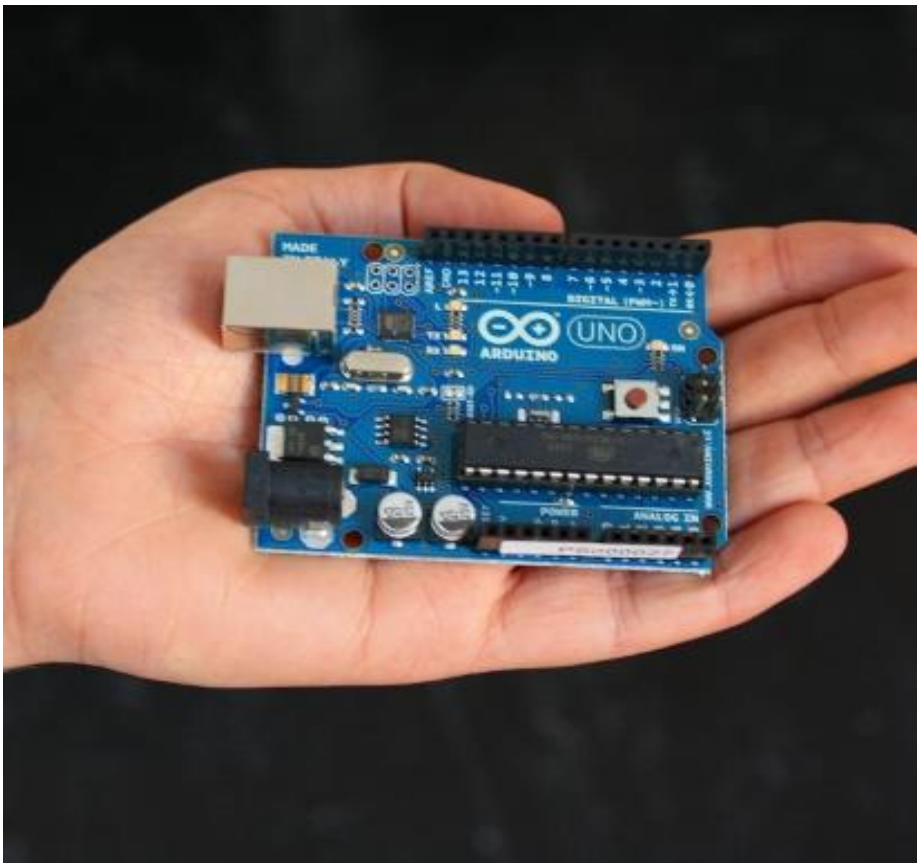
ARDUINO E SEUS CONCEITOS BÁSICOS



AUTORES:

Kaio Setsuo Sato Ramos

João Pedro Botazzo

Jean César Hilário

Lucas Pin Serion

CARO LEITOR E ALUNO...

Este material didático auxiliará você, aluno a uma melhor compreensão sobre o componente ARDUINO, seus conceitos e sua utilização de forma correta, pois leva-se em conta a pouca experiência do indivíduo quanto ao manuseio da placa Arduino.

AGRADECIMENTOS

- Professor João Luiz Trevelim

- Professora Anita Teixeira de Moura

- Professor José Carlos Martins

- Professor Cléber Rodrigo Torres

"Que ninguém se engane, só se consegue a simplicidade através de muito trabalho."

- Clarice Lispector

BONS ESTUDOS!

ÍNDICE

INTRODUÇÃO

- **Quando surgiu o Arduino?**

O Arduino surgiu em 2005, na Itália, com um professor chamado Massimo Banzi, que queria ensinar eletrônica e programação de computadores a seus alunos de design, para que eles usassem em seus projetos de arte, interatividade e robótica. Porém, ensinar eletrônica e programação para pessoas que não são da área não era uma tarefa simples, e outra dificuldade era a inexistência de placas poderosas e baratas no mercado.

