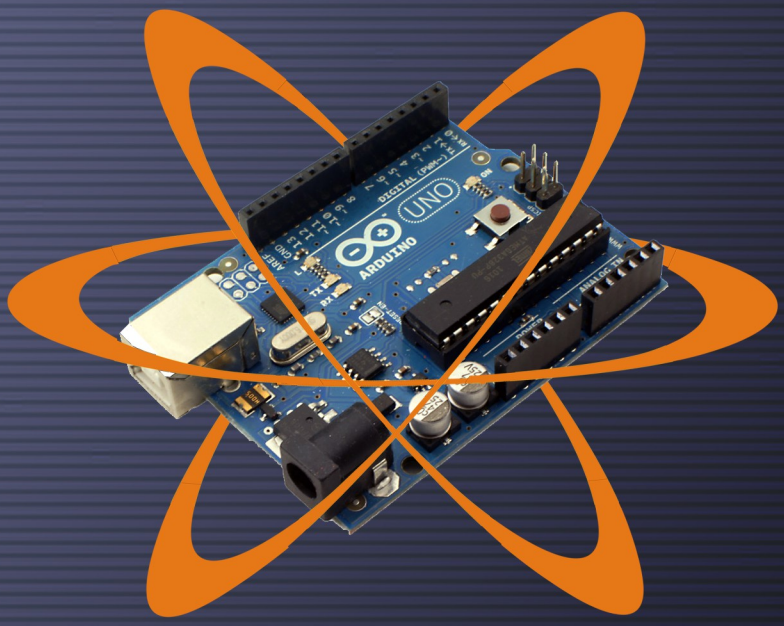
# Minicurso Arduino

Plataforma Open-Source de prototipagem

•Teoria •Projetos •Oficina




Edição 2.1

UDESC Joinville

COLMÉIA
Grupo de Pesquisa em Software Livre

# Sumário



Criado por **Alan Vitor Fonseca**
para o Minicurso Arduino, Junho 2012, Joinville-SC

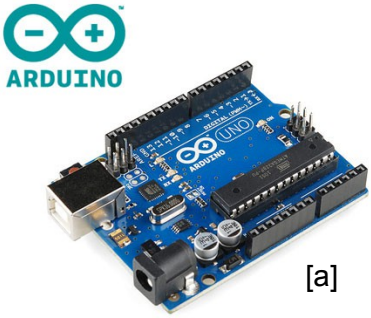
Revisado por **Daniel Scheidemantel Camargo e Gian Nunes**

**Grupo Colméia**
Centro de Ciências Tecnológicas
Universidade do Estado de Santa Catarina
Minicurso Arduino, Setembro de 2013, Joinville-SC

# Introdução

Aprender eletrônica não é uma tarefa simples, exige muito tempo e dedicação, conciliar isso com o resto das nossas atividades diárias acaba sendo um desafio. Tudo isso junto com os altos custos envolvidos, acabam por inviabilizar a criação de dispositivos eletrônicos para muitas pessoas.

Pensando nisso, o co-fundador do Arduino, Massimo Banzi do *Interaction Design Institute Ivrea* na Itália, mobilizou um grupo de pessoas a fim de criar uma solução para este problema, com algo mais barato e fácil de usar do que as plataformas já existentes na época. Eles decidiram fazer isso como um projeto de código aberto, assim todos poderiam ajudar e contribuir. Dessa ideia surgiu o Arduino.



[a]

Arduino é uma plataforma *open-source* de prototipagem com software e hardware flexível e fácil de usar [1]. De uma forma menos técnica, Arduino é um conjunto de ferramentas que possibilitam a criação de aparelhos eletrônicos onde a ideia é fazer com que as pessoas possam fazer o que quiserem com eletrônica, não necessariamente aprendendo tudo antes. É como querer fazer a sua árvore de natal piscar, é complicado fazer se você tiver que aprender eletrônica, mas com o Arduino você pode criar isso em poucos minutos.

Para entendermos melhor como funciona o Arduino, vejamos o que é um microcontrolador:

[b]

***Microcontrolador:*** *Basicamente um circuito integrado em um único chip contendo um microprocessador (CPU) e todos os periféricos essenciais ao seu funcionamento, como memórias RAM e ROM, portas de entrada e saída, time, etc.*
*O microcontrolador executa um programa específico contendo instruções para que uma certa entrada seja processada gerando uma saída correspondente.*

O Arduino torna fácil a criação de programas e a gravação do código em um microcontrolador (ATmega8, ATmega168, ATmega328, entre outros, dependendo da versão da placa) e utilização de suas portas de entrada ou saída.

Uma das grandes vantagens do Arduino é o seu licenciamento sendo ***GPL*** para a IDE Java, ***LGPL*** para as bibliotecas em C/C++ e ***Creative Commons Atribution Share-Alike*** para os esquemas da placa. Por isso qualquer pessoa pode contribuir criando uma biblioteca para simplificar um processo complexo, tornando o mesmo extremamente simples, ou então criando versões da placa para algum objetivo especifico ou até mesmo criando exemplos e tutoriais. Isso faz com que uma simples pesquisa na internet traga inúmeras soluções para o que precisamos.

# Programa

Vamos criar o nosso programa na própria IDE (Ambiente Integrado para Desenvolvimento) do Arduino. A linguagem, baseada em C/C++, pode ter um paradigma orientado à objetos (indicado) ou estruturado.

### Sintaxe

Diferente da linguagem C original, a que usaremos terá algumas diferenças, por exemplo, aqui ao invés de uma função principal - *int main()* - teremos no mínimo dois ambientes, que basicamente terão a seguinte forma:

```
void setup(){
  ...
}
void loop(){
  ...
}
```

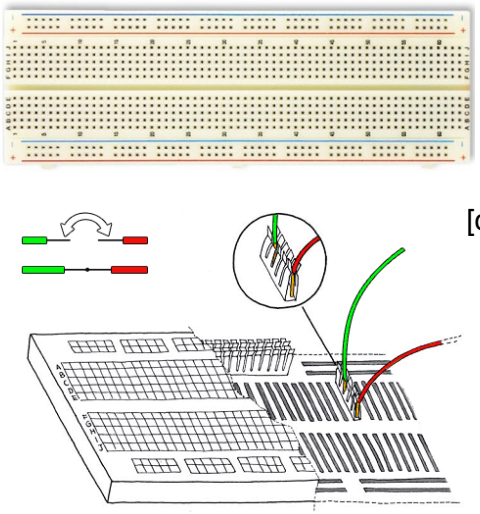
***void setup()*** é usada para fazer a configuração. Como dizer se um pino será entrada ou saída e configurar a taxa da comunicação serial. Essa função é executada uma vez no início do programa.

***void loop()*** é usada para escrever a rotina a ser executada pelo microcontrolador.

Este é o esqueleto básico de um programa, porém como em C/C++ podemos incluir bibliotecas adicionais, declarar variáveis globais, escrever outras funções fora das citadas acima, entre outras funcionalidades.

## Colocando a mão na massa

### Protoboard



A protoboard é uma placa usada para criação de protótipos ou circuitos experimentais onde os componentes e as conexões podem variar sem a necessidade de solda. É uma matriz de contato, com furos tendo conexões verticais e horizontais.

Ela funciona de forma que ao conectarmos um fio no primeiro furo da coluna vertical, este estará em curto circuito com qualquer outro fio conectado na coluna, ou seja, é como se conectássemos diretamente todos os fios da coluna. Com as linhas horizontais o funcionamento é o mesmo.

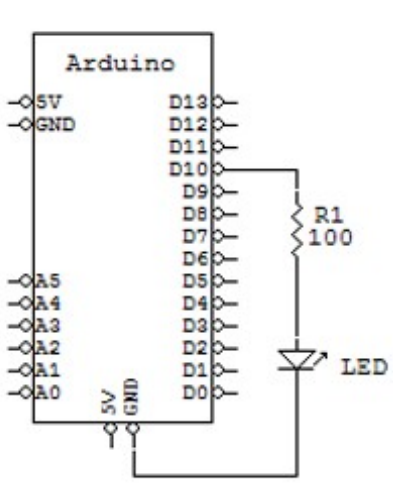
Vamos agora escrever o nosso primeiro programa, mas como o Arduino não fica somente na tela do PC, vamos criar um projeto bem simples e trabalhar em cima dele.

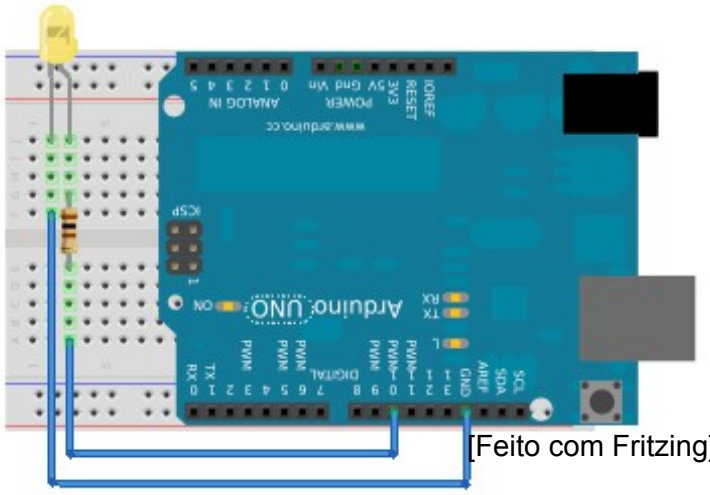
### Projeto 1: Piscar um LED

Começamos então por um projeto clássico, “Pisca LED” , ou seja, queremos fazer um LED ficar aceso por um tempo, depois apagar e assim por diante.

Para começar a escrever o programa precisamos saber em que pinos do Arduino estarão

conectados nossos componentes, nesse caso, somente o LED, então vamos ao nosso circuito elétrico:



[Feito com Fritzing]

Agora, com a parte do hardware já montada, vamos para o programa:

→ Como conectamos nosso LED no pino 10 do Arduino, vamos declarar uma variável do tipo inteiro para guardarmos o número do pinos em que o LED está.
*(esta será uma variável global)*

```
int LED = 10;
```

→ Agora vamos configurar o pino 10 como uma saída

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

→ Escrevemos agora a rotina para acender, esperar algum tempo e depois apagar o LED.

