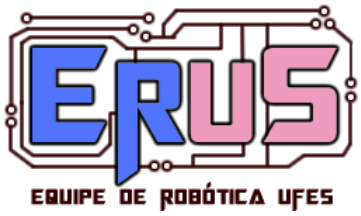



# Minicurso Arduino

**JACEE 2012**

# 1. Introdução

### O que é o Arduino?

Arduino é uma plataforma de prototipagem eletrônica criado com o objetivo de permitir o desenvolvimento de controle de sistemas interativos, de baixo custo e acessível a todos. Além disso, todo material (software, bibliotecas, hardware) é open-source, ou seja, pode ser reproduzido e usado por todos sem a necessidade de pagamento de direitos autorais. Sua plataforma é composta essencialmente de duas partes: O Hardware e o Software.

Neste curso, será utilizado o Arduino Uno.

# 2. O Hardware

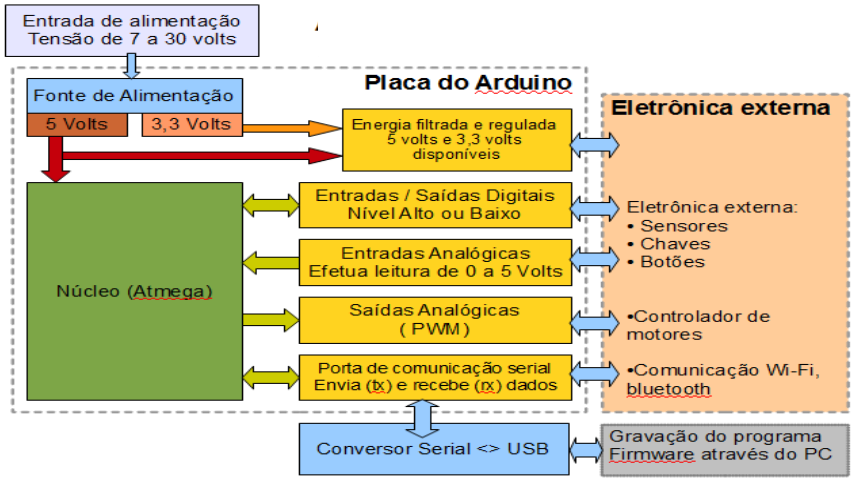


*Figura 1: Arquitetura do Arduino*

O hardware do Arduino é muito simples, porém muito eficiente. Vamos analisar a partir deste momento, o hardware do Arduino UNO. Esse hardware é composto dos seguintes blocos:

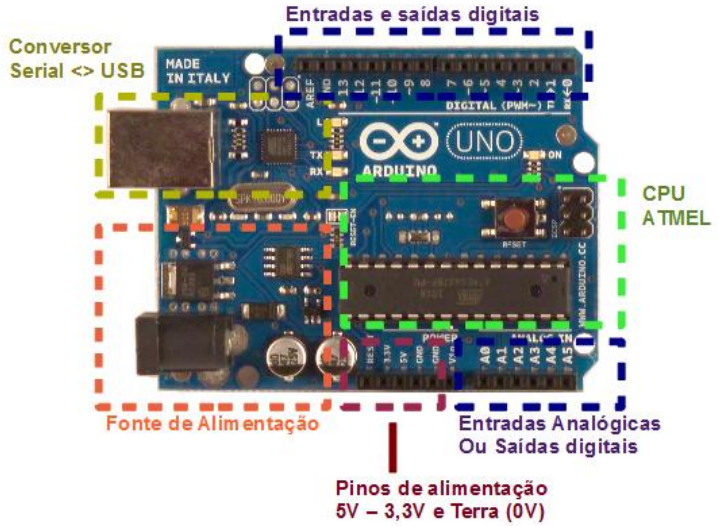


*Figura 2: Blocos Arduino*

## 2.1 Fonte de Alimentação

Responsável por receber a energia de alimentação externa, que pode ter uma tensão de no mínimo 7 Volts e no máximo 35 Volts e uma corrente mínima de 300mA. A fonte filtra e depois regula a tensão de entrada para duas saídas: 5 Volts e 3,3 Volts. O requisito deste bloco é entregar as tensões de 5 e 3,3 Volts para que a CPU e os demais circuitos funcionem.

## 2.2 Núcleo CPU:

O núcleo de processamento de uma placa Arduino é um micro controlador, uma CPU, um computador completo, com memória RAM, memória de programa (ROM), uma unidade de processamento de aritmética e os dispositivos de entrada e saída. Tudo em um chip só. E é esse chip que possui todo hardware para obter dados externos, processar esses dados e devolver para o mundo externo.

Os desenvolvedores do Arduino optaram em usar a linha de micro controladores da empresa ATMEL. A linha utilizada é a ATMega. Existem placas Arduino oficiais com diversos modelos desta linha, mas os mais comuns são as placas com os chips ATMega8, ATMega162 e ATMega328p. Esses modelos diferem na quantidade de memória de programa (ROM) e na configuração dos módulos de entrada e saída disponíveis.

## 2.3 Entradas e Saídas

Comentamos acima que basicamente toda eletrônica ativa está dentro do chip micro controlador. Para entender o que seria essa eletrônica, vamos considerar o chip mais simples usado no Arduino: o ATMega8.