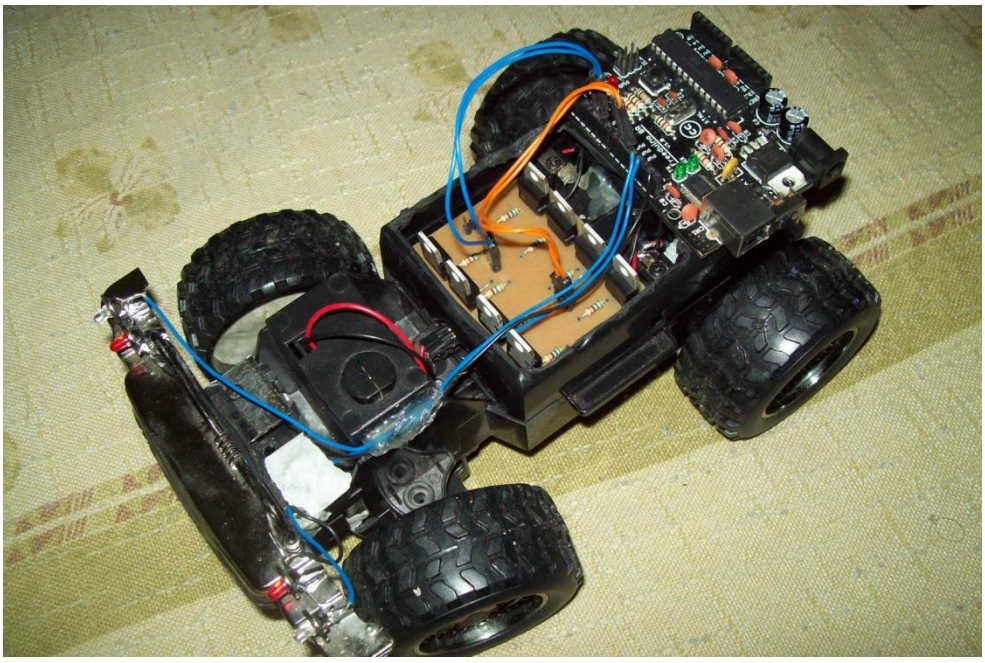
Foi pensando nisso que Massimo e David Cuartielles decidiram criar sua placa própria, com a ajuda do aluno de Massimo, David Mellis, que ficou responsável por criar a linguagem de programação do Arduino. Várias pessoas conseguiram utilizar o Arduino e fazer coisas incríveis, surgindo assim essa febre mundial da eletrônica.

- O que é o Arduino?

O Arduino é um projeto totalmente aberto de protótipos de eletrônica baseados numa plataforma de hardware e software flexível e de fácil utilização. É destinado a artistas, designers, hobbyistas e qualquer tipo de pessoa interessada em criar objetos ou ambientes interativos. É um projeto que trata de software e hardware, e tem como objetivo principal fornecer uma plataforma para prototipação de projetos, 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 com os sensores, motores e outros dispositivos eletrônicos.

→ Figura 1 – Caixa do Arduino UNO

Devido a sua estrutura, pode receber informações de uma grande variedade de sensores, e pode interagir com o ambiente, controlando luzes, motores, entre outros.



→ Figura 2 – Utilização de sensores e motores no Arduino para o desenvolvimento de uma miniatura de carro remotamente controlado. - http://goo.gl/BkkOrL

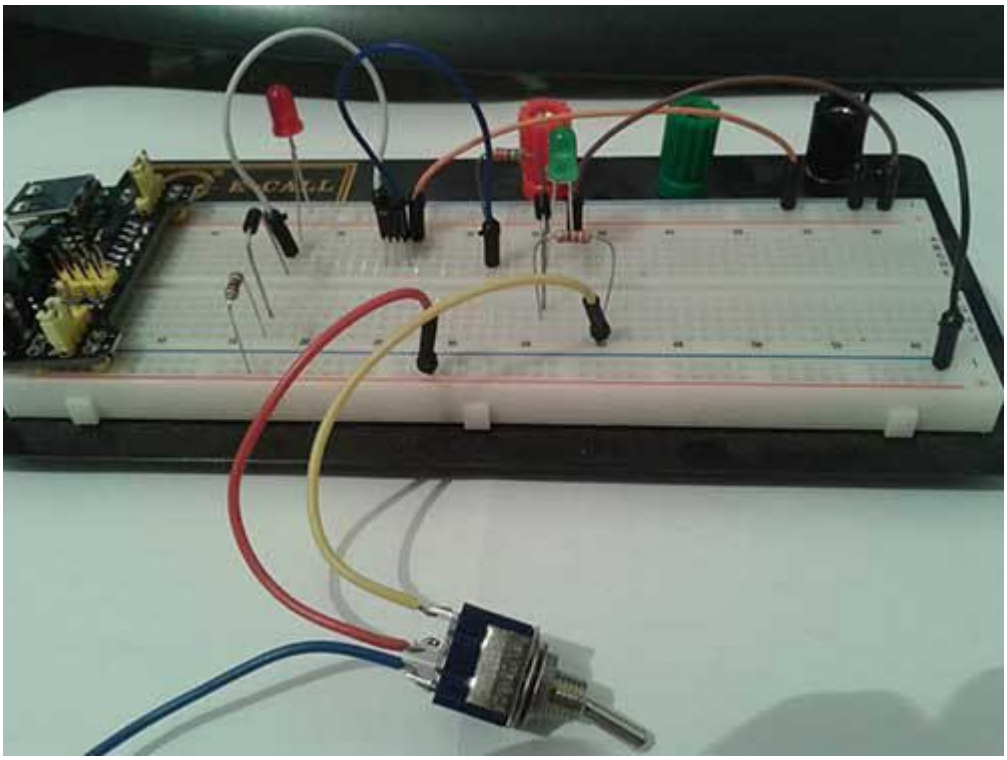
São utilizados os microcontroladores da ATMEL, cuja programação se baseia na linguagem Arduino (feita a partir do C/C++), e por isso também aceita programação direta em C/C++. O software para a programação pode ser encontrado no link: http://arduino.cc/en/Main/Software.