Declaramos a função loop{

Setamos o pino do LED para nível lógico alto;

Esperamos um tempo (1000 ms = 1 s);

Setamos o pino do LED para nível lógico baixo;

Esperamos um tempo (1000 ms);

Fechamos a função loop}

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

Vejamos o que as linhas mais relevantes significam:

| | |
|---|---|
| → pinMode(pino, modo) | Configura um pino específico para se comportar como uma entrada ou uma saída (INPUT ou OUTPUT). |
| → digitalWrite(pino, valor) | Escreve em um pino digital o valor lógico Alto ou Baixo. (HIGH ou LOW) |
| → delay(tempo) | Pausa o programa durante um certo tempo especificado em milissegundos (ms). |

*Obs.: No próprio site do Arduino ( http://Arduino.cc/en/Reference/HomePage ) podemos encontrar referência sobre a linguagem utilizada, o site traz , entre outras coisas, referências sobre a sintaxe, operadores e funções.*

Antes de irmos para o próximo projeto vamos ver algumas noções de conversão numérica e álgebra booleana.

## Conversão numérica

Apesar de que na IDE do Arduino as funções são feitas de forma a facilitar ao máximo o uso dos números e valores usados, ainda em alguns casos é necessário sabermos fazer a conversão de base de um número.

Mas o que é a base de um número? - De uma forma fácil, a base de um número é o total de "símbolos" que usamos para escrevê-lo, ou seja, a base em que mais usamos, a decimal, temos 10 símbolos(0 a 9) para escrever os números que desejarmos e na base binária 2 símbolos (0 e 1).

Como os microcontroladores e até mesmo os computadores, internamente só "entendem" números na base binária, é importante que saibamos escrever números decimais em pelo menos mais duas bases, a binária e a hexadecimal, que também e largamente usada nesse meio.

***Conversão com base decimal para uma base qualquer:*** podemos fazer isso por meio das divisões sucessivas:

vamos converter o número $47_{10}$(base 10) para a base hexadecimal(base 16)

$47/16 = 2$ , resto: $15_{10} = F_{16}$

$2/16 = 0$ , resto: $2_{10} = 2_{16}$

Dividimos o número pela base em que queremos transformar, no caso 16, até que o número zere.

Dai tiramos o nosso número na base hexadecimal $2F_{16.}$

**Conversão com base decimal para uma base qualquer:** podemos fazer isso multiplicando cada algarismo do número por potencias de sua base elevadas a ordem significativa do número, e depois somando todos eles.

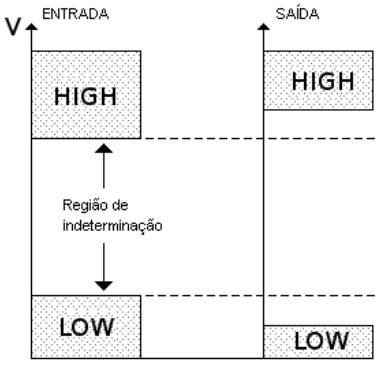
Vamos converter agora o número $2F_{16}$ para a base decimal:

$2F_{16} = 2*16^1 + 15*16^0 = 32 + 15 = 47_{10}$. (visto que $F_{16} = 15_{10}$).

**Conversão com base hexadecimal e binária:** essa conversão fica fácil ao sabermos que, em ordem, cada 4 algarismos binários equivalem a um algarismo hexadecimal, e vice e versa.

| | | |
|---|---|---|
| Valor Numérico | 8 4 2 1 | 8 4 2 1 |
| Número Binário | 0 0 1 0 | 0 1 1 0 |
| Número Hexadecimal | 2 | 6 |

Portanto $26_{16} = 100110_2$




## Álgebra booleana

Antes de tudo devemos ter em mente:

`Sistemas digitais usam a lógica binária de forma que`

os zeros e uns são os **níveis lógicos** nas entradas ou saídas dos dispositivos, os quais trabalham com baixas tensões, no caso dos microcontroladores da Atmel, 5V (entre 1.8 à 5.5V), portanto uma entrada na faixa próxima à 0V é entendida como nível lógico *baixo(LOW)*, já uma faixa em torno de 5V e entendida como nível lógico *alto (HIGH)*.

Na álgebra booleana trabalhamos com esses níveis lógicos de forma que 0 significa *falso* e 1, *verdadeiro*, e usamos algumas operações lógicas:

| | | |
|---|---|---|
| OR | Ou | + |
| AND | E | * |
| NOT | Não | ~ |
| NAND | Não AND | AND invertido(barrado) |
| NOR | Não OR | OR invertido(barrado) |
| XOR | Exclusive OR | ⊕ |

Por exemplo, se tivermos como entrada 3 sinais lógicos, verdadeiro, falso e verdadeiro de novo, e queremos realizar entre eles uma operação lógica AND, temos:

verdadeiro AND falso AND verdadeiro = falso

da mesma forma que:

1*0*1 = 0.

Assim como a operação OR: verdadeiro OR falso OR verdadeiro = verdadeiro → 1+0+1 = 1.

Obs.: Observe que estamos executando uma operação lógica “soma” e não uma soma como estamos acostumados, portanto aqui 1+1 = 1.

A operação lógica NOT inverte o nível lógico, ou seja o que era 0 vira 1 e vice e versa, e é representada por uma barra em cima da função lógica:

(verdadeiro OR falso)AND verdadeiro = (verdadeiro NAND falso)AND verdadeiro = falso → (1+0)*1 = 1*1 = 0*1 = 0.

Já a função XOR e dada como: $a \oplus b = ab + \bar{b}a$ .

## Projeto 2: Semáforo

Nosso objetivo aqui é fazer um programa que controle o trânsito de veículos em uma rua.

Teremos um semáforo convencional de veículos com os 3 passos mostrados ao lado. Sendo que para efeito experimental deixaremos o sinal em vermelho e verde por 5 segundos e amarelo por 1 segundo.

1 2 3

Também teremos um botão que ao pressionarmos o sinal ira impedir a passagem de veículos, isso é, caso o sinal esteja verde ele deve ir imediatamente para vermelho.

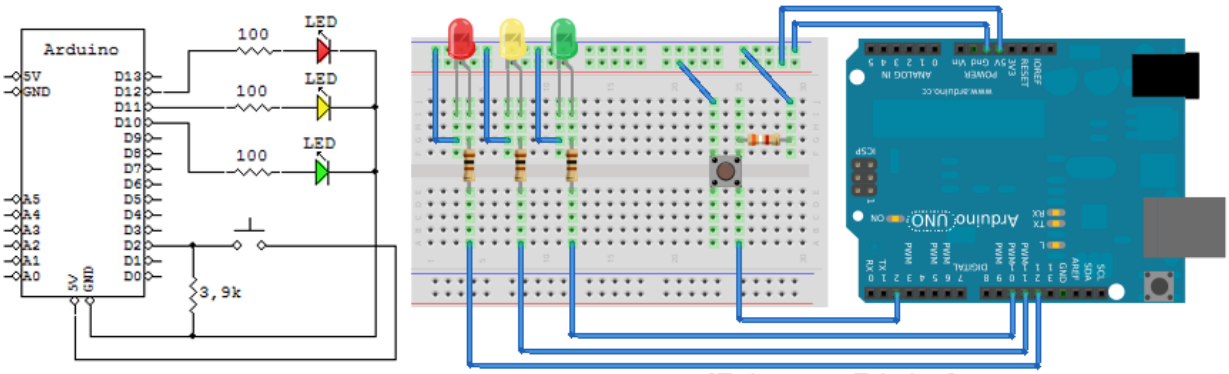
Para isso usaremos algo chamado interrupção externa, que caso ocorra ela irá desviar o programa de sua execução atual, realizar alguma ação (uma função no nosso caso), e depois continuar o programa normalmente.
Uma interrupção pode ser atribuída a um pino do Arduino através da função:

→ attachInterrupt(canal, função, modo) , onde:

canal : indica o pino a ser usado pela interrupção – canal 0 para o pino 2 e 1 para o pino 3.
função : é a função que será executada quando ocorre a interrupção.
modo : é o tipo da interrupção, que pode ser:

→ LOW : interrupção dada quando o pino estiver com nível lógico baixo.
→ CHANGE : interrupção dada quando o pino muda de nível lógico.
→ RISING : interrupção dada quando o nível lógico vai de 0 para 1.
→ FALLING : interrupção dada quando o nível lógico vai de 1 para 0.

Vamos então ao hardware, que será montado com LEDs e um *pushbutton:*




[Feito com Fritzing]

O programa fica assim:

```
int passo = 1;

int vermelho = 12, amarelo = 11, verde = 10;

int botao = 2;

void setup(){

  attachInterrupt(0, irVermelho,FALLING);

  pinMode(botao,INPUT);

  pinMode(verde,OUTPUT);
  pinMode(amarelo,OUTPUT);
  pinMode(vermelho,OUTPUT);
}

void irVermelho(){
  p1();
  passo = 1;
}
```

→ Variável global para armazenar o passo atual.

→ Declaração dos LEDs usados.

→ Declaração do botão conforme o circuito.

→ Aqui temos a função de configuração.

→ Vinculamos a função irVermelho() com uma interrupção externa na borda de descida do canal 0 ( no pino 2).

→ Configuramos o pino do botão como entrada.

→ Configuramos os pinos dos LEDs como saídas digitais.

→ Essa função liga o LED vermelho e desliga os outros, e retorna o passo para o primeiro*.

```
void p3(){
  digitalWrite(vermelho,LOW);
  digitalWrite(amarelo,HIGH);
  digitalWrite(verde,LOW);
}

void p2(){
  digitalWrite(vermelho,LOW);
  digitalWrite(amarelo,LOW);
  digitalWrite(verde,HIGH);
}

void p1(){
  digitalWrite(vermelho,HIGH);
  digitalWrite(amarelo,LOW);
  digitalWrite(verde,LOW);
}

void loop(){


  if(passo == 1){
    p1();
    passo = 2;
    delay(5000);
  }

  if(passo == 2){
    p2();
    passo = 3;
    delay(5000);
  }

  if(passo == 3){
    p3();
    passo = 1;
    delay(1000);
  }
}
```

→ Função para colocar os LEDs no passo 3.

→ Função para colocar os LEDs no passo 2.

→ Função para colocar os LEDs no passo 1.

→ Função loop – aquela executada sempre que o microcontrolador estiver alimentado.

→ Aqui caso a variável que guarda o passo atual estiver em 1, executamos a função que coloca os LEDs no passo 1, então mudamos a variável passo para indicar o novo passo e esperamos 5.000ms ou 5s.

→ Faz o mesmo que o anterior porém para o segundo passo.

→ Executa o mesmo para o terceiro passo, porém agora esperando apenas 1 segundo no sinal amarelo.

* Em uma situação normal poderíamos executar p1() e esperar 5s na própria função de interrupção e depois ir direto para o passo 2, porém aqui a função delay() não funciona. Nesse caso como o tempo é pequeno quase não vemos problema, porém quando o botão é pressionado o passo 1 fica por mais tempo ( o tempo que o programa já estava esperando no passo em que estava mais o tempo de 5 segundos de p1() ). Como exercício para o minicurso tente resolver este problema.
No final da apostila temos mais um projeto com semáforo mais próximo à realidade com esse problema resolvido.

