplificada em nosso exemplo *Blink*, onde acendemos e apagamos um LED ao alternar a tensão da porta 13 entre 5V e 0V, com intervalo de 1 segundo. Para exemplificar a função `digitalRead` utilizaremos um botão, como no diagrama a seguir:

[15] `http://www.processing.org/`

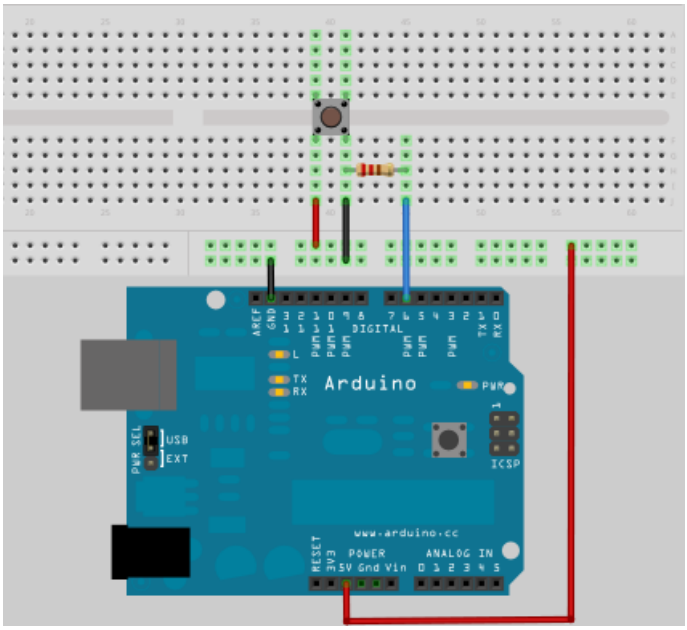



Figura 1.8: Esquema elétrico ligando um botão ao Arduino

Montado o circuito acima, vamos programar o Arduino com o seguinte código:

```
#define BOTAO 2
#define LED 13

void setup() {
  pinMode(LED, OUTPUT);
  pinMode(BOTAO, INPUT);
}

void loop() {
  if (digitalRead(BOTAO) == HIGH) {
    digitalWrite(LED, LOW);
  }
  else {
    digitalWrite(LED, HIGH);
  }
}
```

A função `digitalRead` nos retorna o valor correspondente à tensão que está na porta que passamos entre parenteses. Em nosso exemplo, utilizamos a porta `BOTAO` (que na verdade é uma constante, definida através da diretiva `#define`), cujo valor é 2. O valor retornado é uma constante, mapeado da seguinte forma:

- `HIGH`, caso a tensão na porta seja 5V;
- `LOW`, caso a tensão na porta seja 0V;

O que o programa faz, então, é apagar o LED caso o botão esteja pressionado e acendê-lo caso não esteja. Fica como exercício entender o código a seguir, que é uma otimização do anterior e possui mesma funcionalidade:

```
#define BOTAO 2
#define LED 13

void setup() {
  pinMode(LED, OUTPUT);
  pinMode(BOTAO, INPUT);
}
```

```
void loop() {
  digitalWrite(LED, !digitalRead(BOTAO));
}
```

Dica: o caractere "!", em linguagem C, significa *not* e tem como finalidade negar a expressão que segue à sua direita.

### PWM

PWM significa Modulação por Largura de Pulso (*Pulse-Width Modulation*, do Inglês) e consiste em manipularmos a razão cíclica de um sinal (conhecida do Inglês como *duty cycle*) a fim de transportar informação ou controlar a potência de algum outro circuito. Basicamente, teremos um sinal digital que oscila entre 0V e 5V com determinada frequência (o Arduino trabalha com um padrão perto de 500Hz) – funciona como se fosse um *clock*, porém os tempos em que o sinal permanece em 0V e 5V podem ser diferentes. *Duty cycle* é a razão do tempo em que o sinal permanece em 5V sobre o tempo total de uma oscilação. A Figura 1.9 ilustra melhor esse conceito:

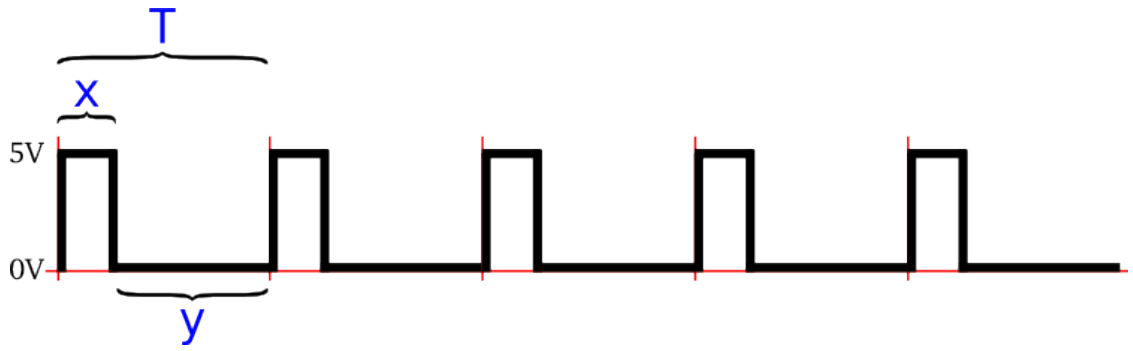



Figura 1.9: Sinal PWM com *duty cycle* de 25%

Assim, temos:

$$duty\ cycle = \frac{x}{x+y} = \frac{x}{T}$$

O que controlamos através de *software* é justamente a *duty cycle*, ou seja, o percentual de tempo em que o sinal fica em 5V. Dessa forma, podemos utilizar essa técnica para limitar a potência de algum circuito. Por exemplo, considere que um LED $L_1$ seja alimentado o tempo inteiro por um sinal constante de 5V; já o LED $L_2$ é alimentado pelo sinal PWM acima (*duty cycle* de 25%). Através de um cálculo simples de potência podemos notar que o LED $L_2$ consumirá apenas 25% da potência do primeiro.