*Figura 3: Micro controlador ATMega8*

O chip acima possui 28 pinos de conexões elétricas, 14 de cada lado. É através desses pinos que podemos acessar as funções do micro controlador, enviar dados para dentro de sua memória e acionar dispositivos externos.

No Arduino, os 28 pinos deste micro controlador são divididos da seguinte maneira:

- 14 pinos digitais de entrada ou saída (programáveis)
- 6 pinos de entrada analógica ou entrada/saída digital (programáveis)
- 5 pinos de alimentação (gnd, 5V, ref analógica)
- 1 pino de reset
- 2 pinos para conectar o cristal oscilador

Os dois primeiros itens da lista são os pinos úteis, disponíveis para o usuário utilizar. Através destes pinos que o Arduino é acoplado à eletrônica externa. Entre os 14 pinos de entrada/saída digitais temos 2 pinos que correspondem ao módulo de comunicação serial USART. Esse módulo permite comunicação entre um computador (por exemplo) e o chip.

Todos os pinos digitais e os analógicos possuem mais de uma função. Os pinos podem ser de entrada ou de saída, alguns podem servir para leituras analógicas e também como entrada digital. As funções são escolhidas pelo programador, quando escreve um programa para a sua placa.

Na placa do Arduino, os pinos úteis do micro controlador são expostos ao usuário através de conectores fêmea (com furinhos) onde podem ser encaixados conectores para construir o circuito externo à placa do Arduino.

### 2.3.1 Entradas Digitais

No total temos disponíveis 20 pinos que podem ser utilizados como entradas digitais. Os 14 pinos digitais mais os 6 pinos analógicos, podem ser programados para serem entradas digitais. Quando um pino é programado para funcionar como entrada digital, através do programa que escrevemos colocamos um comando que ao ser executado efetua a "leitura" da tensão aplicada ao pino que está sendo lido. Então, após a execução deste comando, sabemos se o pino encontra-se em um estado "alto" ou "baixo".

Na prática, o programa pode saber se um pino está alimentado com 0 (zero) ou 5 Volts. Essa função é utilizada geralmente para identificar se um botão está pressionado, ou um sensor está "sentindo" alguma coisa no mundo externo.

Note que a função de entrada digital apenas entrega 0 ou 1, sem tensão ou com tensão. Não é possível saber quanta tensão está sendo aplicada no pino. Para isso usamos as entradas analógicas.

### 2.3.2 Entradas Analógicas

Temos disponíveis no Arduino Uno 6 entradas analógicas. Ao contrário de uma entrada digital, que nos informa apenas se existe ou não uma tensão aplicada em seu pino, a entrada analógica é capaz de medir a tensão aplicada. Através da entrada analógica, conseguimos utilizar sensores que convertem alguma grandeza física em um valor de tensão que depois é lido pela entrada analógica.

### 2.3.3 Saídas Digitais

Com uma saída digital podemos fazer com que um pino libere 0 volts ou 5 volts. Com um pino programado como saída digital, podemos acender um led, ligar um relé, acionar um motor, dentre diversas outras coisas. Podemos programar o Arduino para no máximo 20 saídas digitais, porém podemos utilizar um ou mais pinos para controlar um bloco de pinos.

## 2.4 Pinos com funções especiais

Existem pinos do Arduino que possuem características especiais, que podem ser usadas efetuando as configurações adequadas através da programação. São eles:

PWM: Tratado como saída analógica, na verdade é uma saída digital que gera um sinal alternado (0 e 1) onde o tempo que o pino fica em nível 1 (ligado) é controlado. É usado para controlar velocidade de motores, ou gerar tensões com valores controlados pelo programa. **Pinos 3, 5, 6, 9, 10 e 11.**

Porta Serial USART: Podemos usar um pino para transmitir e um pino para receber dados no formato serial assíncrono (USART). Podemos conectar um módulo de transmissão de dados via bluetooth por exemplo e nos comunicarmos com o Arduino remotamente. **Pinos 0 (rx recebe dados) e pino 1 (tx envia dados).**

Comparador analógico: Podemos usar dois pinos para comparar duas tensões externas, sem precisar fazer um programa que leia essas tensões e as compare. Essa é uma forma muito rápida de comparar tensões e é feita pelo hardware sem envolver programação. **Pinos 6 e 7.**

Interrupção Externa: Podemos programar um pino para avisar o software sobre alguma mudança em seu estado. Podemos ligar um botão a esse pino, por exemplo, e cada vez que alguém pressiona esse botão o programa rodando dentro da placa é desviado para um bloco que você escolheu. Usado para detectar eventos externos à placa. **Pinos 2 e 3.**

Porta SPI: É um padrão de comunicação serial Síncrono, bem mais rápido que a USART. É nessa porta que conectamos cartões de memória (SD) e muitas outras coisas. **Pinos 10 (SS), 11 (MOSI), 12 (MISO) e 13 (SCK).**

## 2.5 Firmware

É simplesmente um software que é carregado dentro da memória do micro controlador. Tecnicamente o firmware é a combinação de uma memória ROM, somente para leitura, e um programa que fica gravado neste tipo de memória. E esse é o caso do micro controlador que a placa Arduino usa.