Esta é somente uma forma de fazer o programa, poderíamos, por exemplo mudar os estados dos LEDs na função loop ao invés de criar funções separadas para isso. A lógica de programação é muito importante nesses casos. Podemos resolver um problema aparentemente muito difícil de uma forma fácil se usarmos a lógica certa.

**A função tone**

Esta função gera uma onda quadrada em uma saída do Arduino com uma frequência especificada. Vejamos como é a função:

→ tone(pino, frequência, duração), onde a duração é um argumento opcional.

Obs.: Nas placas do Arduino que usam o microcontrolador Atmega8, se usado, o argumento "duração" causa um erro no programa. Podemos usar então a função noTone(pino).

Com essa função podemos criar sons, sabendo a frequência de uma nota musical por exemplo podemos tocá-la em um auto falante ou um piezo-elétrico ligado ao Arduino.

# Comunicação Serial

Além de funções prontas, o Arduino possui, na sua própria IDE, o "Serial Monitor" que é uma interface para fazer o debug de programas, assim como enviar e receber dados de forma serial. Nas placas do Arduino que possuem conexão USB, essas podem ser usadas para fazer a comunicação serial com um computador por exemplo.

Primeiro devemos fazer a configuração para a comunicação serial, através da função:

→ Serial.begin(taxa) , onde:
→ taxa: é a taxa de transmissão de dados, em bits por segundo (baud).

Agora podemos enviar dados com as funções Serial.print(), Serial.println() e Serial.write(), e ler os dados recebidos com Serial.read() e Serial.peek().
Para checar se há dados recebidos podemos usar a função Serial.available(), que retorna no número de bytes disponíveis para leitura.

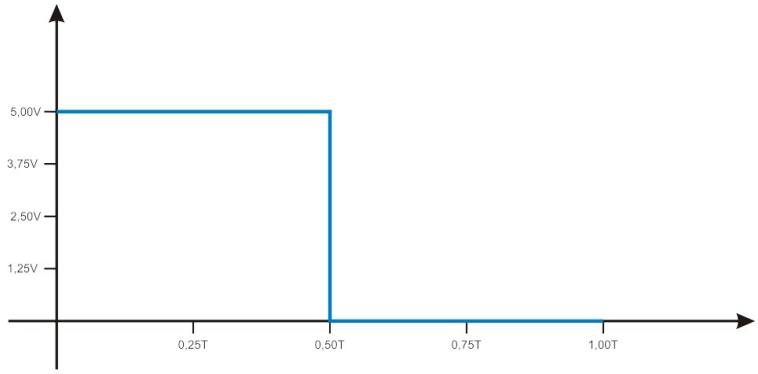
# Entradas e saídas analógicas

Vejamos a seguir como podemos usar as entradas e saídas analógicas do Arduino.

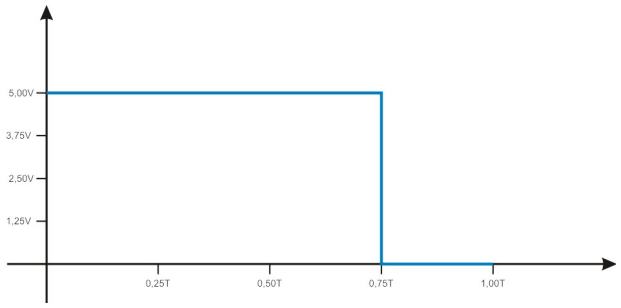
**Saídas analógicas**

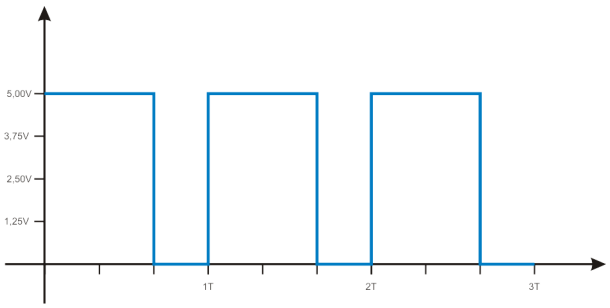
As saídas analógicas do Arduino usam PWM – ou *Modulação por Largura de Pulso* - para variar a transferência de potência em um pino especifico.

Vejamos então como funciona o PWM. Quando temos o nível lógico 1 em um pino do Arduino e mudamos para 0, temos um sinal como na figura à esquerda:




Já quando variamos repetidamente a mudança de nível lógico temos algo como na figura à direita, o que caracteriza uma "onda quadrada", nesse caso se considerarmos um período, veremos que o pino transferiu a metade da potência que transmitiria se estivesse o tempo todo em nível lógico 1. O PWM consiste em manipularmos o tempo, em cada período, que ficaremos em cada um dos níveis lógicos.




No caso da figura acima à direita, por exemplo, temos 75 por cento de potência transferida, porém no Arduino o período dessas ondas é muito baixo, fazendo com que a frequência seja alta (aproximadamente 490 Hz), portanto, se estivermos com um LED conectado a esse pino, não o veremos piscar, mas sim com 75% de sua luminosidade total. Em motores DC podemos controlar a velocidade do motor através de PWM.

O Arduino possui alguns pinos que podem ser usados como PWM, e para isso usamos a função:

→ analogWrite(pino, valor) , onde:

→ pino : será o pino de saída do PWM.
→ valor : valor varia entre 0 ( 0%) e 255 ( 100%) que determina a porcentagem em que o sinal estará ativo (precisão de 8 bits).

Obs.: Não é preciso declarar o pino como saída (usando pinMode()) para usá-lo PWM.

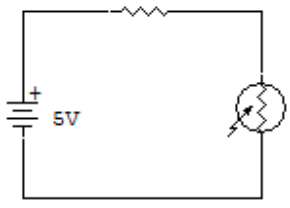
**Entradas Analógicas**

Um pino de entrada analógica tem a função de ler a tensão nele colocada e transformar esse valor em um número inteiro, no caso do Arduino teremos a precisão de 10 bits, ou seja 1024 valores possíveis, sendo 0 para 0V e 1023 para 5V.
A função usada para leitura analógica no Arduino e a seguinte:

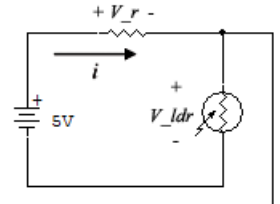
→ analogRead(pino) , onde “pino” indica o pino o qual será feito a leitura.

→ Essa função retorna um valor inteiro de 0 a 1023 correspondente a leitura feita.

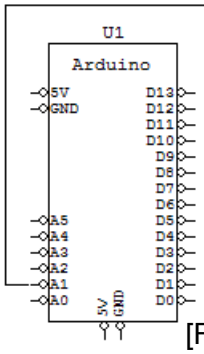
***Como funciona***




No ensino médio você já deve ter estudado alguns circuitos com resistores e viu que a corrente que circula em um circuito é dada por $i=\frac{V}{R_e}$ , onde $R_e$ é a resistência equivalente do circuito, ou então que a diferença de potencial V nos polos de um resistor é $V_R=Ri$ .
Sabendo isso vejamos como ira ficar a tensão que iremos ler no Arduino:



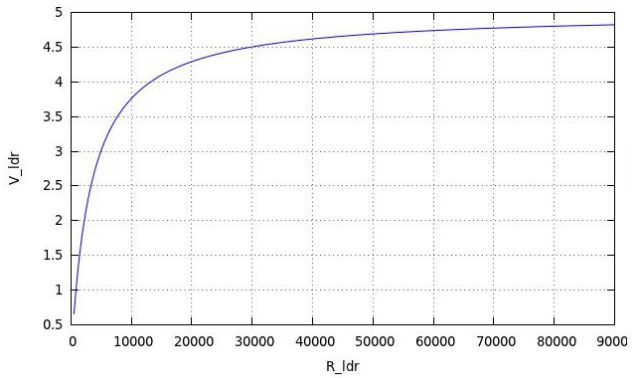

- A corrente que circula no circuito é: $i=\frac{V}{R_e}=\frac{V}{R_1+R_{LDR}}$
- A tensão que estaremos lendo esta nos polos de $R_{LDR}$ , assim:
  $V_{LDR}=R_{LDR}i=R_{LDR}\frac{V}{R_1+R_{LDR}}$ ou $V_{LDR}=\frac{R_{LDR}}{R_1+R_{LDR}}V$
  Isso é o que chamamos de divisor de tensão entre $R_1$ e $R_{LDR}$ .
- Assim, como $R_{LDR}$ varia conforme a luz que incide no LDR, podemos variar $V_{LDR}$

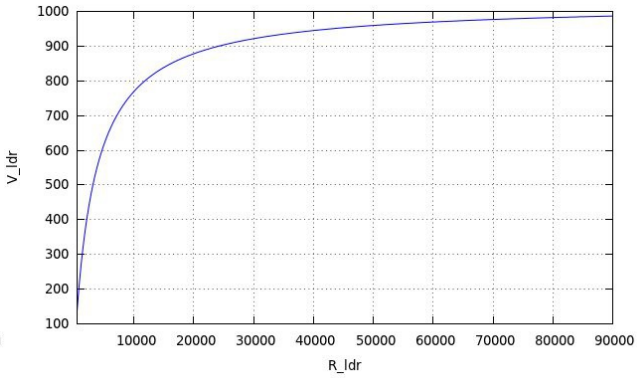



[Feito com Fritzing]

Em um teste feito em uma sala qualquer quando o LDR era submetido a muita luz obtivemos uma resistência de 500Ω, e sem luz de 90.000Ω,
com esses dados a leitura $V_{LDR}$ é mostrada no gráfico a esquerda, e como o

Arduino possui um conversor analógico/digital de 10 bits, ou seja, vai do decimal 0 ao 1023, podemos ver a leitura no gráfico a direita.