Infelizmente, por limitações de *hardware*, o Arduino não possui PWM em todas as portas digitais: apenas as portas 3, 5, 6, 9, 10 e 11 são privilegiadas e podem utilizar esse recurso. Para exemplificar o uso de controle de potência de um circuito utilizando PWM vamos utilizar um LED em série com um resistor ligados à porta 11 (o circuito é o mesmo do experimento *Blink*, só vamos mudar a porta) e o seguinte código:

```
#define LED 11
void setup() {
    pinMode(LED, OUTPUT);
}

void loop() {
    for (int i = 0; i < 255; i++) {
        analogWrite(LED, i);
        delay(30);
    }
}
```

Na função *loop* acima temos um laço `for`, que conta de 0 a 255 (armazenando o valor do contador na variável $i$), chamando a função `analogWrite` (passando como parâmetros a porta do LED (11) e $i$) e esperando por 30 milisegundos a cada iteração.

A função `analogWrite` (apesar de estarmos utilizando uma porta digital) é responsável pelo PWM e recebe como parâmetros a porta e um valor entre 0 e 255 – esse valor corresponde ao *duty cycle*, onde 0 corresponde a 0% e 255 a 100%. Quando você rodar o código perceberá que o LED acende de maneira mais suave – cada etapa de luminosidade diferente corresponde a uma iteração do `for`. Fica como exercício para o leitor modificar o programa para que o LED, além de acender, também apague suavemente. :-)

### 1.6.2 Portas analógicas

Além das portas digitais o Arduino possui as portas analógicas. Ao contrário das portas digitais, as portas analógicas são apenas de entrada e nelas podemos ter como entrada infinitos valores de tensão (delimitados na faixa de 0V a 5V). Como os conversores analógico-digitais (ADC – *analog-digital converter*, do Inglês) do Arduino possuem 10 bits de precisão, a precisão das medições de tensão no Arduino é de por volta de 0,005V ou 5mV.

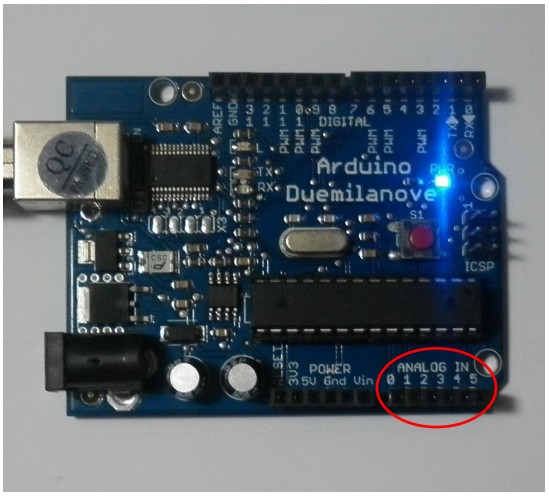



Figura 1.10: Portas analógicas do Arduino, de 0 a 5

Como os nomes de funções no Arduino são bastante intuitivos, utilizamos a função `analogRead` para ler valores analógicos – ao chamar a função passamos como argumento o número da porta que desejamos ler (de 0 a 5). Como exemplo vamos regular a luminosidade de nosso LED (utilizando PWM) através da quantidade de luz detectada por um resistor dependente de luz (ou LDR – *light dependent resistor*, do Inglês). Monte o circuito segundo a figura 1.11.

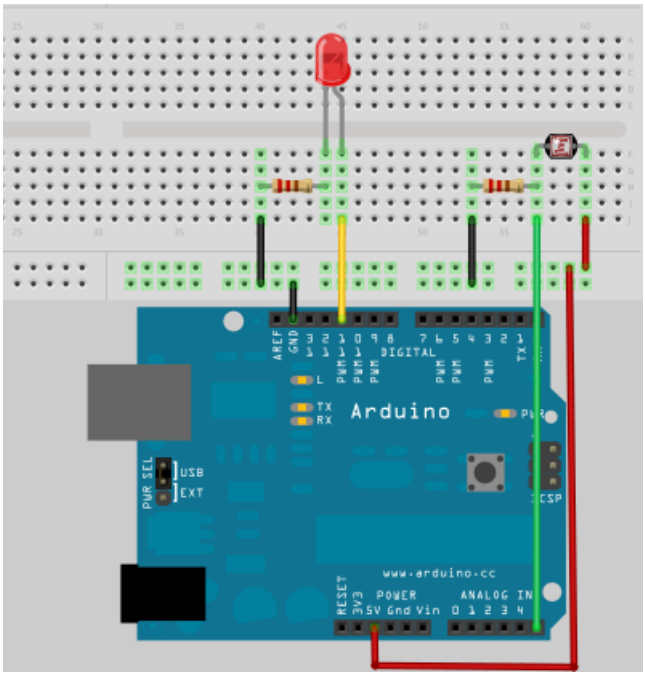



Figura 1.11: Circuito com LDR e LED

Como os conversores analógico-digital possuem 10 bits de precisão, a função `analogRead` nos devolve um valor entre 0 e 1023, onde 0 corresponde a uma leitura de 0V na porta analógica e 1023 corresponde a 5V (para valores intermediários, basta fazer uma regra de três simples, de maneira análoga com o PWM). Vamos carregar em nosso Arduino o seguinte código:

```
#define LED 11
#define LDR 5

void setup() {
  pinMode(LED, OUTPUT);
}

void loop() {
  int leitura = analogRead(LDR);
  int valorPWM = - 0.25 * leitura + 255;
  analogWrite(LED, valorPWM);
}
```