Normalmente é necessário construir os circuitos para entrada e saída do Arduino, o que permite uma grande variedade de soluções para o mesmo problema. Na internet é possível observar diversos projetos, com diferentes finalidades, o que ajuda muito no aprendizado do aluno. E por conta disto, o próprio site do Arduino (www.arduino.cc).

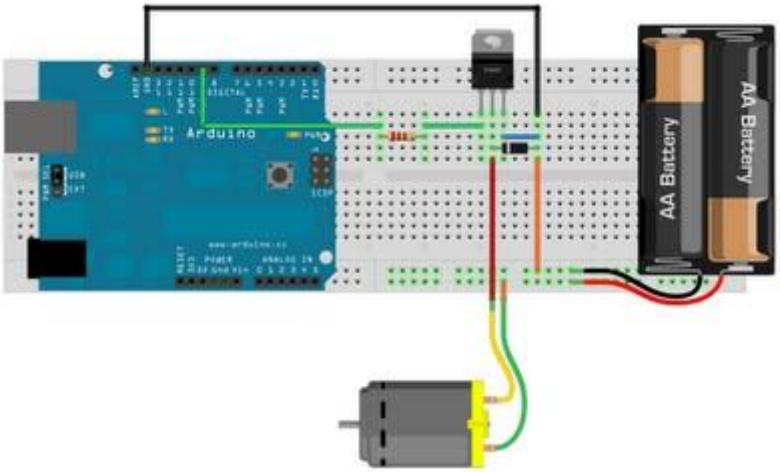
Há uma grande diversidade nos projetos que podem ser executados, podendo ser utilizado somente o Arduino, ou estabelecer uma comunicação com outros equipamentos, como o computador.