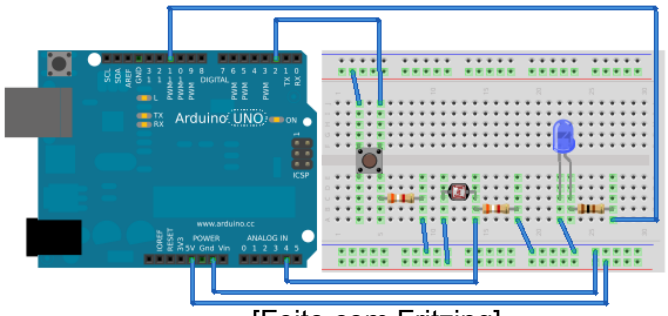




**Projeto 3: Controle de luminosidade**

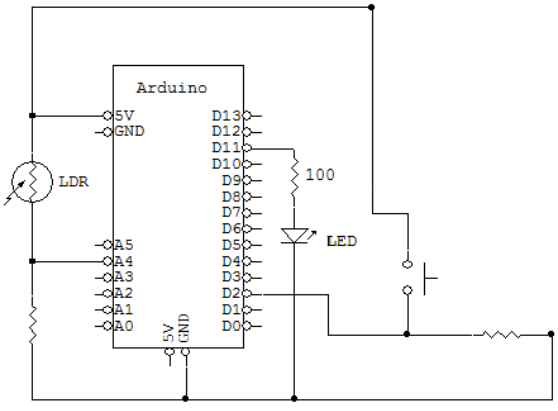
Para este projeto queremos controlar a luminosidade em um local, de forma que teremos um botão que liga/desliga o sistema de iluminação. Quando o sistema estiver ligado usaremos um LDR para verificar a luminosidade no local, quanto menor, a luz (LED) deverá ter uma intensidade maior, ou menor.

| | | |
|---|---|---|
|  | Botão | Liga/Desliga o sistema de iluminação |
|  | LDR | Usado como sensor de luminosidade. |
|  | LED | Iluminação. |

O Circuito para esse projeto está nas figuras abaixo, a programação fica como exercício para o minicurso.




[Feito com Fritzing]



Vejamos alguns passos para resolvermos o exercício:

1. Teremos que saber a faixa de valores que estamos lendo no sensor de luz (LDR).
   - Para isso começamos o nosso programa somente com uma comunicação serial, lemos os valores no pino de leitura analógica a qual o LDR esta conectado e os enviamos para o computador. Lembrando de colocar um “delay” entre cada envio.

```
\\testando o LDR

int LDR=4;

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

void loop(){
  int val = analogRead(LDR);
  Serial.println(val);
  delay(200);
}
```

2. Agora teremos de ajustar os valores lidos para escreve-los no LED.
   - O Arduino faz leituras analógica em 10 bits, ou seja, teremos uma leitura na faixa de 0 a 1023 ( $2^{10}=1024$ possibilidades) e a saída analógica usa 8 bits, então esta na faixa de 0 a 255 ( $2^{8}=256$ possibilidades). Por isso teremos de adaptar o valor lido ao valor que vamos escrever no pino do LED. Para isso usamos a função map, explicada logo a seguir.
3. Feito isso falta somente a rotina do programa.
   - Teremos que fazer uma rotina que leia o valor do LDR e escreva esse valor ajustado no LED caso o sistema estiver ligado ou então desligar o LED caso o sistema for desligado.
4. Por fim podemos limpar o código:
   - Agora que já ajustamos o que queríamos podemos retirar o “delay” e a comunicação do código fazendo com que o processamento seja mais rápido.

A seguir temos a função para ajustar os valores lidos no pino do LDR:

→ map(valor, minIn, maxIn, minOut, maxOut) , onde:

valor : é a variável que contem o valor que se deseja adaptar.
minIn : valor minimo da entrada.
maxIn : valor máximo da entrada.
minOut: valor minimo da saída.
MaxOut: valor máximo da saída.

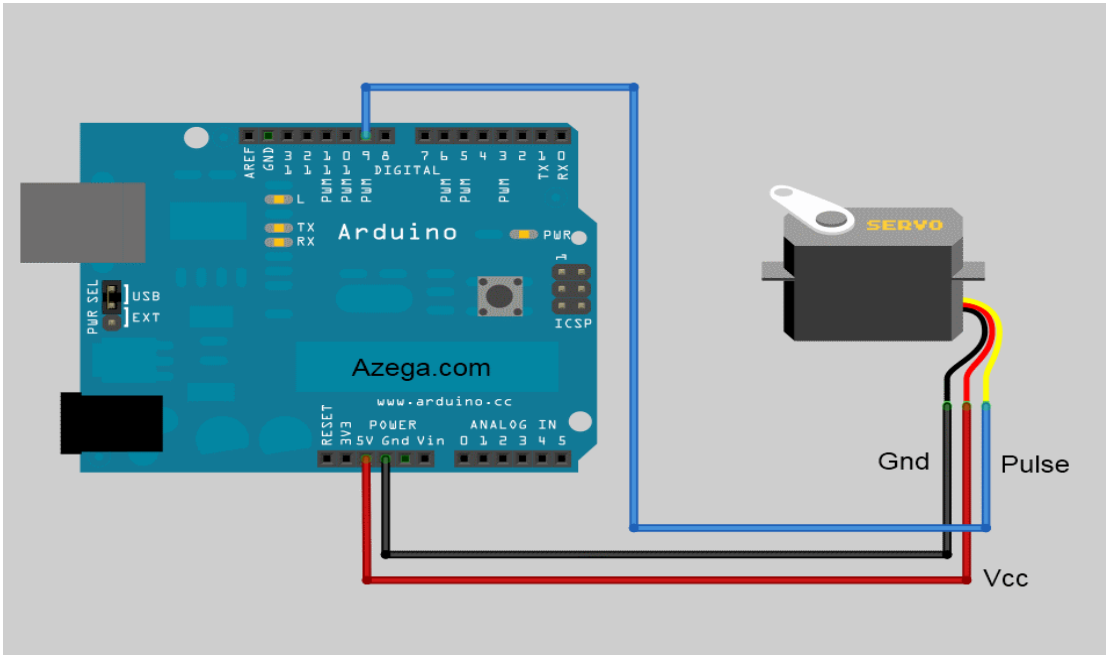
O retorno da função é um valor inteiro adaptado. Para adaptar diretamente os valores lidos podemos fazer: “int y = map(LeituraLDR, 0, 1023, 0, 255);” porém veremos que o LDR não lê exatamente de 0 a 1023 porque teremos um divisor de tensão entre o LDR e o resistor usado.

**Projeto Novo 1: Servo Motores**

Servo motores, são motores que dependem de um sinal para realizar seu movimento. Apresenta movimento proporcional a um comando, em vez de girar ou se mover livremente sem um controle mais efetivo de posição como a maioria dos motores.

Recebe um sinal de controle, verifica a posição atual e vai para a posição desejada. Possui a liberdade de cerca de 180º graus, mas são precisos quanto a posição.

Possui uma caixa de redução com relação longa que ajuda a amplificar o torque. [Fonte: wikipedia].

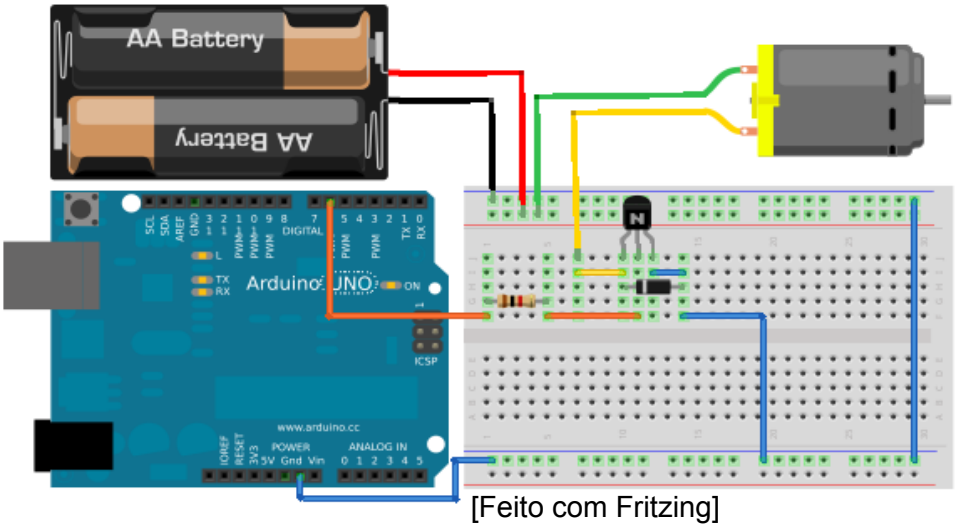



[Fonte imagem: http://ninad243.weebly.com/project64-2d-to-3d-modelling-using-image-processing.html]

**Motores DC**

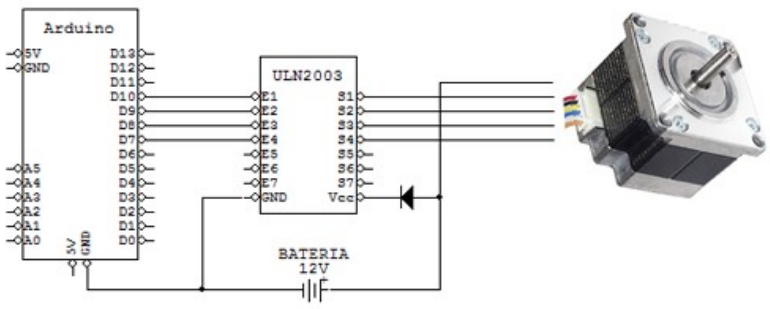
Quando falamos de motores elétricos e Arduino, é comum ouvirmos falar nos Servomotores, o qual temos uma livraria com várias funções prontas para trabalharmos com eles. Alguns desses motores são encontrados geralmente de forma que funcionam ligados diretamente aos pinos do Arduino.

Porém quando falamos de motores em geral, o Arduino geralmente não fornece a corrente e voltagem que o motor precisa, na maioria dos casos, teremos de fazer um circuito auxiliar para fazer o motor funcionar e não trazer nenhum risco à placa, como queimar algum componente. Esse circuito pode ser feito com um transistor, como na figura abaixo:




[Feito com Fritzing]

Onde usamos um transistor como se fosse um interruptor de uma lampada. Quando enviamos LOW no pino que esta conectado à base do transistor ele impede a passagem de corrente entre o coletor e o emissor, já quando enviamos HIGH ele permite a passagem de corrente fazendo o motor girar. O funcionamento é similar para o uso de PWM (escrita analógica no pino). O transistor mais comum usado é o BC548, mas caso o motor precise de uma corrente maior, podemos usar o TIP122 ( até 5A) lembrando que este ultimo possui a pinagem diferente, por isso devemos fazer algumas alterações no circuito.

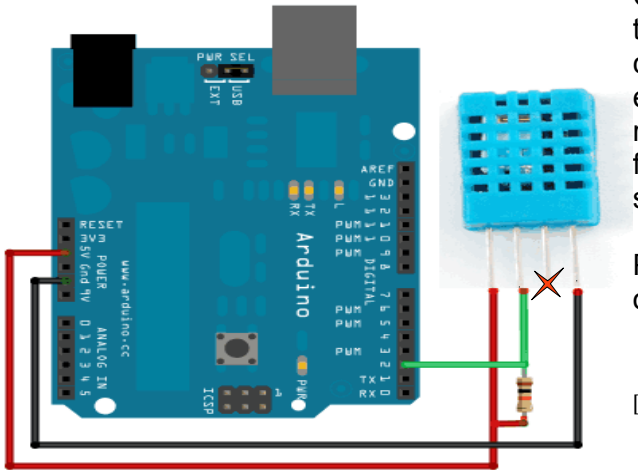



No caso de um motor de passo podemos usar o Circuito integrado ULN2003 que possui um conjunto de transistores Darlington. Com este CI podemos conectar cada transistor a uma bobina do motor e com o Arduino podemos ligar e desligar cada bobina separadamente fazendo o motor girar uma quantidade especifica de graus.