# 3. O Software

Quando tratamos de software na plataforma do Arduino, podemos referir-nos: ao ambiente de desenvolvimento integrado do Arduino e ao software desenvolvido por nós para enviar para a nossa placa. O ambiente de desenvolvimento do Arduino é um compilador gcc (C e C++) que usa uma interface gráfica construída em Java. Basicamente se resume a um programa IDE muito simples de se utilizar e de estender com bibliotecas que podem ser facilmente encontradas. As funções da IDE do Arduino são basicamente duas: Permitir o desenvolvimento de um software e enviá-lo à placa para que possa ser executado.

Para realizar o download do software basta ir até a página oficial do Arduino (http://www.arduino.cc/), escolher o seu SO (existe pra Linux, Mac e Windows) e baixa-lo. Obviamente, por ser open source, é gratuito. Depois de baixado não necessita de nenhuma instalação, é só abrir o IDE e começar a utilizar.

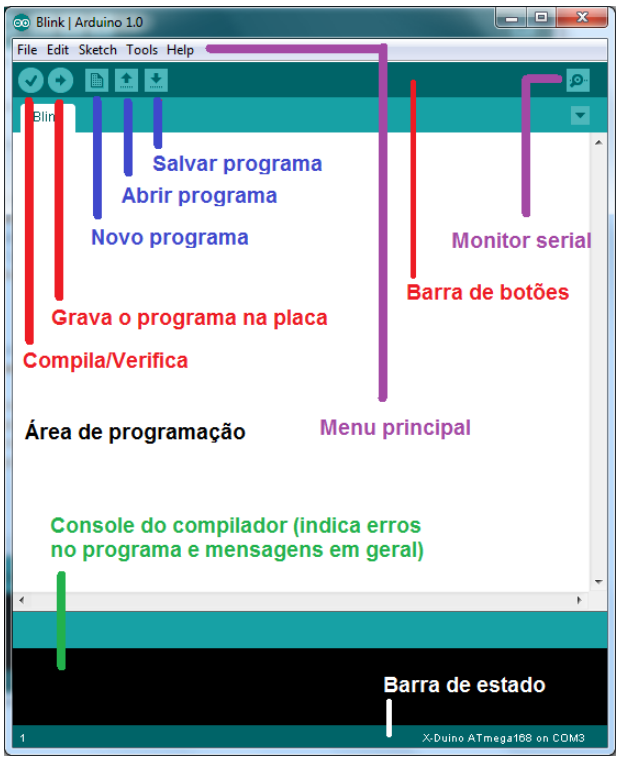


*Figura 4: Ambiente de programação*

Para começar a utilizar, devemos escolher qual placa estamos utilizando, assim vamos em Tools > Board e à escolhemos. O IDE possui também uma quantidade imensa de exemplos. Para utilizá-los basta ir a File > Examples e escolher qual deles você quer testar, de acordo com sua necessidade.

## 3.1 Estrutura do programa

Aproveitando os exemplos que o IDE fornece, nosso primeiro programa vai ser acender um led.

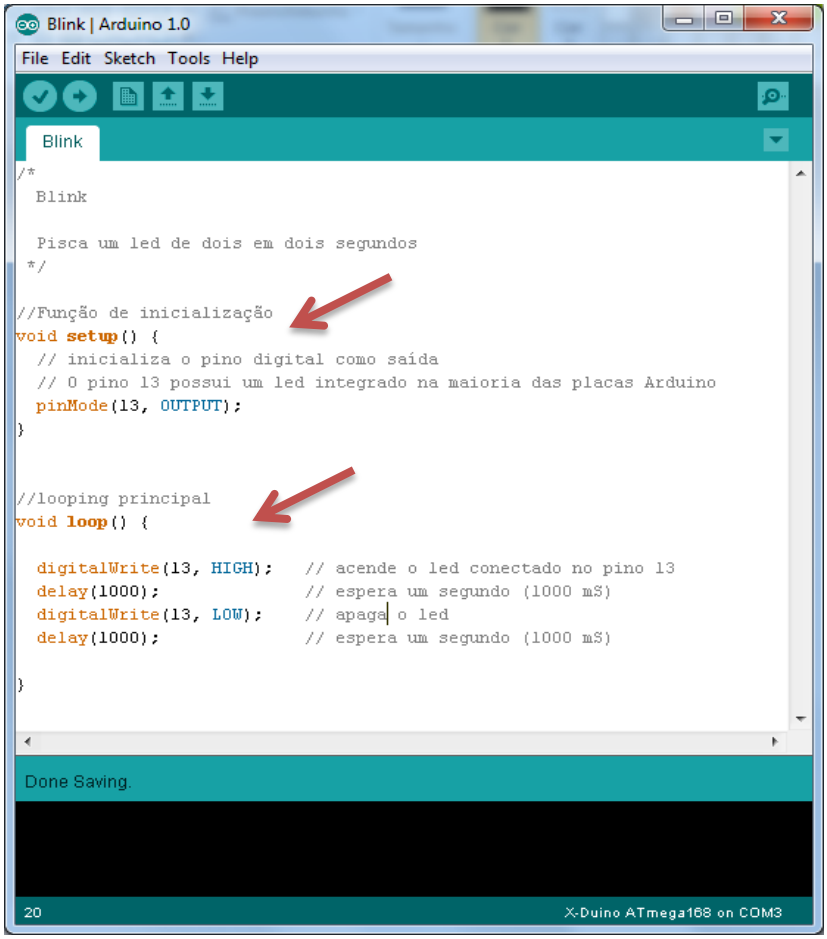


*Figura 5: Entendendo o programa*

O programa para o Arduino é dividido em duas partes principais: **Setup** e **Loop**, como indicam as setas na imagem.