EXEMPLOS DE APLICAÇÃO

- LED – luz de indicação ON/OFF



- Controle de Motor DC – ON/OFF

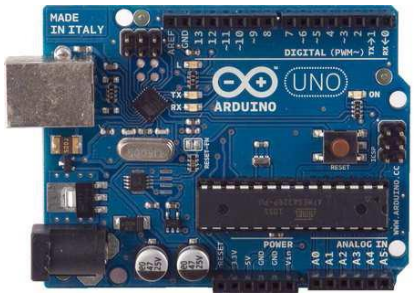


http://bit.ly/1m542zk

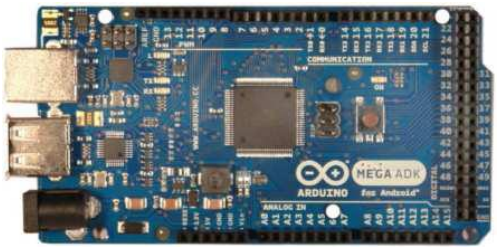
- Programação para o circuito acima:

```
int motor= 9;

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

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

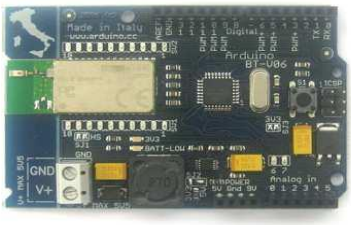
- **Principais Modelos de Arduino:**



Arduino UNO



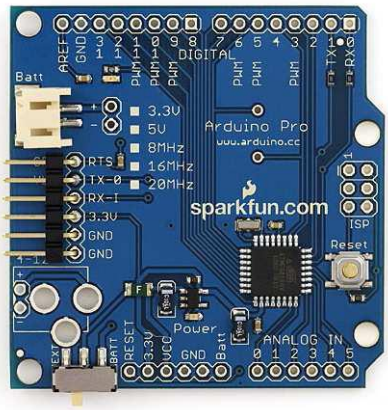
Arduino ADK



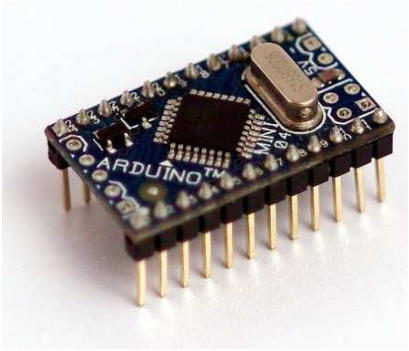
Arduino BT400

Arduino MEGA 2560

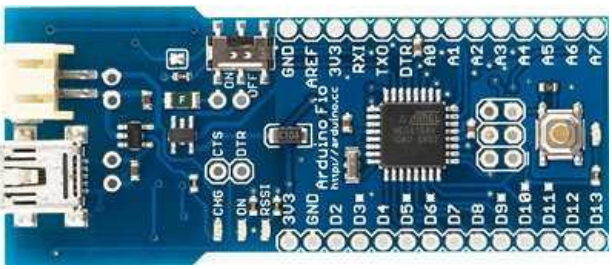
Arduino Nano



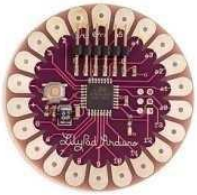
Arduino PRO



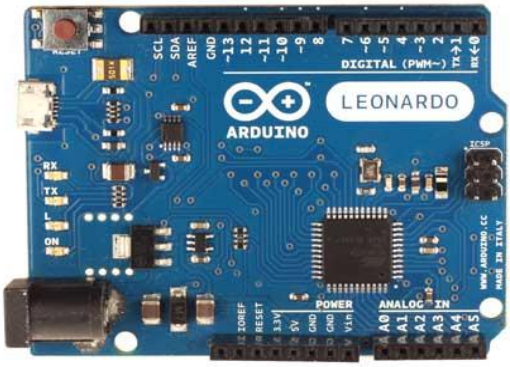
Arduino Mini



Arduino Fio



Arduino LilyPad



Arduino Leonardo

- **HARDWARE:**

→ Características dos principais modelos:

Arduino UNO

(Existem várias versões de Arduino UNO como UNO SMD, UNO R2 e o último UNO R3)

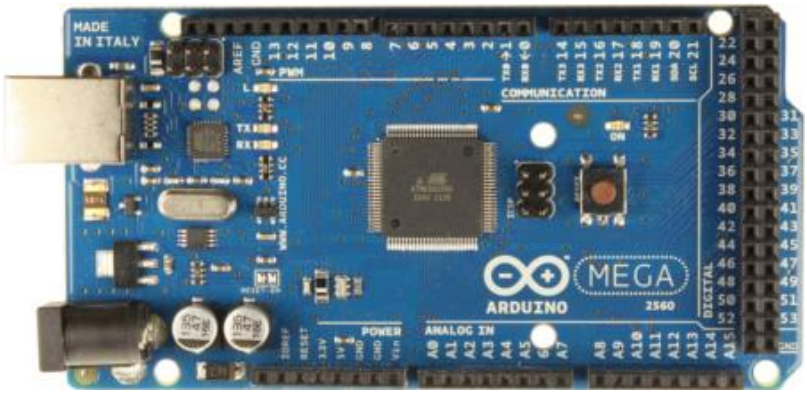
Microcontrolador	ATmega328
Tensão de operação	5V
Tensão de entrada (recomendada)	7-12V
Tensão de entrada (limites)	6-20V
Pinos de I/O Digitais	14 (6 deles com saída PWM)
Pinos Analógicos	6
Corrente CC por I/O Pino	40 mA
Corrente do Pino 3.3V	50 mA
Memória Flash	32 KB (ATmega328) 0.5 KB usado pelo bootloader
SRAM	2 KB (ATmega328)

EEPROM	1 KB (ATmega328)
Velocidade do Clock	16 MHz

Este é o Arduino recomendado para iniciantes, pois comporta desde pequenos projetos, até alguns mais ousados, utilizando diversas portas em seu protótipo.

Arduino **MEGA**

(Existem outras versões também como MEGA UNO, MEGA R3)

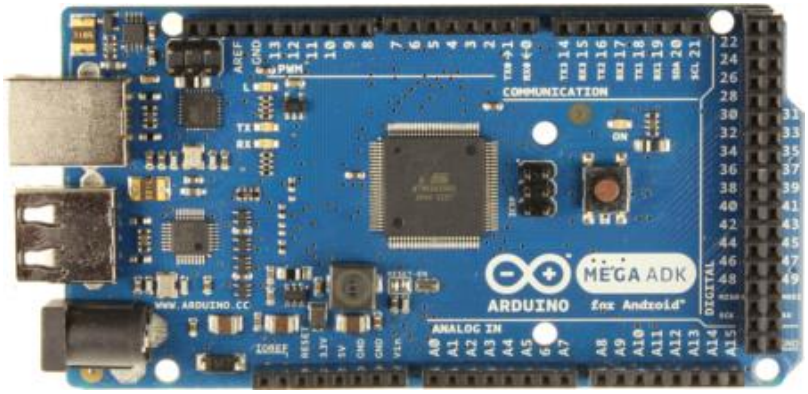


Microcontrolador	ATmega2560
Tensão de operação	5V
Tensão de entrada (recomendada)	7-12V
Tensão de entrada (limites)	6-20V
Pinos de I/O Digitais	54 (14 deles com saída PWM)
Pinos Analógicos	16
Corrente CC por I/O Pino	40 mA
Corrente do Pino 3.3V	50 mA
Memória Flash	256 KB (ATmega2560) 8 KB usado

	pelo bootloader
SRAM	8 KB (ATmega2560)
EEPROM	4 KB (ATmega2560)
Velocidade do Clock	16 MHz

Este modelo de Arduino é recomendado para a elaboração de grandes projetos, devido a sua maior capacidade de processamento e a quantidade extra de portas. Pode ser utilizado para a automação em linhas de montagem.

Arduino ADK

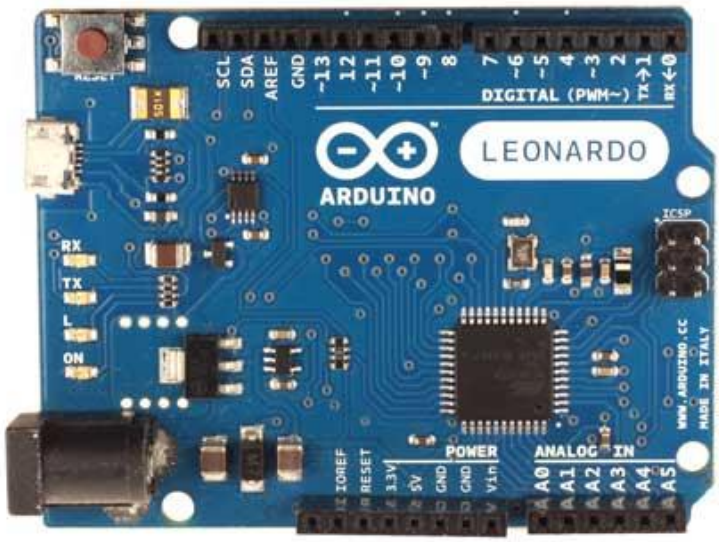


Microcontrolador	ATmega2560
Tensão de operação	5V
Tensão de entrada (recomendada)	7-12V
Tensão de entrada (limites)	6-20V
Pinos de I/O Digitais	54 (14 deles com saída PWM)
Pinos Analógicos	16
Corrente CC por I/O Pino	40 mA
Corrente do Pino 3.3V	50 mA
Memória Flash	256 KB (ATmega2560) 8 KB usado

	pelo bootloader
SRAM	8 KB (ATmega2560)
EEPROM	4 KB (ATmega2560)
Velocidade do Clock	16 MHz

Como podemos ver a única diferença do ADK para o Arduino MEGA em si é que o ADK possui uma porta USB que serve para ligar em aparelhos que possua o sistema operacional Android. Essa placa serve para principalmente para execução de projetos que tenham interação com o S.O. Android.

Arduino Leonardo (Essa placa é a mais recente, foi criada a pouco tempo)



Microcontrolador	ATmega32U4
Tensão de operação	5V
Tensão de entrada (recomendada)	7-12V
Tensão de entrada (limites)	6-20V