Um transistor possui 3 pinos, uma base, um coletor e um emissor, no caso do ULN2003, as entradas(base) são barradas, ou seja, seu sinal é invertido, portanto se o alimentarmos com 12V (coletor), e tivermos nível lógico 0 na entrada então teremos 12V na saída(emissor), e no caso de nível lógico 1 na entrada teremos uma saída de 0V.

ATENÇÃO: Para evitar que algum componente seja queimado, devem ser observadas as especificações de corrente e tensão suportadas.

**Projeto Novo 2: Sensor de Temperatura e Umidade DHT***



O sensor da linha DHT* faz o sensoreamento de temperatura e umidade através de uma saída digital calibrado. Possui uma boa confiabilidade e estabilidade a longo prazo, tornando-o atrativo ao relacionar custo-benefício. O sistema de interface é feito no modo one wire, tornando a ligação mais simples.

Para o experimento será utilizado o modelo DHT11, o qual possui como faixa de leitura:

- Temperatura de 0 a 50ºc, erro de +/- 2ºc,
- Umidade: 20 a 90% RH, erro de +/- 5%;

[Fonte texto: http://www.micro4you.com/files/sensor/DHT11.pdf]

[Fonte imagem:http://openhardwarebrasil.org/blog/?p=115]

**Projeto Novo 3: LCD 16x2**

Um diferencial em qualquer projeto microcontrolado é utilização de um display LCD para indicação de parâmetros e informações diversas. Um simples Display LCD 16x2 torna o projeto muito mais amigável ao usuário além de aumentar a gama de funções e operações com um custo relativamente baixo.

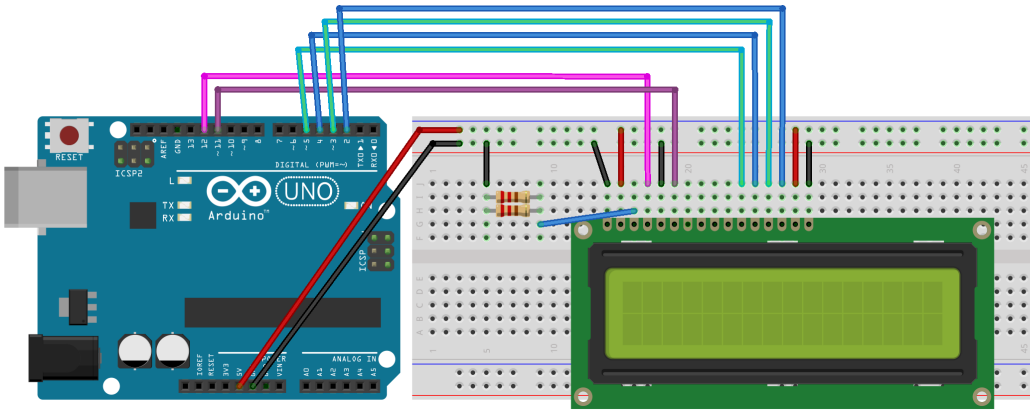
O fato da simplicidade de como se configura um Display LCD para trabalhar em conjunto com um Arduino faz com que se pense duas vezes em não utilizá-lo em seus projetos.

O modelo utilizado neste material é o JHD 162A, possui backlight com ajuste de luminosidade e contraste dos caracteres controláveis. É fácil de encontrá-lo no mercado por ter o preço mais

acessível. [Fonte: http://www.marcelomaciel.com/2012/03/configuracao-display-lcd-no-pic.html]

A IDE do Arduino possui alguns códigos de exemplo prontos. Observe que ao carregar qualquer um destes códigos, a configuração dos pinos se dá através da função:
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

É utilizando esta configuração que montamos o sketch abaixo na protoboard:



[Feito com Fritzing]

**Projeto 4: Verificação de superfície**

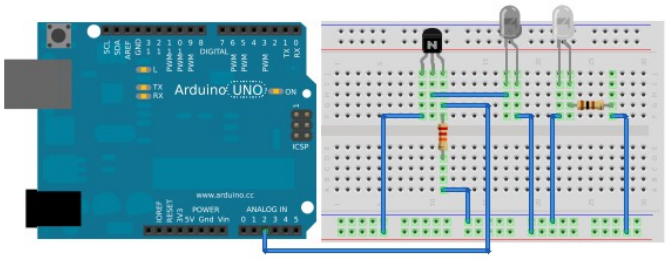
Existem robôs que seguem uma linha no chão em competições ou até mesmo em indústrias. Veremos nesse projeto como alguns desses robôs fazem isso.

Na figura a esquerda temos sensores e um emissor de luz infravermelha, eles são usados, por exemplo, nas TVs, onde o controle remoto é o transmissor e um sensor junto a TV é o receptor infravermelho.

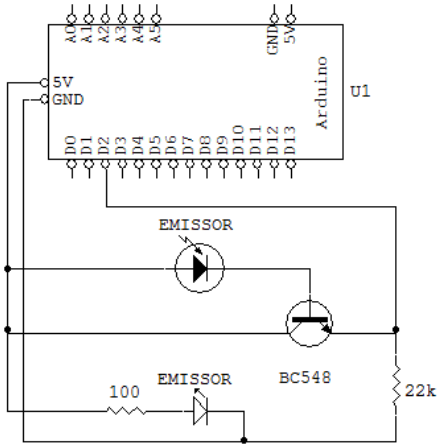
Você também já deve ter ouvido falar que a cor preta não reflete os raios de luz, isso também ocorre com o infravermelho, quando temos um arranjo como na figura ao lado, os raios de luz infravermelha saem do emissor, refletem no obstáculo e chegam até o receptor. Se esse objeto for de cor preta a reflexão será bem menor do que se ele for de cor branca, com isso podemos distinguir, por exemplo, uma linha escura em um chão claro.

O objetivo desse projeto e distinguir um objeto branco de um preto através desses sensores.

Este projeto também fica como exercício para o minicurso, uma possível implementação está no final desse material. Vamos ao circuito usado no projeto:



[Feito com Fritzing]




## Projetos extras

Aqui temos alguns projetos parecidos com os anteriores, porém com alguns problemas a mais, vejamos a solução deles.

## Projeto Extra 1: Semáforo

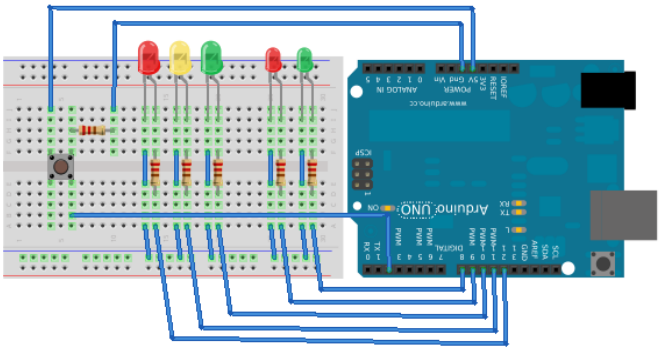
Agora temos um semáforo com o sistema um pouco mais próximo ao que encontramos nas ruas. Nosso objetivo aqui é fazer um programa que controle o trânsito como no projeto 2 porém agora teremos um grupo veicular e um de pedestres.

O semáforo será como mostrado abaixo, onde o sinal com 3 cores é o veicular e o com 2 cores o dos pedestres, e obedecendo as regras convencionais de um semáforo.

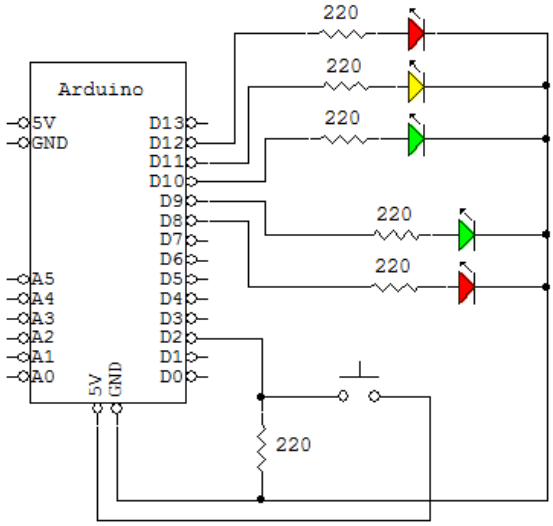
1 2 3 1 2/3

Porem, além disso, teremos um botão que quando pressionado reduzirá o tempo que o pedestre tem de esperar para passar.

Vamos então ao hardware, que será montado com 5 LEDs e um *pushbutton:*



[Feito com Fritzing]




O programa fica assim:

```
int tempo = 0;                          → Variável global inicializada com zero.

void reduzTempo(){                      → Função para reduzir o tempo de espera caso seja
  if(tempo >= 10 && tempo <=19)           apertado o botão.
```

```
    tempo = 19;
}

void setup(){

  attachInterrupt(0, reduzTempo,FALLING);


  for(int i = 8 ; i <=12 ; i++){
    pinMode(i,OUTPUT);
  }
  pinMode(4,INPUT);
}

void p3(){              //carro atenção
  digitalWrite(12,LOW);  // Vermelho
  digitalWrite(11,HIGH); // Amarelo
  digitalWrite(10,LOW);  // Verde
  digitalWrite(9,LOW);    // Pedestre-Seguir
  digitalWrite(8,HIGH);   // Pedestre-Parar
}



void p2(){              //carro andando
  digitalWrite(12,LOW);
  digitalWrite(11,LOW);
  digitalWrite(10,HIGH);
  digitalWrite(9,LOW);
  digitalWrite(8,HIGH);
}

void p1(){              //carro parado
  digitalWrite(12,HIGH);
  digitalWrite(11,LOW);
  digitalWrite(10,LOW);
  digitalWrite(9,HIGH);
  digitalWrite(8,LOW);
}

void piscaVermelho(){
  for(int x = 0; x<= 3;x++){
    digitalWrite(8,HIGH);
    delay(500);
    digitalWrite(8,LOW);
    delay(500);
  }
}

void addTempo(){
  delay(1000);
  tempo++;
  if(tempo > 22) tempo = 0;
}

void loop(){
```

→ Aqui a nossa função de configuração

→ Vinculamos nossa função reduzTempo() com uma interrupção externa na borda de descida do canal 0 ( no pino 2).