A função **setup** serve para inicialização da placa e do programa. Esta sessão é executada uma vez quando a placa é ligada ou resetada através do botão. Aqui, informamos para o hardware da placa o que vamos utilizar dele. No exemplo, vamos informar para a placa que o pino 13 será uma saída digital onde está conectado um LED (no Arduino UNO o pino 13 possui um led integrado).

A função **loop** é como se fosse a main () da placa. O programa escrito dentro da função **loop** é executado indefinidamente, ou seja, ao terminar a execução da última linha desta função, o programa inicia novamente a partir da primeira linha da função **loop** e continua a executar até que a placa seja desligada ou o botão de reset seja pressionado.

Analisando o resto do programa, o comando **digitalWrite** escreve na saída do pino 13 o nível de tensão **HIGH** (5v), acendendo o Led. O comando **delay** é apenas para o programa aguardar 1000 milésimos. Em seguida, o nível de tensão é alterado para **LOW** (0v) e o Led apaga. E assim é repetido infinitamente, até ser desligado.

Com o programa feito, compilamos o mesmo para verificarmos se não existe nem um erro. Caso não contenha erro, agora temos que enviá-lo para placa através do botão de upload (os botões estão especificados na figura 4). Após o envio os Led's RX e TX deverão piscar, informando que o código está sendo carregado. Logo após o arduíno começará a executar o programa que lhe foi enviado.

## 3.2 Serial Monitor

Esse monitor é usado para que possamos comunicar nossa placa com o computador, mas também é muito útil para a depuração do programa. Basicamente conectamos a placa no computador e através desta tela podemos ver as informações enviadas pela placa.

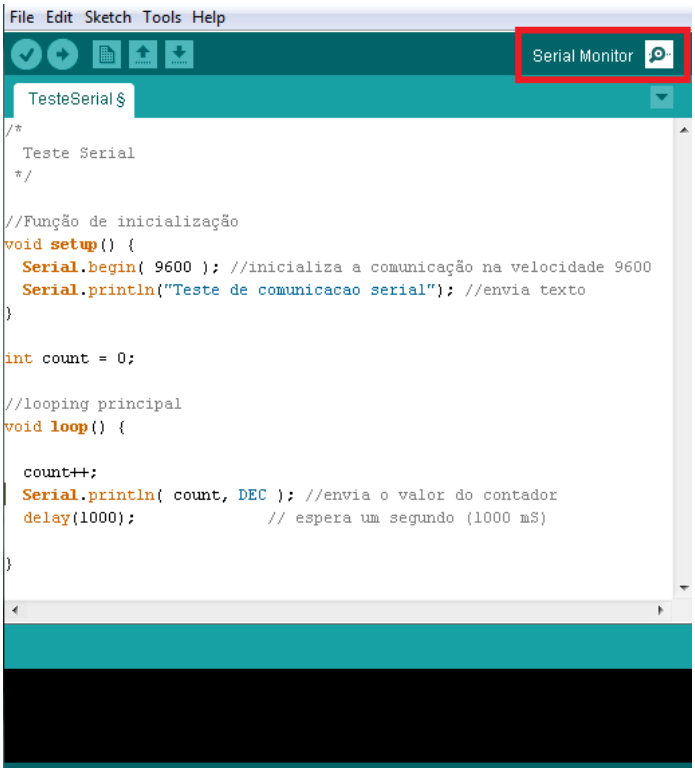


*Figura 6: Exemplo serial monitor*

No exemplo, o comando **Serial.begin(9600)** inicializa a comunicação com uma taxa de 9600 baunds (taxa de bits). O comando **Serial.println ('argumento')** envia a mensagem para o computador.

Após compilar e enviar o programa para placa, abrimos o serial monitor. As informações enviadas pela nossa placa Arduino aparecem no console.

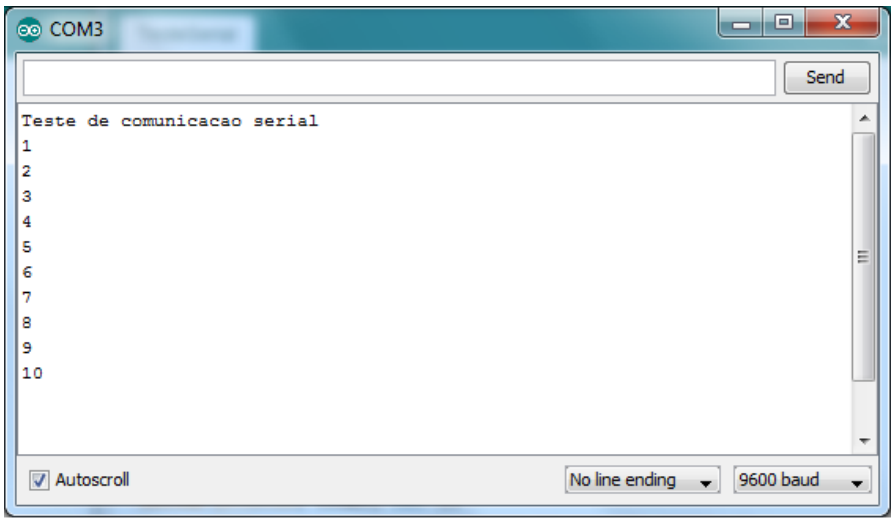


*Figura 7: Saída no serial monitor*

## 3.3 Outros comandos

Como já foi informado, e vocês já devem ter observado, a linguagem base para a programar um Arduino é C. Logo, suas estruturas de controle (if, else, while, for...), seus elementos de sintaxe ( #define, #include, {}...), operadores aritméticos (+, -, *, ^ ...), operadores de comparação (==, !=, <, > ...), enfim, todos são utilizados aqui no IDE. Portanto, saber C é primordial para programar o Arduino em alto nível.