Pinos de I/O Digitais	20 (7 deles com saída PWM)

Pinos Analógicos	12
Corrente CC por I/O Pino	40 mA
Corrente do Pino 3.3V	50 mA
Memória Flash	32 KB (ATmega32U4) 4 KB usado pelo bootloader
SRAM	2,5 KB (ATmega32U4)
EEPROM	1 KB (ATmega2560)
Velocidade do Clock	16 MHz

Essa placa é recente, serve para quem já está acostumado com o Arduino, pois na programação foi criado mais funções com ele, como por exemplo, usar a placa como teclado ou mouse. Outra coisa que difere das outras placas é que o processamento dele é mais rápido.

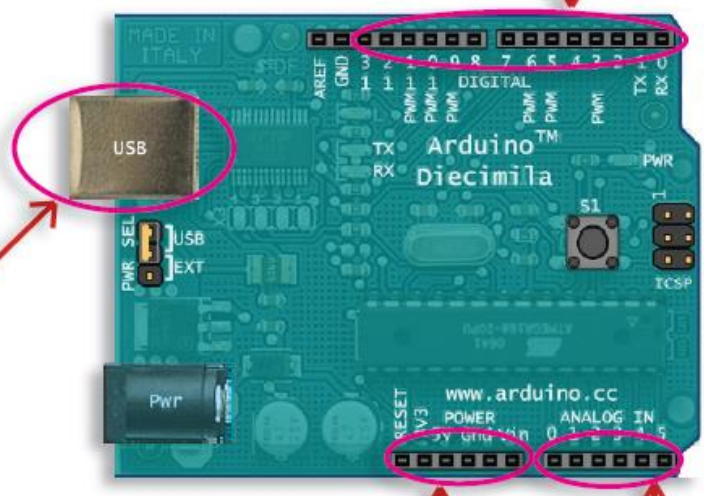
Existem outros tipos como Fio, Nano, Pro Nano, LilyPad. Mas esses tipos de Arduino são muito pequenos, perfeitos para projetos pequenos e que gostariam que fossem discretos.

Para mais informações, consulte o site: http://arduino.cc/en/Main/Products.

→ PINOS E FUNÇÕES

Pinos Digitais
usados para deteção ou transmissão de controles digitais

Funções em C:
pinMode( )
digitalRead( )
digitalWrite( )
analogWrite( )
attachInterrupt( )
pulseIn( )



Porta USB - usada para comunicação serial com um computador

Funções em C:
begin( )
end( )
available( )
read( )
print( )
println( )

Pinos de Alimentação
usados para alimentação de circuitos externos e reset do Arduino

Pinos analógicos
usados para leitura de sinais de sensores

Função em C:
analogRead( )

• AS PORTAS DE E/S DO ARDUINO E SUAS FUNÇÕES

OS PINOS DIGITAIS São 14 pinos marcados com o nome **DIGITAL** logo abaixo de duas barras de 8 pinos. São numerados de 0 a 13 da direita para a esquerda e podem ser configurados pela função **pinMode( )** para detetarem ou transmitirem níveis lógicos digitais (verdadeiro/falso, 1/0 ou HIGH/LOW).

Pinos AREF e GND: o pino AREF é a entrada de tensão de referência para o conversor A/D do Arduino; o pino GND é o terra, comum a todos os outros pinos.

Pinos 3, 5 e 6 e 9 a11 (PWM): 6 pinos dos 14 pinos digitais podem ser usados para gerar sinais analógicos com a função **analogWrite( )** utilizando a técnica de Modulação por Largura de Pulso (PWM).

Pinos 0 e 1: os dois primeiros pinos digitais são conectados a USART do microcontrolador do Arduino para comunicação serial com um computador.

Pinos 2 e 3: pinos que chamam uma ISR (Interrupt Service Routine) para tratar uma interrupção com a função **attachInterrupt( )** nesses pinos.

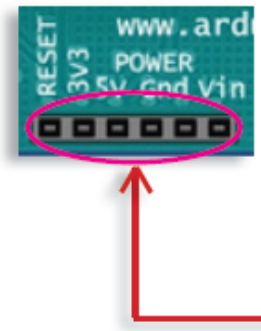
• **OS PINOS ANALÓGICOS** São 6 pinos em uma só barra com o nome **ANALOG IN**, localizada no lado oposto às barras dos pinos digitais. São numerados de 0 a 5, agora da esquerda para a direita. Esses pinos são usados para leitura de sinais analógicos de sensores conectados ao Arduino, e podem ser de quaisquer valores entre zero a 5 volts. Os pinos de entradas analógicas não precisam ser previamente configurados com a função **pinMode( )**.

Pinos 0 a 5: esses 6 pinos aceitam tensões entre zero e 5 volts CC que vão ao conversor A/D de 10 bits no microcontrolador do Arduino. O pino AREF, numa das barras de pinos digitais, é a entrada de tensão de referência para esse conversor.

- A PORTA SERIAL DO ARDUINO E SUAS FUNÇÕES EM C