→ Configuramos os pinos de 8 a 12(LEDs) como saídas digitais.

→ Configuramos o pino 4 (*pushbutton*) como entrada digital.

→ Função para colocar os LEDs no passo 3.

→ Função para colocar os LEDs no passo 2.

→ Função para colocar os LEDs no passo 1.

→ Função para piscar 3 vezes o LED que indica para os pedestres pararem.

→ Função usada para adicionar 1 segundo (1000ms) na variável que guarda o tempo.

→ Função loop – aquela executada sempre que o

| | |
|---|---|
| | microcontrolador estiver alimentado. |
| `if (tempo == 0) p1();` | → Para o tempo entre 0 e 9 segundos colocamos os sinais na posição 1. |
| `if (tempo == 10){`<br>`  digitalWrite(9,LOW);`<br>`  piscaVermelho();`<br>`  p2();`<br>`}` | → Para o tempo entre 10 e 19 segundos,piscamos o LED vermelho e depois mudamos os sinais para a posição 2. |
| `if (tempo == 20) p3();` | → Para o tempo entre 20 e 22 segundos colocamos os sinais na posição 3. |
| `addTempo();`<br>`}` | → Executamos a função para adicionar um segundo a cada loop. |

Esta é somente uma forma de fazer o programa, poderíamos, por exemplo mudar os estados dos LEDs na função loop ao invés de criar funções separadas para isso.
A lógica de programação é muito importante nesses casos, no programa acima, consideramos que o tempo do ciclo não sairá da faixa citada, e que não haja erro na sua adição, se pulássemos do segundo 9 para o 11 , por exemplo, não teríamos a mudança dos sinais para os segundo estado.

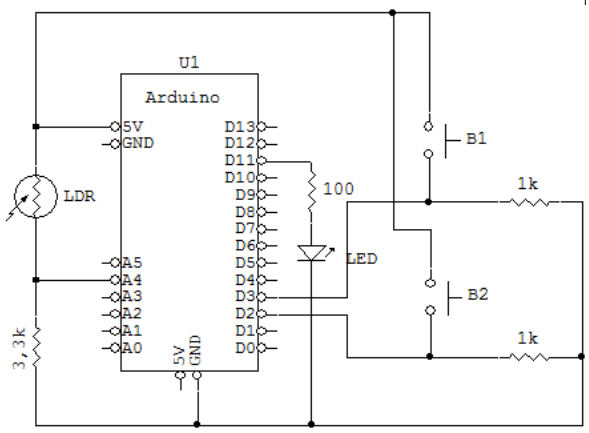
**Projeto extra 2: Controle de luminosidade**

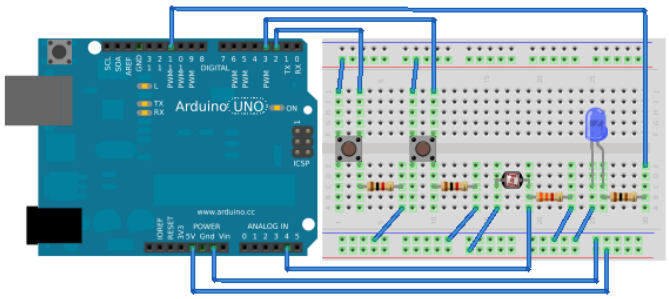
Para este projeto queremos controlar a luminosidade em um local, de forma que teremos 2 botões, um deles liga/desliga o sistema de iluminação, e o outro quando pressionado iniciará ou adicionará tempo um “timer” que desligará a iluminação quando o tempo terminar.

Quando o sistema estiver ligado usaremos um LDR para verificar a luminosidade no local, quanto menor, a luz (LED) deverá ter uma intensidade maior.

| | | |
|---|---|---|
|  | Botão 1 | Liga/Desliga o sistema de iluminação |
| | Botão 2 | Seta um timer para desligar o sistema. |
|  | LDR | Usado como sensor de luminosidade. |
|  | LED | Iluminação. |

Circuito usado no projeto:





[Feito com Fritzing]

Código fonte:

cialização das variáveis:
mpo do timer.
mpo decorrido durante o timer.
mpo de segurança contra mau contato.

urante que o botão só dê efeito se apertado
s de 500 ms, isso previne mal contato nos fios.

```
    tempo+=10000;

    if(tempo >=50000){
      tempo = 0;
      tempoDecorrido = 0;
    }
  }
}
```

→ Aqui adicionamos 10s a variável “tempo” a cada vez que apertarmos o botão, e zeramos ela caso passe de 5x pressionadas.

```
void ligar(){
  if(millis() > tmpseg+500){
    tmpseg = millis();
    ligado = !ligado;
    tempo = 0;
    tempoDecorrido = 0;
  }
}
```

→ Segue o mesmo da função timer.

→ Aqui invertemos o valor da variável booleana “ligado” a cada vez que pressionamos o botão, e zeramos as outras variáveis.

```
void setup(){
  attachInterrupt(0,timer,RISING);
  attachInterrupt(1,ligar,FALLING);
}
```

→ Pino 2 – botão timer.
→ Pino 3 – botão ligar.

```
void loop(){

  if(ligado){
    int ldr = analogRead(4);
    int luminosidade = map(ldr,320,830,0,255);
    analogWrite(11,luminosidade);


 if(tempo > 0){
      if(tempoDecorrido < tempo){
        delay(100);
        tempoDecorrido+=100;
      }else{
        ligado = false;
      }
    }
  }else{
    analogWrite(11,0);
  }
}
```

→ Faz a leitura do LDR.
→ Ajusta o valor da conversão AD para o PWM.
→ Escreve o valor ajustado para a luminosidade do LED.

→ Nessa parte verificamos se o timer esta ativado ( tempo >0), caso isso ocorra:

Adicionamos 100 a variável tempoDecorrido para cada 100 milissegundos.
E caso “tempoDecorriso” seja maior que o tempo do timer desligamos o sistema.

→ Caso a variável “ligado” seja *false* desligamos o sistema.

# Oficina

## Introdução

Durante o minicurso fizemos vários projetos usando princípios básicos do Arduino. Agora na oficina faremos um projeto integrando alguns desses conceitos com uma rede de computadores.

Quando queremos criar uma solução para um problema, seja ela um projeto de automação, controle ou então um software de gerenciamento, devemos ter em mente o lugar e o processo que queremos melhorar, ou seja, a “planta”. Na maioria dos projetos a solução deve se adaptar a planta e não o contrario.

Para essa oficina nossa planta será uma sala com temperatura, luminosidade e acesso controlados. Para manter esses requisitos devemos ter um sistema onde:

- Para manter a temperatura controlada a porta do local deve permanecer fechada, e quando aberta deve ficar aberta pelo minimo de tempo possível, por isso devemos acender um sinal de aviso caso ela fique aberta por mais de 30 segundos.

- Ao entrar pela porta cada pessoa deve identificar-se. Sendo que o sistema deve guardar uma lista das pessoas que entraram ou saíram do local.

No local também teremos um painel indicando o status do sistema e um botão de alerta para caso de erros no sistema.

### Solução

Para a solução utilizaremos um Arduino para a identificação e o sensoriamento da porta conectado remotamente a um servidor de uma rede de computadores. Em um outro computador também ligado a rede teremos um outro Arduino que servira como um painel indicando o status do sistema. A Figura 1 mostra um esquemático do funcionamento do sistema.

O Arduino ligado ao servidor terá um sensor que ira verificar se a porta esta aberta, caso ela esteja por mais de 30 segundo ele enviara um sinal de aviso , além disso ele terá um sistema de identificação para as pessoas que entrarem na sala. Este Arduino estará conectado através de uma rede sem fio ao computador servidor.

O servidor terá um banco de dados para armazenar os parâmetros do sistema e um servidor

php para disponibilizar o acesso ao sistema por qualquer computador conectado a rede. Além disso devera estar rodando um programa que faça a comunicação entre o Arduino e o banco de dados. A Figura 2 ilustra como isso funciona.

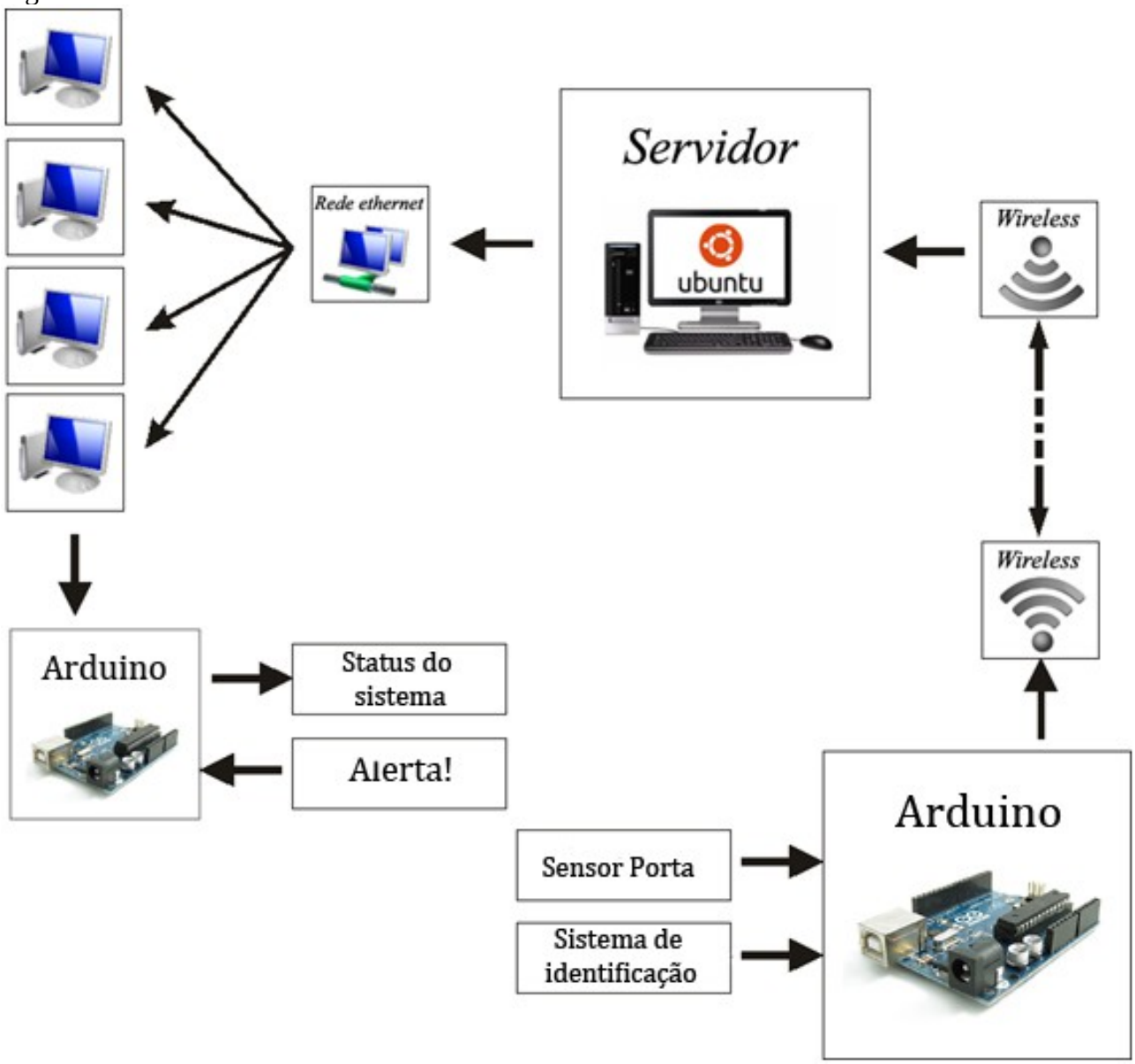



Figura 1 – Funcionamento do sistema.