Os valores 0.25 e 255 da linha que definem a variável `valorPWM` devem ser calibrados conforme iluminação do ambiente, resistores utilizados e luminosidade desejada. Para o código acima, teremos o seguinte comportamento do valor que colocamos na porta PWM a partir dos valores lidos na porta analógica:

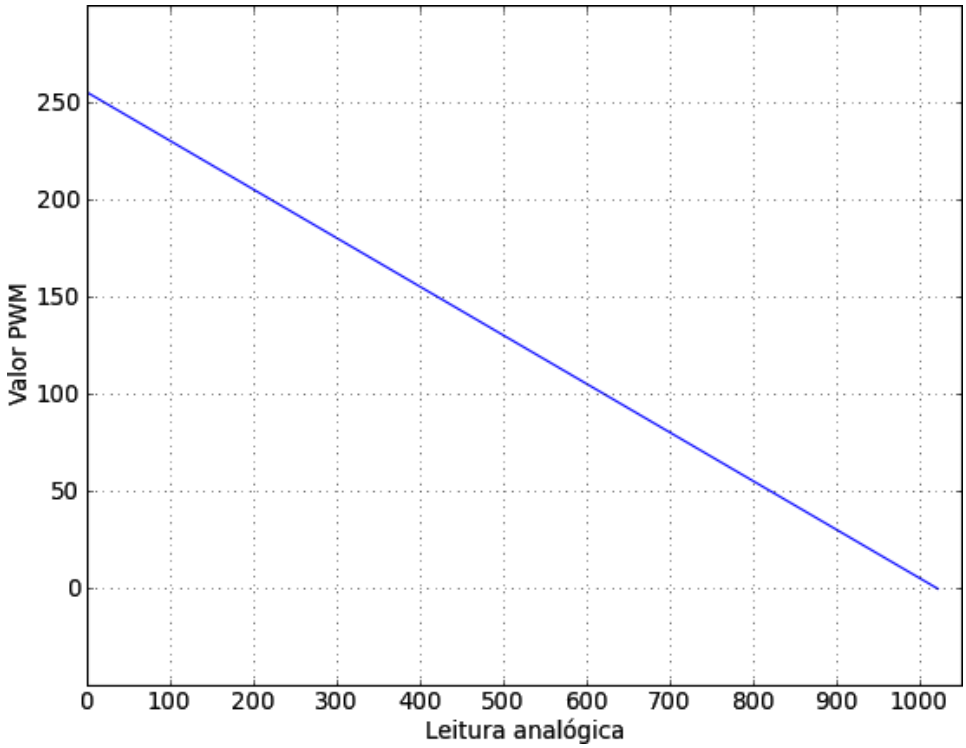



Figura 1.12: Gráfico da função PWM *versus* leitura analógica

Ficam três exercícios:

- Aprenda a utilizar um potenciômetro e o utilize para regular o brilho do LED, em vez do LDR;
- Volte o LDR para seu lugar anterior e utilize o potenciômetro para configurar os valores 0.25 e 255;
- Leia o *datasheet* do circuito integrado LM35 e monte um circuito parecido com o anterior, porém sensível a temperatura (e não mais a luz).

# Capítulo 2

# Fundamentos de Eletrônica

## 2.1 Resistores e Lei de Ohm

Resistores são dispositivos utilizados com a finalidade de transformar energia elétrica em energia térmica e/ou limitar a corrente elétrica em um circuito. Como o próprio nome sugere, eles têm como função oferecer uma resistência à passagem da corrente elétrica – medimos essa resistência através da unidade Ω (ohm).

Por consequência, eles causam uma queda de tensão na região do circuito onde se encontram – muitas vezes acabamos utilizando esse efeito para conseguirmos tensões intermediárias, caso as fontes de tensão do circuito não consigam fornecê-las. Sabendo-se a tensão e corrente em um resistor, podemos calcular sua resistência através da fórmula:

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

Por sua vez, a resistência pode ser calculada através das características do material resistivo:

$$R = \frac{\rho L}{A}$$

Onde $\rho$ é a resistividade do material, $L$ é seu comprimento e $A$ sua área, ou seja, a resistência é proporcional ao comprimento e indiretamente proporcional à área.

Se, para um determinado circuito, $V$ e $I$ têm uma relação linear, ou seja, $R$ é constante para inúmeros valores de $V$ e $I$, então chamamos o material (que possui resistência $R$) de ôhmico.

### 2.1.1 Resistores em série

Se possuirmos resistores em série em determinado circuito, podemos calcular a resistência equivalente do mesmo somando-se as resistências, através da fórmula:

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

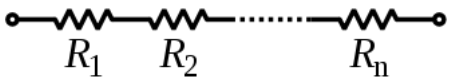



Figura 2.1: Resistores em série
Retirado de `http://en.wikipedia.org/wiki/Resistor`

### 2.1.2 Resistores em paralelo

Caso os resistores estejam em paralelo, a resistência equivalente será o inverso da soma dos inversos das resistências, como na fórmula a seguir:

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

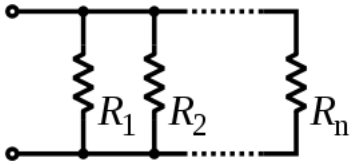



Figura 2.2: Resistores em paralelo
Retirado de `http://en.wikipedia.org/wiki/Resistor`

### 2.1.3 Código de cores

Os resistores possuem um código de cores que nos permite identificar qual sua resistência. Para isso, mapeamos as cores das diversas faixas do resistor e utilizamos a seguinte fórmula:

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

onde $a$, $b$ e $c$ são as primeiras faixas e $t$ a última faixa (geralmente prata ou dourada), que representa a tolerância.