Abaixo segue as principais funções para controlar o arduíno (algumas já foram especificados acima):

**pinMode (pin, mode)**: Configura o pino especificado para que se comporte como entrada ou saída, sendo Pin = número do pino e mode = INPUT ou OUTPUT

**digitalWrite (pin,value)**: escreve um valor HIGH ou LOW em um pino digital. Se o pino foi configurado como saída sua voltagem será determinada ao valor correspondente: 5V para HIGH e 0V para LOW. Se o pino estiver configurado como entrada escrever um HIGH levantará o resistor interno de 20kΩ. Escrever um LOW rebaixará o resistor. Obviamente pin = numero do pino e valor = HIGH ou LOW.

int **digitalRead (pin):** Lê o valor de um pino digital especificado, HIGH ou LOW. Pin = numero do pino. Retorna HIGH ou LOW.

Int **analogRead (pin):** Lê o valor de um pino analógico especificado. Pode mapear voltagens entre 0 a 5v, sendo 4,9mV por unidade.

**analogWrite (pin, value):** Escreve um valor analógico (onda PWM, explicaremos mais abaixo). Pode ser utilizada para acender um LED variando o brilho ou girar um motor a velocidade variável. Após realizar essa função o pino vai gerar uma onda quadrada estável com ciclo de rendimento especificado até que o próximo analogWrite() seja realizado (ou que seja realizado um digitalRead() ou digitalWrite() no mesmo pino).

Para mais comandos, consultar o guia de referências em http://arduino.cc/en/Reference/HomePage

# 4. Utilizando um protoboard

Até aqui já acendemos um Led interno do arduíno, utilizando o Led já incluso do pino 13. Agora vamos acender outro Led, agora utilizando o protoboard.

Primeiro devemos saber qual a tensão necessária para acender o Led. Para isso utilizamos a tabela abaixo, sendo que cada cor tem a sua tensão específica

| Vermelho | Laranja | Amarelo | Verde | Azul | Branco |
|---|---|---|---|---|---|
| 2v | 2v | 2.1v | 2.2v | 3.3v | 3.3v |

Sabemos que a porta digital escreve HIGH (5v) ou LOW (0v). Logo, se colocarmos 5v o Led irá queimar. Portanto devemos colocar um resistor em série com o Led para limitarmos a corrente que passa por ele. Essa corrente deve ser de algo em torno de 20mA. Assim, utilizando um pouquinho de teoria de circuitos temos:

$$R = \frac{V_{fonte} - V_{Led}}{I_{Led}}$$

Assim teremos a tensão do Led mais a tensão do resistor igual a 5v (tensão da fonte), e nosso Led estará seguro contra sobre corrente. Nem sempre vamos conseguir encontrar um resistor igual ao valor calculado, pois ele pode ser um valor não comercial. Quando isso ocorrer, basta associar resistores ou utilizar algum outro um pouco acima do valor encontrado. Outro ponto importante é a polaridade do Led (ou de qualquer outro componente). Observe que este possui uma perna maior que a outra. A maior é positivo e a menor negativo. Portanto na hora de ligar, não podemos confundir caso contrário isso também queimará o Led. Aqui o exemplo foi um Led, mas poderia ser um sensor, um motor ou algum outro componente. Portanto é preciso **atenção** para não queimarmos nossos equipamentos.

Bom, calculado o valor do resistor, devemos escolher um pino (aqui vamos manter o pino 13) para que seja ligado nosso circuito. Com auxílio de jumpers ligue semelhante a figura abaixo:

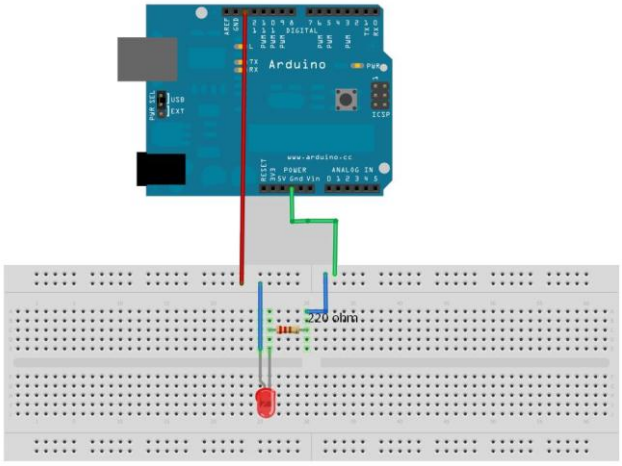


*Figura 8: Exemplo de ligação no protoboard*

Ligado o circuito conforme a figura 8, podemos passar para o Arduino o código da figura 5 (o exemplo de como acender Led) que o nosso Led externo vai acender assim como o interno.