O conector USB: É por meio desse conector USB fêmea do tipo A que o Arduino se comunica atraves de um cabo a um computador ou a outros dispositivos que tenham tambem uma interface USB. É tambem por esse conector que o Arduino recebe 5 volts diretamente da fonte de alimentação do computador.

- OS PINOS DE ALIMENTAÇÃO Ficam na barra com 6 pinos, marcada como **POWER**, localizada ao lado dos pinos analógicos. O primeiro pino dessa barra, **RESET**, quando forçado ao potencial de terra serve para resetar o Arduino. Do outro lado, **Vin** é um pino que tambem pode servir para alimentar o Arduino se nele for aplicada uma tensão entre 9 e 15 volts.



Pinos 3V3, 5V e *Gnd*: dos 6 pinos dessa barra somente os quatro do meio servem para alimentar um circuito externo conectado ao Arduino: o pino de 5V e o terra (os dois pinos Gnd entre 5V e Vin); e o pino 3V3 que disponibiliza essa tensão com uma corrente máxima de 50mA.

SOFTWARE

➔ Linguagem de Programação:

Um programa possui um corpo, uma função principal, onde o programa será rodado.

Em Arduino, é obrigatório que o programa tenha a função void setup() e a função void loop(). Caso seu programa não tenha estas funções, ele não será capaz de identificar a linguagem e não realizará a compilação.

A função void setup() funciona como uma “função de inicialização”, o que estará contido nela rodará apenas uma vez no programa. Nela, deve conter as informações principais para iniciar o seu programa, que estará contido dentro da função void loop(). Em void loop() deverá conter tudo o que você quiser que o programa faça.

ENTRADA E SAÍDA DE DADOS I

Quando desejarmos introduzir dados ou ler dados recebidos de uma porta, devemos usar alguns comandos específicos.

➔ pinMode(pino,MODO)

No Arduino, como lidamos com entrada e saída de dados, o pinMode configura um pino específico da placa como entrada ou saída. O pino é um número inteiro e ele é identificado na própria placa. O MODO ele pode ser de entrada (INPUT) ou saída (OUTPUT) de dados.

Observação: Quando se tratar de uma leitura de dados analógicos, no caso os pinos A0 ao A5, não é necessário realizar a declaração destes pinos em void setup(), porém, os pinos analógicos, também podem ser usados como pinos digitais, assim, quando estiver se referindo ao uso destes pinos como digitais, você deverá identificar no programa na função de inicialização como pino 14 (no caso do A0) até o 19 (no caso do A5).

PORTAS DIGITAIS

➔ digitalRead(pino) Este comando é responsável pela leitura de dados nas portas digitais disponíveis. Como falamos de leitura de dados digitais, ela será feita como HIGH ou LOW.

➔ digitalWrite(pino, VALOR) Se você configurar em void setup() um pino como OUTPUT (saída de dados), você poderá definir se ela será alta (HIGH) ou baixa (LOW), assim, você poderá comandar qualquer aplicação que se encontrará conectado em tal pino.

PORTAS ANALÓGICAS

Se você ler dados em uma entrada analógica, você obterá como retorno um valor entre 0 e 1023, pois, o Arduino estudado possui um conversor analógico digital de 10 bits, então, 2^10 = 1024. 1023 é o valor máximo, assim, será colocado na entrada do pino o valor de tensão máximo de sua tensão de referência (AREF – Analog Reference).