**VALOR NOMINAL**

| COR | PRETO | MARROM | VERMELHO | LARANJA | AMARELO | VERDE | AZUL | VIOLETA | CINZA | BRANCO |
|---|---|---|---|---|---|---|---|---|---|---|
| VALOR | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

**VALOR DA TOLERÂNCIA**

| COR | DOURADO | PRATA | SEM COR |
|---|---|---|---|
| VALOR | ±5% | ±10% | ±20% |

Figura 2.3: Código de cores
Retirado de `http://pt.wikipedia.org/wiki/Resistor`

### 2.1.4 Divisor de tensão

Como citado acima, resistores criam uma queda de tensão na região do circuito em que estão. Utilizando esse efeito, podemos criar o que chamamos de divisores de tensão[1]: circuitos com resistores que, aplicada uma tensão, têm como saída uma fração (daí o nome divisores) dessa tensão de entrada.

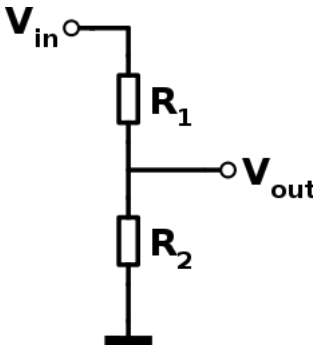



Figura 2.4: Circuito divisor de tensão

A partir do circuito acima, temos que:

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

Esse recurso é bastante útil quando precisamos medir um tensão maior do que nossos circuitos conseguem. Por exemplo: se quisermos medir uma tensão que varia de 9 a 12V no Arduino precisaremos colocá-la na faixa de 0 a 5V, já que as portas analógicas do Arduino trabalham nessa faixa menor. Para isso, poderíamos utilizar um divisor de tensão cujos valores de resistência resultassem em $V_{out} = \frac{V_in}{3}$. Dessa forma, os valores lidos em $V_{out}$ seriam de 3 a 4V.

Porém, atenção: caso precisemos conectar resistores ou outros circuitos resistivos no pino $V_{out}$, o cálculo das tensões muda e $V_{out}$ passa a depender das novas resistências do circuito.

## 2.2 Capacitores e indutores

Apesar de não estudarmos a fundo esses dois elementos básicos de circuitos, eles podem ser importantes no desenvolvimento de futuros projetos. Basicamente, capacitores e indutores armazenam energia (pense como

[1] http://pt.wikipedia.org/wiki/Divisor_de_tensão

uma bateria em que você carrega e descarrega de tempos em tempos, porém com capacidade bem limitada) e são bastante utilizados em filtros de sinais, estabilizadores de tensão, circuitos ressonantes (como transmissores e receptores de rádio), dentre outros.

### 2.2.1 Capacitores

Os capacitores são componentes que armazenam energia em forma de campo elétrico. São formados por duas placas metálicas com um dielétrico (isolante) no meio. A unidade de medida é o Farad (F), porém como 1 Farad é algo bem grande, comumente encontramos capacitores na casa dos $mF$ (miliFarad), $\mu F$ (microFarad) e $pF$ (picoFarad).


Figura 2.5: Capacitor eletrolítico bastante comum no mercado

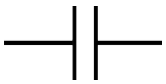


Figura 2.6: Representação de um capacitor em circuito elétrico

### 2.2.2 Indutores

Os indutores são componentes que armazenam energia em forma de campo magnético. Geralmente são formados por bobinas (fio enrolado) com um condutor no meio. A unidade de medida é o Henry (H).

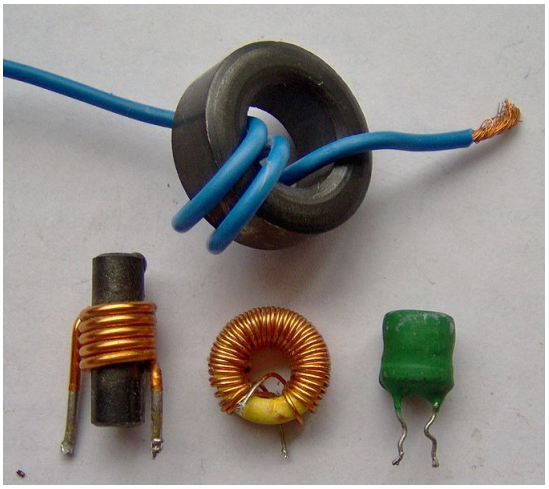


Figura 2.7: Indutores de vários tipos, comuns no mercado

Figura 2.8: Representação de um indutor em circuito elétrico

## 2.3 Diodos

Diodos são componentes que têm a capacidade de conduzir corrente elétrica em uma direção e bloqueá-la em outra – esse comportamento é chamado de retificação e pode ser utilizado para converter corrente alternada (CA ou AC – *alternating current*, do Inglês –, a energia que temos em nossas tomadas) em corrente contínua (CC ou DC – *direct current*, do Inglês –, a forma como as baterias nos fornecem energia). Outros usos de diodo são proteção de circuitos (contra corrente reversa) e extração de modulação de sinais (por exemplo, para uso em circuitos de comunicação sem fio).

Figura 2.9: Foto de um diodo comum no mercado – a faixa menor (cinza, para esse diodo) corresponde ao terminal negativo

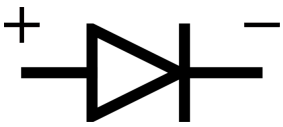


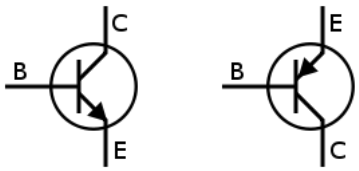

Figura 2.10: Representação de um diodo em circuito elétrico

O diodo conduzirá corrente elétrica caso a tensão em seu terminal positivo (+) seja maior que a tensão em seu terminal negativo (−), ou seja, funcionará como um curto-circuito. Caso contrário, ele funcionará como circuito aberto (não conduzirá). É importante notar que para diodos reais existe uma queda de tensão de 0,7V em sua junção P-N e, com isso, a tensão do lado positivo precisa ser maior que a tensão negativa + 0,7V para que ele conduza.