Este exemplo de Led externo não tem a intenção de aprimorar a programação, e sim chamar atenção para alguns detalhes citados acima que pode danificar seus componentes. Perder um Led não é um alarde, pois este é bem barato, porém, se não tomados os devidos cuidados, você pode perder motores, sensores, dentre outros componentes bem mais caros.

# 5. Utilizando o PWM

PWM significa modulação por largura de pulso (Pulse Width modulation) e é basicamente uma técnica para obtermos resultados analógicos em com meios digitais. O controle digital é usado para criar uma onda quadrada, um sinal alternado entre ligado e desligado. Assim a técnica consiste em manipularmos a razão cíclica de um sinal, o **duty cycle** afim de transportar informação ou controlar a potência de algum outro circuito**.** Com isso, teremos um sinal digital que oscila entre 0v e 5v com determinada frequência (o Arduino trabalha com um padrão próximo a 500Hz). O duty cycle é a razão do tempo em que o sinal permanece em 5v sobre o tempo total de oscilação, como está ilustrado na figura abaixo:

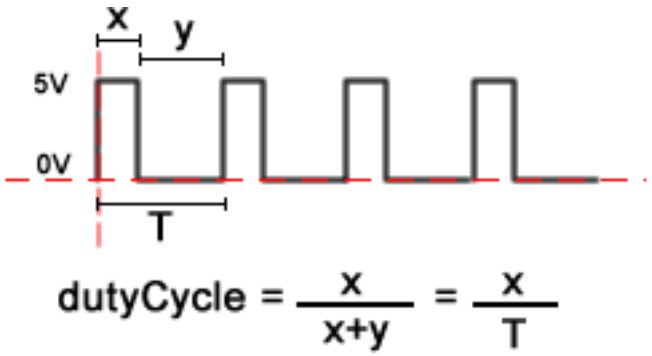


*Figura 9: Sinal PWM*

Portanto, o que controlamos através do software é justamente o **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, como por exemplo, controlar um servo motor.

Para exemplificar, vamos controlar a luminosidade de um Led. Mas antes, relembrando o que já foi dito acima, **os pinos que possuem PWM são os pinos 2, 5, 6, 9, 10 e 11** (no Arduino UNO). Faça as ligações necessárias (semelhante como já foi explicando anteriormente) e execute o exemplo abaixo.

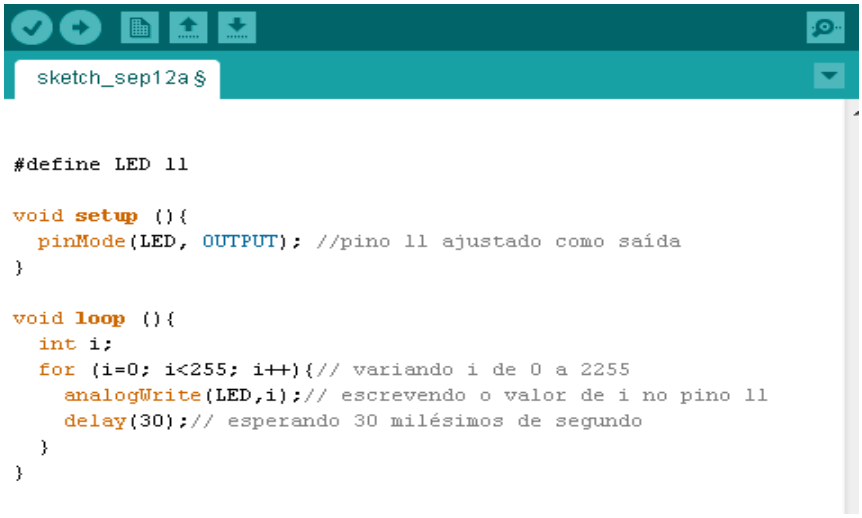


*Figura 10: Exemplo de utilização do PWM*

A função analogWrite (), apesar de estarmos utilizando uma porta digital, é a responsável pelo PWM e recebe como parâmetro o pino e um valor entre 0 – 255, **em que o 0 corresponde a 0% e 255 corresponde a 100% do duty cycle.** Quando rodarmos o código, podemos observar que o LED acenderá de maneira mais suave. Cada etapa de luminosidade diferente corresponde a uma iteração do for. Agora, tente modificar o código para que o LED também apague suavemente.