Mudar a tensão de referência pode ser uma boa idéia, porém, alterando-a em hardware, deve-se também alterar o software. A tensão de referência padrão dp Arduino é de 5V, mas, também há uma tensão de referência interna no valor de 1,1V (no ATMMega168), e uma tensão de referência externa, definida pelo técnico.

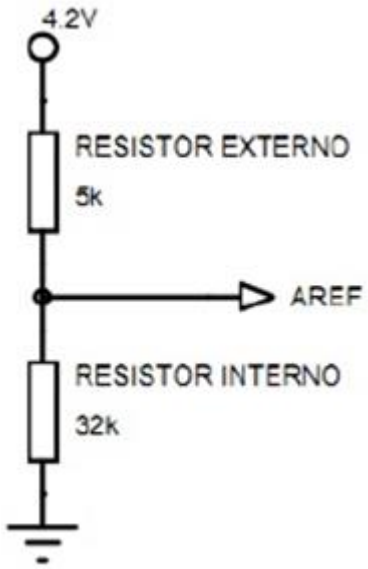
Se por exemplo, for usada a tensão padrão (5V), dizemos que:

5V/1024= 0,0048v=5mV

Isso significa que a cada bit será adicionado 5mV à entrada e que o Arduino apenas conseguirá detectar tensões superiores à 5mV. Alterando o valor de referência é possível obter uma maior sensibilidade, no caso de sensores que não necessitam de uma tensão muito alta e uma grande precisão.

OBSERVAÇÃO:

O recomendado é caso for usada uma tensão de referência externa, que seja colocado um resistor de 5kΩ, assim, caso possua alguma configuração errada no software e o microcontrolador não conseguir entender, isso previne que ocorra lagum dano no equipamento, porém, internamente há uma resistência no valor de 32kΩ, isso significa que você terá um divisor de tensão de referência será diferente da tensão aplicada antes do resistor de 5kΩ.



Exemplo: Se for aplicada uma tensão de 4,2V na entrada do equipamento, a tensão de refrência será:

$$AREF = \frac{4,2V \; x \; 32kOms}{32kOms + 5kOms} = 3,63V$$

Se for definido o uso de uma tensão de referência externa, as portas disponíveis com valores de 5V e 3,3V serão fechadas e não funcionará mais.

AnalogReference(TIPO) em software, este comando é usado para definir qual é a tensão de referência usado em hardware. Ou seja, se o TIPO será padrão (DEFAULT), interna (INTERNAL) ou externa (EXTERNAL).

AnalogRead(pino) este comando é usado para utilizar a leitura de alguma entrada analógica. Uma entrada analógica não precisa ser definida na função de inicialização como de entrada ou saída.

O uso desta função retorna um valor inteiro, que como já foi informado, será de 0 a 1023 unidades, sendo equivalente um valor aproximado de 5mV por unidade. Para realizar a leitura de uma entrada digital é necessário um tempo aproximado de 100us.

AnalogWrite(pino, valor) Possibilita usar os pinos PWM (Pulse Width Modulation) do Arduino. Esta funcionalidade serve tanto para variar o valor de um LED quanto para variar a velocidade de um motor. O sinal PWM se mantém até ser modificado por outra instrução. A frequência de um sinal PWm é aproximadamente 490Hz. Os pinos PWM no Arduino Uno são 3, 5, 6, 9, 10 e 11. O valor pode alterar entre 0 (sempre desligado) vê 255 (apresenta um sinal constante). Haverá mais explicações sobre sinais PWM posteriormente.

TEMPO

Delay(tempo em ms) Este comando possibilita uma pausa do programa em excução em uma quantidade de milissegundos específica que é nomeada no campo tempo em ms.

delayMicroseconds (tempo em us) Este comando possibilita uma pausa do programa em execução em uma quantidade de microssegundos específica que é nomeada no campo tempo em us.

Millis Este comando possibilita o retorno da quantidade de tempo que passou, em milissegundos, desde que o programa atual começou a ser executado. Para usar este comando, é necessário o uso da variável unsigned long.