## 2.4 Transistores

Transistores são dispositivos semicondutores usados como amplificadores ou chaveadores. Sua entrada é uma corrente/tensão que altera a corrente/tensão de saída. Os transistores são a base de todos os circuitos integrados e placas modernos – alguns, como os microprocessadores, possuem milhões deles.

Para nossos estudos, iremos focar nos transistores de estrutura bipolar de junção (ou BJT, *Bipolar Junction Transistor*, do Inglês) com polaridade NPN e PNP, porém existem outros tipos, como os FETs.




(a) NPN (b) PNP

Figura 2.11: Símbolos de transistores BJT em circuito elétrico

Os transistores possuem três terminais: base, coletor e emissor. E suas principais equações características são:

$$I_C + I_B = I_E$$

$$I_C = \beta I_B,$$

onde $\beta$ é uma constante (também referida como $h_{FE}$) característica do transistor (fator de amplificação).

A segunda equação nos evidencia o poder de amplificação de um transistor: dependendo da corrente que colocarmos na base (corrente $I_B$), ele permitirá maior ou menor corrente no coletor (corrente $I_C$).

No mercado encontramos transistores NPN e PNP com vários encapsulamentos diferentes. Alguns comuns são o 2N2904 (NPN) e 2N3906 (PNP).

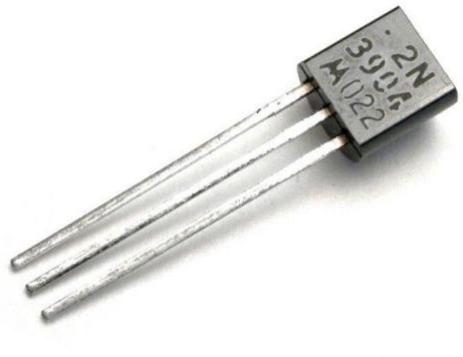



Figura 2.12: Componente 2N3904: transistor NPN com encapsulamento TO-92

### 2.4.1 Utilização de transistores com relés

Relés são componentes úteis quando precisamos isolar eletronicamente um circuito de controle de um circuito de potência. Se quisermos, por exemplo, acender uma lâmpada incandescente (que utiliza corrente alternada) através do Arduino, podemos utilizar um relé: o Arduino controlará o relé, que então fará a conexão (ou não) da lâmpada com a tomada. Existem relés mecânicos e de estado sólido (SSD ou *Solid-state relay*, do Inglês), porém iremos utilizar um relé mecânico em nosso exemplo, por ser mais barato e fácil de se encontrar no mercado.

Como relés possuem correntes de ativação maiores que as portas digitais do Arduino conseguem suprir, precisamos amplificar a corrente que sai das portas digitais do Arduino para que ela seja capaz de acionar o relé – e isso faremos utilizando um transistor NPN.

Utilizando um relé acionado por 5V, um transistor NPN (2N3904) e dois resistores (10Ω para o relé e 470Ω para a base do transistor) podemos utilizar o circuito a seguir para controlar qualquer carga de corrente alternada – basta ligar a carga às saídas do relé, que não estão indicadas no diagrama:

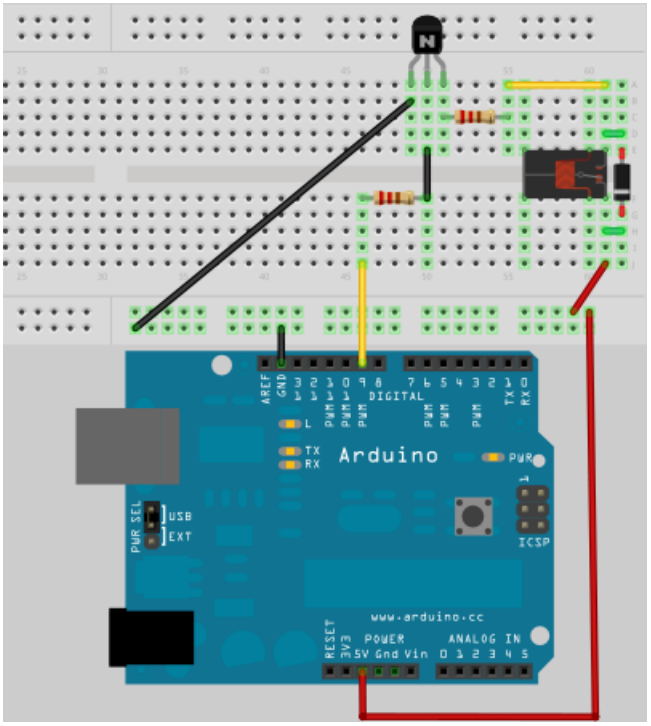



Figura 2.13: Circuito para ligar um relé (controlado por transistor) no Arduino

Fica como exercício para o leitor verificar como seria o uso de um transistor PNP.

### 2.4.2 Ponte-H

Em alguns projetos precisamos inverter a tensão de entrada de determinado circuito. Por exemplo: os motores de corrente contínua giram para um lado caso apliquemos tensão positiva em seu terminal esquerdo e negativa em seu terminal direito. Porém, para fazê-los girar em sentido contrário, precisamos aplicar tensão negativa em seu terminal esquerdo e positiva em seu terminal direito.