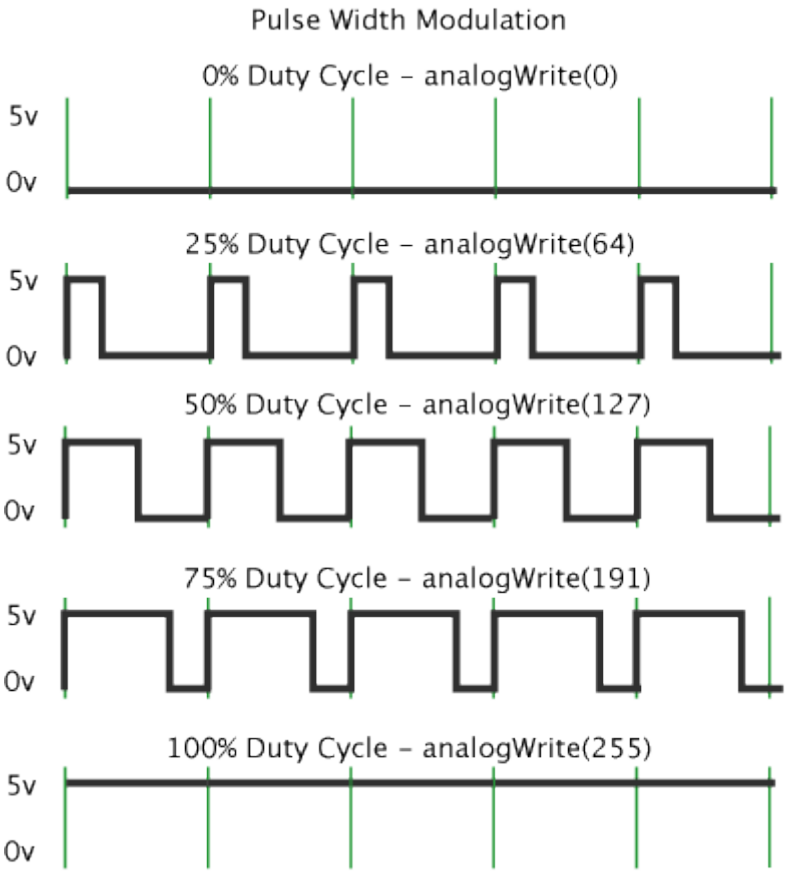


*Figura 11: Gráfico da variação do duty cycle*

# 6. Shields

O Arduino possui o que chamamos de Shields, que são nada mais, nada menos do que outras plaquinhas que se acoplam à placa original, agregando diversas outras funcionalidades.

Existe diversos Shields, com diversas funções. Alguns servem como entrada, como saída e outros como entradas e saída. Com eles conseguimos, por exemplo, fazer com que o Arduino se comunique numa rede Ethernet ou via USB com um celular Android. Alguns Shields possuem circuitos integrados prontos para controlarmos motores sem que precisemos nos preocupar com complicações eletrônicas envolvidas, como é o caso do Motor Shield (figura 12), que possui uma ponte H que nos auxilia no controle de motores.


*Figura 12: Motor Shield*

# 7. Controlando um Ultrassom

O Arduino possui muitos sensores. Aqui vamos aprender a controlar o Ultrassom. Primeiramente vamos entender como o ultrassom funciona. Este sensor emite um sinal na faixa de frequência do ultrassom (por volta de 30kHz). Este sinal se propaga pelo ar até encontrar um obstáculo. Ao colidir com o obstáculo uma parte do sinal é refletida e captada pelo sensor. Portanto, um único sensor de ultrassom, possui um receptor e um emissor.


*Figura 13: Sensor Ultrassom*

Como temos um receptor e um emissor, precisaremos de dois pinos um para cada (existe ultrassons que possuem um pino para os dois, porém os mais comuns utilizam dois). Portanto declaramos dois pinos, um como saída (que emite o sinal) e outro como entrada (que recebe o sinal). O pino que envia o pulso é chamado de trigger e o que recebe echo. Observe o código abaixo (bastante comentado):


*Figura 14: Programando o ultrassom*

Observe que começamos com o trigger desligado, e logo após 2 microssegundos o trigger emite o sinal. Após 10 microssegundos o trigger volta a ser desligado. Na variável duration fica armazenando o tempo entre a emissão do sinal até o echo entrar em HIGH. Assim, a distância é calculada utilizando a formula de velocidade V = S/t. Como o tempo é de ida e volta, devemos dividir o mesmo por 2. A velocidade do som no ar é de aproximadamente 340 m/s. Após algumas conversões de unidades temos que para que o resultado saia em centímetros, temos que dividir o tempo (duration) por 29. Por fim o resultado é impresso no serial monitor.

Para ligarmos o sensor precisaremos de auxílio de um protoboard. Observe que no sensor vem especificado, trigger, echo, vcc e gnd. Agora é só ligar cada pino em seu correspondente e testar o seu programa.

# 8. Utilizando a interrupção

Uma interrupção de entrada e saída (IO) é uma função que é executada quando existe uma mudança estado no pino correspondente, independente do ponto em que se

encontra a execução do programa. O Arduino UNO possui dois pinos que podem ser ligados como interrupção, **os pinos 2 e 3** (no Arduino mega temos 6 interrupções).

A função **attachInterrupt (interrupcao,funcao,modo)** permite configurar uma função para ser executada caso haja uma mudança no pino de IO correspondente. Os parâmetros da função são utilizados da seguinte forma:

1 – interrupção: um inteiro que informa o número da interrupção, sabendo que a interrupção 0 está associada ao pino 2 e a interrupção 1 está associada ao pino 3.

2 – função: É a função a ser chamada caso ocorra a interrupção. Essa função não deve ter parâmetros e nem retornar nada. É conhecida como rotina de serviço de interrupção.

3 – modo: Define quando a interrupção deve ser acionada. Quatro constantes são pré-definidas como valores válidos: **LOW** para acionar a interrupção sempre que o pino for baixo, **CHANGE** para chamar uma interrupção sempre que o pino sofrer alguma alteração de valor, **RISING** para acionar quando o pino vai LOW para HIGH e por fim **FALLING** para acionar quando o pino vai de HIGH para LOW.