Em um outro computador da rede estará conectado um outro Arduino, onde teremos alguns LEDs indicando o status do sistema e um botão de alerta para o caso de algum erro no sistema.

**Servidor**

Para o servidor web usamos o pacote LAMP que é um acrônimo para a combinação dos softwares livres:

- **L**inux
- **A**pache
- **M**ySQL
- **P**HP

Para instalar o LAMP no Ubuntu basta executar o comando *“sudo apt-get install lamp-*

*server^ ”* e o pacote é instalado diretamente do repositório. Outro software que é útil para trabalharmos com banco de dados MySQL é o “*MySQL Workbench”,* através dele podemos ver o banco de dados através de uma interface gráfica.

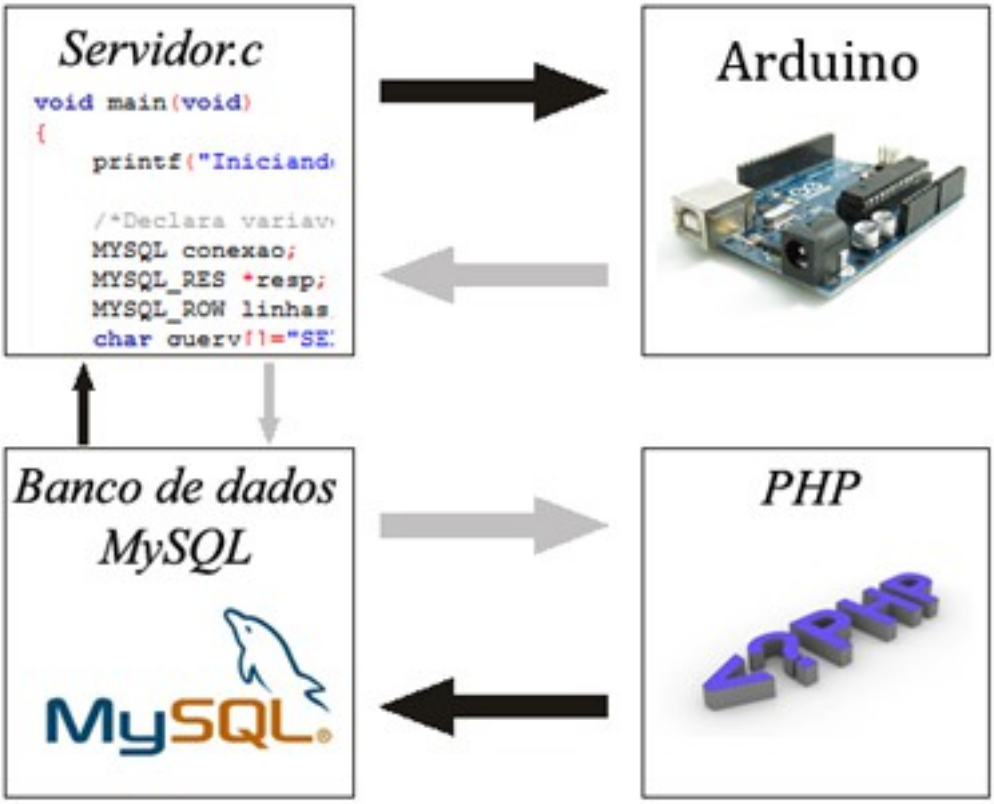



Figura 2 – Funcionamento do servidor.

***Banco de dados***

Com o LAMP instalado já temos o banco de dados MySQL funcionando, primeiro temos de criar um usuário e uma database. Também precisamos das tabelas onde colocaremos os parâmetros do sistema. Para isso executamos no terminal os comandos:

**$ mysql --user=NomeUsuario -p**
Enter password: **Senha**
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 38 to server version: 3.23.51-log
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> **connect NomeDatabase ;**
Connection id: 39
Current database: NomeDatabase

mysql> **create table NomeTabela (**
→ **`NomeColuna1` Tipo_de_variavel,**
→ **`NomeColuna2` Tipo_de_variavel**
→ **);**
Query OK, 0 rows affected (0.00 sec)

mysql> **exit**

| | |
|---|---|
| **Ex:**<br>mysql> **create table Status (**<br>→ **`Alerta` VARCHAR( 30 ) NOT NULL,**<br>→ **`Porta` VARCHAR( 30 ) NOT NULL ,**<br>→ **`ID` INT NOT NULL,**<br>→ **unique ( `ID` )**<br>→ **);**<br>Query OK, 0 rows affected (0.00 sec) | Nesse caso será criado uma coluna para o Alerta e outra para o status da Porta, além de uma extra para facilitar a manipulação de dados ID, que será um valor inteiro único, ou seja não pode ser repetido em outra linha. |

Devemos criar uma ou mais tabelas com colunas o suficiente para guardar a identificação da pessoa que entrara na sala e o status do sistema. Feito isso podemos receber os valores enviados dos dois Arduinos.

***PHP***

Nessa parte configuramos a pagina que será exibida no navegador de qualquer computador conectado a rede. Primeiro precisamos fazer com que essa pagina tenha conexão com o banco de dados, assim temos:
Arquivo Conecta.php:

```
<?
/* Este arquivo conecta um banco de dados MySQL - Servidor = localhost*/
$dbname="Nome_Database"; // Indique o nome do banco de dados que será aberto
$usuario="Nome_do_Usuario"; // Indique o nome do usuário que tem acesso
$password="Senha"; // Indique a senha do usuário
//1° passo - Conecta ao servidor MySQL
if(!($id = mysql_connect("localhost",$usuario,$password))) {
   echo "Não foi possível estabelecer uma conexão com o gerenciador MySQL. Favor Contactar o
Administrador.";
   exit;
}
//2° passo - Seleciona o Banco de Dados
if(!($con=mysql_select_db($dbname,$id))) {
   echo "Não foi possível estabelecer uma conexão com o gerenciador MySQL. Favor Contactar o
Administrador.";
   exit;
}
?>
```

O arquivo a seguir serve para executar um comando SQL no banco de dados:
Arquivo Executa.php:

```
<?php
/*
Esta função executa um comando SQL no banco de dados MySQL
$id - Ponteiro da Conexão
$sql - Cláusula SQL a executar
$erro - Especifica se a função exibe ou não(0=não, 1=sim)
$res - Resposta
*/
function mysqlexecuta($id,$sql,$erro = 1) {
```

```
    if(empty($sql) OR !($id))
       return 0; //Erro na conexão ou no comando SQL
   if (!($res = @mysql_query($sql,$id))) {
      if($erro)
        echo "Ocorreu um erro na execução do Comando SQL no banco de dados. Favor Contactar o
Administrador.";
      exit;
   }
    return $res;
 }
?>
```

E então a configuração da pagina que será exibida. Abaixo temos um bloco com estruturas que poderão ser usadas para criar a pagina que será exibida, esse código também devera ser salvo como php (por exemplo Oficina.php).

```
<html>
<body>

<?

    include "Conecta.php"; // Conecta ao banco de dados
    include "Executa.php"; // Executa a cláusula SQL

    /* Executa a consulta */
    $sql = "SELECT * FROM NomeTabela";
    $res = mysqlexecuta($id,$sql);

                                                .
                                                .
                                                .

    /* Exibição de imagem */
      echo "<tr>";
      echo "<td><img src=\"image.jpg\" alt=\"\" /><b>{$linha['sobrenome']}</b>
{$linha['nome']}</td>";
      echo "</tr>";
?>

                                                .
                                                .
                                                .



/* Exibição de dados em tabela */
<table width=15% cellpading=0 cellspacing=0>
<?
   echo "--------  Status do sistema  --------";
   while ($row = mysql_fetch_array($res)) {
?>
  <tr>
     <td><?echo "Titulo1: <br />\n";?></td>
     <td><?echo $row['Linha1'];?></td>
  </tr>
  <tr>
     <td><?echo "<br />\n Titulo2: <br />\n <br />\n";?></td>
     <td><?echo $row['Linha2'];?></td>
  </tr>

                                                .
```

```
                                                    .
                                                    .

<?
 }
?>

</table>
                                                   .
                                                   .
                                                   .

/* Botão */

<form action="" method="post">
     <input type="submit" value="NomeExibido" name="Nome1">
     <input type="submit" value="NomeExibido" name="Nome2">
</form>

/* Executa ação caso precionado algum botão*/
<?php

   if(isset($_POST["Nome1"])){
     $sql = "UPDATE Tabela1 SET Coluna1 = 'String' WHERE ID = 0;";
     $res = mysqlexecuta($id,$sql);
   }
   if(isset($_POST["Nome2"])){
     $sql = "UPDATE Tabela1 SET Coluna2 = 'String' WHERE ID = 0;";
     $res = mysqlexecuta($id,$sql);
   }

?>

                                                   .
                                                   .
                                                   .

</body>
</html>
```

Todos esses arquivos deves estar na pasta “/var/www”, com isso digitando no navegador “[ip do servidor]/Oficina” veremos nossa pagina com os parâmetros do sistema de qualquer computador da rede. Para acessar a pagina do computador servidor basta digitar no navegador “localhost/Oficina”.