Podemos implementar circuitos que fazem essa inversão de tensão a partir de 4 transistores funcionando como chave. Esse tipo de circuito se chama ponte-H por conta da disposição dos transistores com relação ao motor, como pode ser visto no diagrama a seguir:

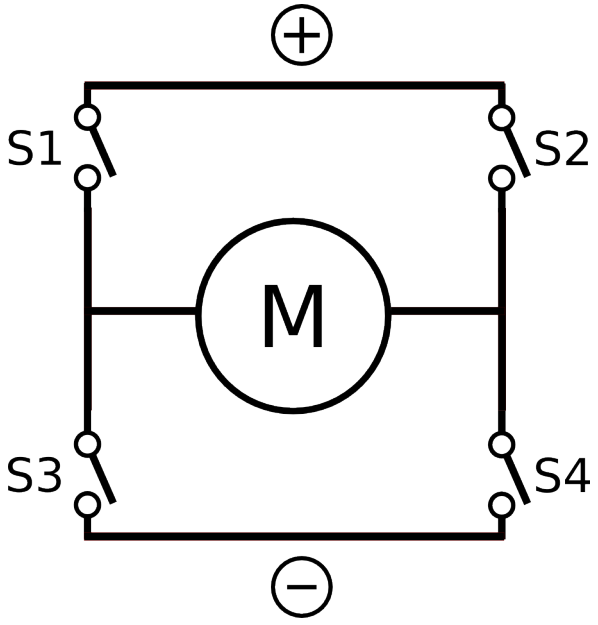



Figura 2.14: Diagrama de uma ponte-H

Quando fechamos as chaves S1 e S4, o terminal esquerdo do motor recebe tensão positiva e o terminal direito recebe tensão negativa. Já quando fechamos as chaves S2 e S3, o terminal esquerdo do motor recebe tensão negativa e o terminal direito recebe tensão positiva. O que precisamos fazer é substituir as chaves por transistores, para então podermos controlar o sentido de rotação do motor através do Arduino.

A seguir temos um circuito bastante completo de ponte-H que utiliza diodos de proteção e acopladores ópticos para dar mais segurança à solução como um todo:

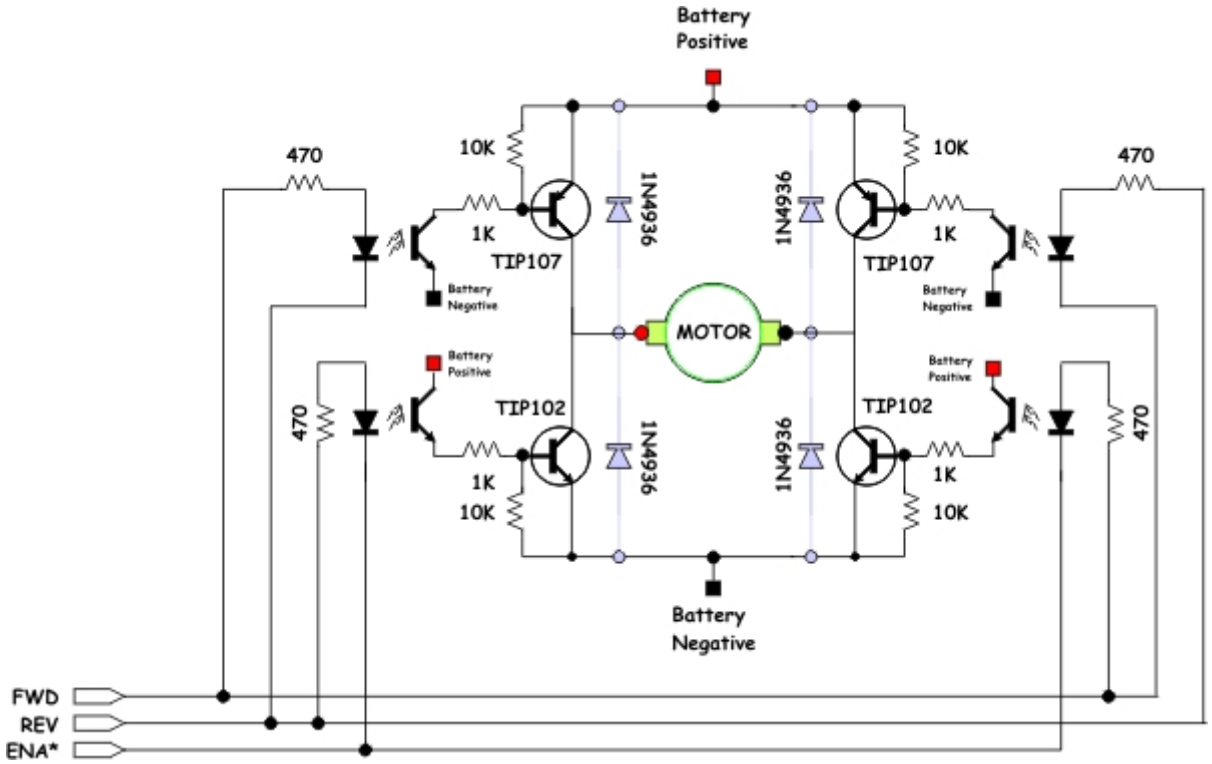



Figura 2.15: Esquema de uma ponte-H com transistores
Retirado de `http://www.mcmanis.com/chuck/robotics/tutorial/h-bridge/`

Para quem não quer ter trabalho montando o circuito, existe a opção de comprar um circuito integrado pronto com a ponte-H – uma das opções é o L293D.

### PWM e ponte-H para controle de velocidade

Como com PWM conseguimos controlar a quantidade de potência que será entregue a um circuito, caso utilizemos PWM como entrada do controle de uma ponte-H, podemos limitar a quantidade de potência entregue ao motor e, com isso, controlar sua velocidade.