Segue o exemplo abaixo em que um Led é acendido ou apagado quando ocorre alguma alteração no pino 13:

```
int pino = 13;
volatile int estado = LOW; // declarando o estado

void setup (){
  pinMode (pino, OUTPUT);
  attachInterrupt (0, blink,CHANGE); // chamando a função com a interrupção 0, logo associada
                                     // ao pino 2, a função blink é a função de interrupção
                                     // setada com o modo CHANGE
}

void loop (){
  digitalWrite (pin, estado);
}

blink void (){
  estado = estado!;
}
```

*Figura 15: Exemplo de interrupção*

Como pratica, tente adicionar uma interrupção com um sensor ultrassom.

# 9. Adicionando uma biblioteca

Possivelmente algum dia você irá precisar adicionar uma biblioteca para trabalhar com seu com algum sensor ou outro componente no seu Arduino. Existem diversas bibliotecas disponíveis na internet, que você pode baixar e utilizá-las. Entretanto você tem que adicioná-las ao seu IDE para que o mesmo reconheça os comandos dela que você está utilizando. Para

mostrar como proceder, vamos adicionar a biblioteca MsTimer2.h como exemplo. Primeiro vamos baixa-la na página do Arduino (http://arduino.cc/playground/Main/MsTimer2). Feito isso, descompacte o arquivo.zip que foi baixado. Agora vá até a pasta onde você "instalou" o seu IDE para o Arduino e procure pela pasta libraries. Dentro deste diretório copie a pasta que foi extraída anteriormente. Por fim, vamos verificar se a biblioteca foi mesmo detectada pelo IDE. Vá em Files > Examples e verifique se a biblioteca que acabamos de adicionar está ali. Se sim, a instalação ocorreu tudo bem e você já pode começar a utilizar a sua nova biblioteca. Agora é só "chama-la" no seu código, que neste caso ficaria: #include <MsTimer2.h>. Vale a pena destacar que na própria página onde você baixou a biblioteca, possui as instruções de como utilizar a mesma.

# 10. Explorando um servomotor

O servomotor é um dispositivo eletromecânico que, a partir de um sinal elétrico em sua entrada, pode ter seu eixo posicionado em uma determinada posição angular. Ou seja, eles não foram feitos para girar livremente, e sim para ir para uma posição escolhida de acordo com um limite, que é geralmente 180 graus. Por serem pequenos, compactos, além de permitir um posicionamento preciso de seu eixo, os ser servomotores são largamente utilizado em robótica e modelismo. O IDE do Arduino já possui uma biblioteca "de fábrica" para controlarmos o servomotor, que é a Servo.h.

*Figura 16: Servomotor*

Como podem observar na figura acima, o servomotor possui 3 fios: 2 para alimentação (vcc e gnd) e um para controlar sua posição. Este controle da posição é feito através do PWM.

A conexão do servo no Arduino é simples. O fio escuro (geralmente preto ou marrom) é o gnd e obviamente deve ser ligado no pino gnd. O vermelho é o vcc e deve ser ligado no pino de 5v. E por fim o fio amarelo ou laranja é o fio de controle e deve ser ligado em algum pino que possua PWM.

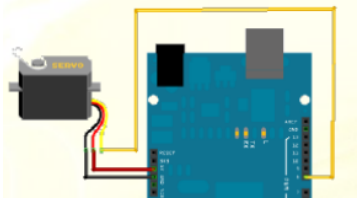

*Figura 17: Ligação do servomotor*

Com as ligações devidamente realizadas agora vamos controlar o nosso servomotor utilizando a biblioteca já inclusa no Arduino. Antes de mais nada, temos que importa-la. Como em C, utilizamos o comando #include <Servo.h>. Mas e agora, como se utiliza essa biblioteca ?

1 - Para começar declaramos uma variável do tipo Servo: **Servo motor1;** por exemplo. No nosso caso de Arduino UNO podemos criar no máximo 8 servos.

2 – Temos que associar a variável ao pino que ligamos anteriormente, e isso é feito da seguinte forma: **motor1.attach(pino);**

3 - Temos que escrever o ângulo para o qual o servo deverá girar (entre 0 e 180): **motor1.write(ângulo);**

4 – Podemos também desejar saber o ângulo que o meu servo se encontra, para isso: **ang = motor1.read ();**

5 – Podemos utilizar também as funções **motor1.attached()** para checar se o servo está associado a algum pino e também a **motor1.detach()** para desassociar o servo ao seu pino declarado do item 2 acima.

Agora, vamos ao exemplo:

```
#include <Servo.h>// incluindo a biblioteca

Servo motor1; // declarando o nosso motor como tipo servo
int pos = 0; //variavel para armazenar a posição do servo

void setup()
{
  motor1.attach(8); //associando o pino 8 ao servo
}

void loop()
{
  for(pos = 0; pos < 180; pos += 1)  // fazer com que o servo vá de 0 a 180 graus
  {                                  // de 1 em 1 grau
    motor1.write(pos);               // escrevendo a posição no servo
    delay(15);                        // esperar 15ms
  }
  for(pos = 180; pos>=1; pos-=1)     // fazer com que o servo vá de 180 a 0 graus
  {
    motor1.write(pos);               // escrevendo a posição no servo
    delay(15);                        // esperar 15ms
  }
}
```

Agora para praticar, você pode controlar um servo com um potenciômetro.

# 11. Referências


1. http://arduino.cc/
2. http://www.robotizando.com.br/
3. http://www.cursodearduino.com.br/