***Conexão PHP ↔ Banco de dados ↔ Arduino***

Precisamos agora de uma aplicação que leia os dados enviados pelo Arduino e envie esses dados para o banco de dados. Para isso criamos o seguinte programa em C:
Arquivo Servidor.c:

```
/* Lista de inclusão geral */
#include <stdio.h>          // Biblioteca Standard input and output
#include <string.h>         // Biblioteca com funções de manipulação de strings
#include <unistd.h>         // Biblioteca para o delay : usleep(us);
#include <time.h>           // Biblioteca para obter o tempo do sistema

/* Lista de inclusão - MySQL*/
```

```c
#include <mysql/mysql.h>

/* Lista de inclusão - USB */
#include <fcntl.h>
#include <termios.h>

void IniciarPorta(int *fd, unsigned int baud)    // Configura e inicia a porta USB
{
    struct termios options;
    tcgetattr(*fd,&options);
    switch(baud)
    {
    case 9600:
        cfsetispeed(&options,B9600); /* Taxa de transferência em Bits/segundo, 9600 baud */
        cfsetospeed(&options,B9600); /* esse número deve ser igual                       */
        break;                       /* configuração do Serial.begin().                  */
    case 19200:
        cfsetispeed(&options,B19200);
        cfsetospeed(&options,B19200);
        break;
    case 38400:
        cfsetispeed(&options,B38400);
        cfsetospeed(&options,B38400);
        break;
    default:
        cfsetispeed(&options,B9600);
        cfsetospeed(&options,B9600);
        break;
    }
    options.c_cflag |= (CLOCAL | CREAD);
    options.c_cflag &= ~PARENB;
    options.c_cflag &= ~CSTOPB;

    options.c_cflag &= ~CSIZE;
    options.c_cflag |= CS8;
    tcsetattr(*fd,TCSANOW,&options);
}

void main(void)
{
    printf("Iniciando Servidor...\n");

    /*Declara variaves de comunicação com o banco de dados MySQL*/
    MYSQL conexao;
    MYSQL_RES *resp;
    MYSQL_ROW linhas;
    char query[]="SELECT * FROM NomeTabela;";

    /* Inicia conexão com o banco de dados*/
    mysql_init(&conexao);

    /* Conecta em localhost (Servidor Local) com usuario “Usuario” e senha “Senha” */
    /* na database “Database”                                                       */
    if ( mysql_real_connect(&conexao,"localhost","Usuario","Senha","Database",0,NULL,0) )
    {
        printf("Conectado com sucesso ao Banco de Dados MySQL!\n");
    }
    else           /* Caso haja falha na conexão avisa e fecha o programa*/
    {
        printf("Falha de conexao\n");
        printf("Erro %d : %s\n", mysql_errno(&conexao), mysql_error(&conexao));
        return;
```

```
    }

    /* Variaveis para a conexão sem fio */
    int fd, res=0;
    int alerta = 0;
    char buffer[25];


    /* Inicia comunicação USB em ttyUSB0 */
    fd = open("/dev/ttyUSB0", O_RDWR | O_NOCTTY | O_NDELAY);

    if(fd == -1)   // Checa se a comunicação foi estabelecida
    {
        perror("open_port: Não foi possivel abrir a porta ttyUSB0\n");
        return;
    }
    IniciarPorta(&fd,9600);         //Configura porta serial para 9600 bits/s, 8 bits ,n,1

    while(1){
        strcpy(buffer,"");

        usleep(100000);              // Da um atraso de tempo para que o comando inteiro
                                     // seja enviado do Arduino.

        res = read(fd,&buffer,20);   // Recebe e armazena em “buffer” o comando com 20 bytes.

                                                .
                                                .
                                                .

        write(fd, "x", 1);          // Envia o caractere 'x' para o Arduino.

                                                .
                                                .
                                                .

        /* Cria uma string para enviar um comando para o banco de dados */
        sprintf(squery,"INSERT INTO `NomeTabela` (`Coluna1`, `Coluna2`) VALUES ('%s',
'%s');", String1, String2 );
        /* Insere na tabela “NomeTabela” String1 na Coluna1 e String2 na Coluna2 */
        mysql_query(&conexao,squery);



                                                .
                                                .
                                                .

         /* lê o primeiro campo da tabela, caso seja “On”, envia um sinal de alerta*/
        /* caso seja “Off” envia sinal de alerta desligado                          */
        mysql_query(&conexao,query);
        resp = mysql_store_result(&conexao);     //recebe a consulta
        if (resp)                                //se houver consulta
        {

            /* Enquanto retornar registros, compara os valores */
            /* e envia os comandos 'a' para ligar alerta e 'n' para desligar*/


            while ((linhas=mysql_fetch_row(resp)) != NULL)
            {
                if(!strcmp(linhas[0],"On") && !alerta )
```

```
                {
                    write(fd, "a", 1);     // comando liga alerta
                    printf("\n----Alerta Ativado----\n\n");
                    alerta = 1;
                }
                if(!strcmp(linhas[0],"Off") && alerta )
                {
                    write(fd, "n", 1);      // comando desliga alerta
                    printf("\n----Alerta Desativado----\n\n");
                    alerta = 0;
                }
            }
        }

    }
    mysql_close(&conexao);  // termina a conexão com o MySQL
}

/*****************************************************************************/
/********************** Codigo para compilação ******************************/
/*                                                                           */
/* gcc -o Servidor $(mysql_config --cflags) Servidor.c $(mysql_config --libs) */
/*                                                                           */
/*****************************************************************************/
```

### *Arduinos*

Os dois Arduinos devem ser programados de forma que se comuniquem corretamente com o computador, ou seja, caso o Arduino envie “PrtA” caso a porta seja aberta, o servidor deve entender o comando para salvar no banco de dados.

O segundo Arduino também precisara de algum programa auxiliar para poder ver o status do sistema, isso também é um problema para resolvermos na oficina.

# Proposta de implementação dos exercícios

**Projeto 3:**

| | |
|---|---|
| `#define BOTAO 2`<br>`#define LDR 4`<br>`#define LED 11` | → Definição dos pinos onde estão conectados os componentes. Poderíamos também fazer isso por meio de variáveis: `int LDR = 4;` |
| `boolean ligado= false;` | → Variável booleana(Verdadeiro ou Falso) que guarda o status do sistema(ligado ou desligado). |
| `void setup(){` | → Função de inicialização. |
| `  Serial.begin(9600);` | → Inicia comunicação serial com o computador para os ajustes. |
| `  pinMode(BOTAO,INPUT);`<br>`}` | → Define o pino que esta conectado o botão como saída digital. |
| `void loop(){` | → Função loop. |
| `if(ligado){` | → Checa se o sistema esta ligado. |
| `    int x = analogRead(LDR);` | → Caso esteja, lê o valor do LDR. |
| `    Serial.println(x);` | → Envia o valor lido no LDR para o computador. Usado para os ajustes. |
| `    int y = map(x,150,750,255,0);` | → Ajusta os valores lidos ( 150 para pouca luz e 750 para muita luz) para os valores a serem escritos ( 0 para LED totalmente desligado e 255 para LED totalmente ligado). A lógica é invertida( 255-0 ao invés de 0-255) porque queremos que o LED diminua sua luminosidade quanto mais luz incide no |

```
   analogWrite(LED,y);

   //delay(500);

   if(digitalRead(BOTAO)){
     ligado = false;

     analogWrite(LED,0);

     delay(1000);
    }
  }


  else{
    if(digitalRead(BOTAO)){
      ligado = true;
      delay(1000);
    }
  }
}
```

LDR.

→ Escreve o valor ajustado no pino do LED.

→ Espera 500ms, usado para ajustes.

→ Caso o botão for apertado quando o sistema estiver ligado, status vai para false = desligado.

→ Desliga o LED.

→ Espera 1 segundo – como a leitura é feita milhares de vezes por segundo esse “delay” evita que o botão seja lido mais de uma vez por segundo.

→ Aqui caso o botão não esteja pressionado, esperamos ele ser pressionado, então mudamos o status do sistema para true = ligado.

**Projeto 4:**

```
int receptor = 2;
int claro = 550;
int escuro = 190;
String Cor = "",Verificar = "";

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

void loop(){
  int x = analogRead(receptor);
  if(x > claro)Cor = "Objeto Claro";
  if(x < escuro)Cor = "Objeto Escuro";
  if(x < claro && x > escuro)
     Cor = "Aproxime o Objeto";

  if(Cor != Verificar){
    Serial.println(Cor);
    Verificar = Cor;
  }
  delay(500);
}
```

→ Ligamos o receptor IR no pino 2 (entrada analógica).
→ A leitura do receptor foi de 550+ para objetos claros,
→ E foi de 190- para objetos escuros.
→ Inicializamos as strings para enviar serialmente para o PC.

→ Inicializamos a comunicação serial na função setup.

→ Lemos a entrada analógica ligada ao receptor.
→ Caso a leitura seja 550+ a string guardara “Objeto Claro”
→ Caso seja 190- guardara “Objeto Escuro”
→ Caso o valor esteja entre 190 e 550 pediremos para aproximar um objeto.

→ Aqui verificamos se a string a ser enviada é a mesma que a anterior, caso sim evitamos a repetição da cor do objeto, caso não enviamos serialmente a cor do objeto.

→ Esperamos um tempo para prevenir ruídos na leitura.

## *Bibliografia*

**[1] Main Page do site do Arduino. *http://www.arduino.cc/***

***http://vimeo.com/31389230*** [Documentário Arduino]

JORDÃO, Fabio “Arduino: a plataforma open source que vai automatizar a sua vida”; Local: *http://www.tecmundo.com.br/android/10098-arduino-a-plataforma-open-source-que-vai-automatizar-a-sua-vida.htm#ixzz2cSM17nfy* ; (05/2011); acessado em 08/2013

SADIKU, Matthew N. O. “Fundamentos de circuito elétricos”. Local: Mcgraw Hill Brasil técnicos 3ª Ed.; 2008.

IRWIN, J.D.; "Análise de Circuitos em Engenharia". Makron Books do Brasil Editora LTDA; 4ª Ed.; 2000.]

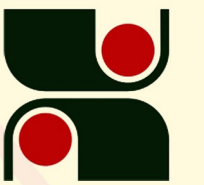
## *Material Extra*

***http://fritzing.org:*** Software para desenho de circuitos usados neste material.

[a] *http://commons.wikimedia.org/wiki/Category:Arduino*

[b] http://commons.wikimedia.org/wiki/Category:Atmel_AVR

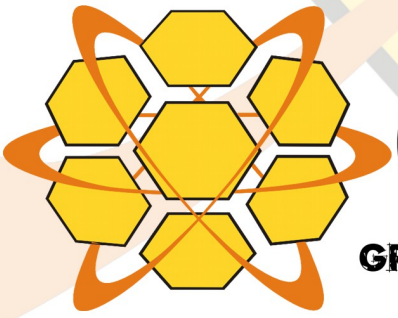
[c] http://nerduino.blogspot.com.br/2013/04/como-usar-protoboard-eletronica-basica.html



UDESC
Joinville

open source
hardware

ARDUINO



COLMÉIA

GRUPO DE PESQUISA EM SOFTWARE E HARDWARE LIVRE