# Capítulo 3

# Eletrônica Digital

## 3.1 Introdução

Dizemos que um circuito é digital quando suas entradas e saídas trabalham com sinais digitais, ou seja, sinais com valores bem definidos. Geralmente esses circuitos trabalham apenas com dois valores e, por isso, chamamos esses sistemas de digitais binários.

Quando estamos falando de circuitos digitais, estamos falando de transporte de informação. E como temos somente dois valores possíveis de tensão, teremos toda a informação codificada em binário – chamamos cada informação binária de bit (dígito binário ou ***b**inary dig**it***, do Inglês) e os representamos por 0 e 1.

Dessa forma, se nossos circuitos trabalham com tensões de 0V e 5V, dizemos que 0V equivale ao bit 0 e 5V equivale ao bit 1 – agora passamos a falar de bits (informação) em vez de tensões, ou seja, estamos pensando uma camada acima.

## 3.2 Portas lógicas

Assim como na Matemática possuímos operações básicas como soma, subtração, multiplicação e divisão, na aritmética binária temos operações que podemos fazer com nossos bits. Para simplificar, vamos tratar as operações com uma ou duas entradas, apenas uma saída, tratar o bit 0 como sinônimo de **falso** e o bit 1 como sinônimo de **verdadeiro**. Dessa forma, temos as seguintes operações:

- **AND**: operação que resulta em bit 1 somente quando os dois bits de entrada são 1 (ou seja, só resulta em "verdadeiro" se somente o primeiro **e** o segundo bit forem "verdadeiros".);
- **OR**: operação que resulta em 1 quando pelo menos uma das entradas é 1 (ou seja, resulta em "verdadeiro" quando o primeiro **ou** o segundo bit forem "verdadeiros".);
- **NOT**: operação que resulta na inversão do bit de entrada;
- **XOR**: também chamade de *exclusive or* ("ou exclusivo"), essa operação só resulta em bit 1 quando somente um dos bits de entrada é 1.

O circuito abaixo exemplifica a criação de uma porta do tipo AND:

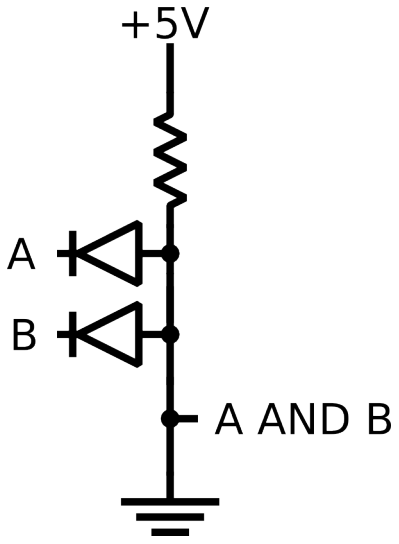



Figura 3.1: Porta lógica AND criada a partir de diodos e resistor

### 3.2.1 Tabela-verdade

Para um número finito de entradas podemos aplicar as operações lógicas em todos os possíveis valores dessa entrada e obter todos os possíveis resultados da operação/função lógica. Chama-se tabela-verdade a tabela que lista todas essas possibilidades.

Para as funções lógicas acima, quando temos duas entradas temos um total de 4 possíveis combinações de entradas diferentes (o número de combinações binárias é sempre $2^{número\ de\ entradas}$) e as seguintes tabelas:

| A | B | A AND B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

(a) Operação AND

| A | B | A OR B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

(b) Operação OR

| A | NOT A |
|---|---|
| 0 | 1 |
| 1 | 0 |

(c) Operação NOT

| A | B | A XOR B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

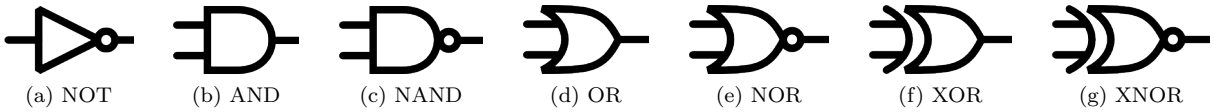
(d) Operação XOR

Figura 3.2: Tabelas-verdade das operações lógicas básicas

Além das operações básicas, temos as negações nas mesmas, como NAND, NOR e XNOR. Para saber a tabela-verdade dessas, basta negar a saída da tabela verdade das outras operações (AND, OR e XOR, respectivamente).

### 3.2.2 Representação das operações

As operações lógicas citadas acima podem ser representadas em circuitos pelos seguintes símbolos:



(a) NOT (b) AND (c) NAND (d) OR (e) NOR (f) XOR (g) XNOR

Figura 3.3: Símbolos das operações lógicas

Além disso, podem ser escritas por extenso:

| Operação | Representação |
|---|---|
| NOT A | $\overline{A}$ |
| A AND B | $A + B$ |
| A NAND B | $\overline{A + B}$ |
| A OR B | $A \cdot B$ |
| A NOR B | $\overline{A \cdot B}$ |
| A XOR B | $A \oplus B$ |
| A XNOR B | $\overline{A \oplus B}$ |

### 3.2.3 Funções lógicas compostas

Assim como na Matemática, podemos fazer composições das funções lógicas básicas para obter novas funções (compostas). A função XOR, por exemplo, pode ser obtida através da composição das funções NOT, AND e OR:

$$A \oplus B = \overline{A} \cdot B + A \cdot \overline{B}$$

ou

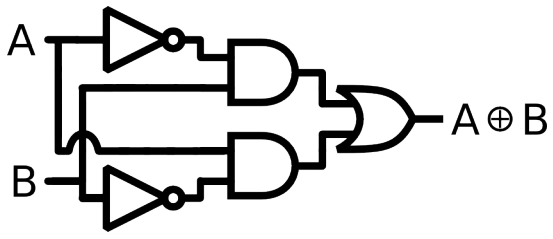



Figura 3.4: Diagrama da função composta XOR

A partir de funções lógicas compostas e técnicas como realimentação conseguimos criar dispositivos mais complexos como *latches*, *flip-flops*, *coders/decoders*, *mux/demux*, dentre outros. Esses dispositivos são a base para criar circuitos lógicos de alto nível, por exemplo: com alguns *flip-flops* conseguimos criar registradores, que podem evoluir para memórias e fazer parte do circuito de um microprocessador.

## Capítulo 4

# Fazendo barulho com o Arduino

Para quem gosta de fazer música, o Arduino possui uma função pronta para criar uma onda quadrada na frequência e no tempo desejados. Apesar de serem notas simples e a onda ser quadrada, adicionando circuitos extras (para filtros e distorções) e um pouco de criatividade, conseguimos criar sons legais para nossos projetos.

A função que faz esse trabalho é chamada `tone`. Vamos criar um projeto-exemplo ligando um *buzzer* – componente que reproduz sons de acordo com as variações de tensão em seus terminais – para tocar nosso som da seguinte forma:

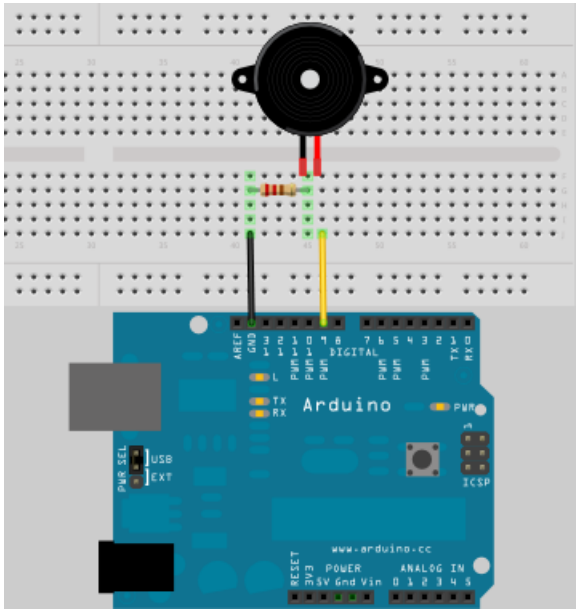



Figura 4.1: Circuito com *buzzer*

Utilizaremos o seguinte código:

```
#define BUZZER 9

int notas[] = { 524, 588, 660, 699, 785, 881, 989 };

void setup() {
  pinMode(BUZZER, OUTPUT);
}

void loop() {
  for (int i = 0; i < 7; i++) {
    tone(BUZZER, notas[i], 1000);
    delay(1000);
  }
  delay(1000);
}
```

Além do laço `for`, utilizamos também um vetor de inteiros chamado `notas`. Um vetor nada mais é que um local onde armazenamos várias variáveis de mesmo tipo. Os vetores são indexados e para acessar cada item guardado neles utilizamos índices que variam de 0 a $n-1$, onde $n$ é o número total de elementos. No exemplo acima, utilizamos a variável `i` para percorrer o vetor e, por isso, para acessar os elementos utilizamos `i`.

Os segredos do código acima são:

- Saber a frequência das notas[1] e
- Saber utilizar a função `tone`. A função `tone` recebe três parâmetros, respectivamente: pino (precisa ser um pino que tenha suporte a PWM), frequência da nota e duração do som em milissegundos.

[1] Saiba mais em http://pt.wikipedia.org/wiki/Série_harmônica_(música)

## Capítulo 5

# Armazenando na EEPROM

EEPROM (*Electrically Erasable Programmable Read-Only Memory*, do Inglês) é um tipo de memória não volátil, ou seja, que não se apaga ao retirarmos energia de seu circuito (o que não é verdade para as memórias do tipo RAM, por exemplo). O ATMega328, microcontrolador presente no Arduino Duemilanove, possui um circuito EEPROM integrado de 1024 *bytes* (ou $1KiB$) – em outras versões do Arduino, como o Mega (que usa ATMega1280 ou ATMega2560, dependendo do modelo), a EEPROM pode chegar até $4KiB$.

Ter o circuito EEPROM integrado significa que nosso *software* pode armazenar $1KiB$ de dados que não serão perdidos (mesmo que desliguemos o Arduino da fonte de alimentação) – funciona como se fosse um *micro-pendrive*. Para utilizar a EEPROM não precisamos de circuitos adicionais: precisamos apenas utilizar a biblioteca `EEPROM.h`. Porém, para nosso exemplo, vamos ligar um LED na porta 11 e utilizar o seguinte código:

```
#include <EEPROM.h>
#define LED 11

void setup() {

  for (int i = 0; i < 16; i++) {
    EEPROM.write(i, i * i);
  }
}

void loop() {
  for (int i = 0; i < 16; i++) {
    byte leitura = EEPROM.read(i);
    analogWrite(LED, leitura);
    delay(50);
  }
  delay(1000);
}
```

A diretiva `#include` diz ao compilador que queremos incluir a biblioteca `EEPROM.h` – isso quer dizer que, além do código que digitamos, utilizaremos um código já criado pela equipe do Arduino para facilitar a utilização da EEPROM. O exemplo nada mais faz que escrever 16 *bytes* na EEPROM (na função `setup`), ler um por um e configurar o valor lido como saída PWM da porta 11 (na função `loop`).

A função de escrita, `EEPROM.write`, recebe dois parâmetros: o endereço onde ela vai escrever (um valor entre 0 e 1023, no caso do Arduino Duemilanove com ATMega328) e o *byte* que escreveremos nesse local, já a função `EEPROM.read` recebe apenas um parâmetro (o endereço de onde ela fará a leitura) e nos retorna o valor lido na memória (um *byte*).