```
/*
Entrada Analógica
Demonstra a leitura de um sensor analógico (A0) e a
emissão de luz de um LED (13).
O tempo que o LED permanecerá acesso dependerá do valor
obtido do sensor.
Este exemplo de código é de domínio público.
Modificado por Micael Bronzatti Gaier
*/
// #include <SoftwareSerial.h>
#define sensor A0 // seleciona o pino de entrada do
//potenciômetro e o identifica como sensor
int led = 13;       // seleciona o pino para o LED
int valSensor;  // variável que armazena o valor do sensor

void setup() {
  Serial.begin(9600);
  pinMode(led, OUTPUT);
}
```

```
void loop() {
valSensor = analogRead(sensor); // lê sensor e define
// valor como variável global
Serial.println(valSensor); // envia valor do sensor  //
para porta serial
piscarLED(); // função secundária
 }
void piscarLED(){
digitalWrite(led, HIGH); // ligar o LED
delay(valSensor);  //pausa de acordo com o valor da
//variável
digitalWrite(led, LOW); // apagar o LED
delay(valSensor);
    }
```

BIBLIOTECAS

Você, como programador, deve saber que devemos simplificar ao máximo nossos códigos visando economizar espaço na memória do programa. Para isso quando você obtém um equipamento, sensor, etc., o fabricante muitas vezes fornece uma biblioteca para facilitar a programação, ou seja, invés de você realizar uma longa e extensa programação, você utiliza a biblioteca e simplifica o código.

Quando você obtém uma biblioteca e a armazena na pasta de bibliotecas do Arduino, ao abrir um programa e clicar no menu em **Sketch > Import Library**, você poderá escolher a biblioteca que desejar. Quando escolher, aparecerá no seu programa algo como no exemplo acima, porém, sem ser comentado.

#include <NomeDaBiblioteca.h>

Quando adicionar uma biblioteca ao seu programa, é necessário verificar as instruções corretas para realizar a progração.

#include e #define

Quando você usa #include, você está dizendo ao programa que ele deve incluir o arquivo-cabeçalho ao programa.

O uso da ferramenta #define é interessante para ser usado com o Arduino. Observando p exemplo 2, você percebe que logo após a declaração da ferramenta, é colocado um nome em geral. Após ele foi usado o local onde está o sensor no Arduino, o pino A0. Com isso, você percebe que em todo o programa quando quiser se referir ao pino A0, se usa a palavra sensor. Assim, podemos definir:

#define nome Variavel dados (OBS: não é usao ‘;’ no final)

Com isso dizemos que podemos usar um nome qualquer, invés de colocas os dados diretamente no programa. Isso facilita a programação, pois ela ficará mais legível e caso seja necessário alterar o valor dos dados, será

necessário modificar apenas uma vez. A ferramenta #define, é uma ferramenta local, ou seja, você a usa apenas para programar, quando você for compilar o programa, o nome da variável será substituído automaticamente pelo dado contido, assim, você acaba economizando espaço na memória do microcontrolador.

VARIÁVEIS E MODIFICADORES

VARIÁVEIS

Os nomes das variáveis apenas devem obedecer algumas regras: elas podem começar com letras ou sublinhado (_) e não podem ter nome idêntico a alguma das palavras reservadas pelo programa ou de alguma biblioteca.

As variáveis devem ser declaradas antes de serem usadas. Observe:

tipo_de_variável *lista_de_variáveis*

Ex: **int** *ledPin, potenciômetro*

CLASSE DE VARIÁVEIS

Um código que se encontra dentro de outra função, entre { e }, se encontra isolado de todo o resto do código e os dados contidos lá apenas podem ser armazenados através de variáveis. Existem três classes de variáveis: locais, globais e estáticas.

VARIÁVEIS LOCAIS

Quando uma variável é declarada dentro de uma função específica, ela é denominada variável local. Estas variáveis apenas existem enquanto o bloco onde está armazenada estiver sendo executado. A partir do momento em que o programa voltar à função principal, esta variável deixará de existir.

VARIÁVEIS GLOBAIS

Uma variável global é aquela variável que é conhecida por todo o programa, ou seja, independente da função que estiver sendo executada ela será reconhecida e rodará normalmente. Uma variável global ela é declarada antes mesmo da função void **setup**().

VARIÁVEIS ESTÁTICAS

Funcionam de forma parecida com as variáveis globais conservando o valor durante a execução de outras funções, porém, só são reconhecidas dentro da função onde é declarada. Como exemplo, podemos dizer que uma variável estática é uma variável declarada dentro da lista de parâmetros de uma função.

TIPOS DE DADOS E MODIFICADORES

Em linguagem C, e também em Arduino, existe 5 tipos de dados: **char, int, float, void, double**.

void é usado apenas na declaração de funções, indica que a função chamada não retornará nenhum valor para a função que a chamou.

*Exemplo: void **loop**()*

boolean – *variáveis booleanas* estas variáveis memorizam um de dois valores: verdadeiro (true) ou falso (false).

Exemplo: condicao = false;

char é um tipo de dado que dedica 1 byte de memória para armazenar o valor de um caractere.

Exemplo: palavra = 'Arduino';

byte byte memoriza um número de 8-bits, de 0 a 255. Esse tipo de dados não memoria número negativos.

Exemplo: byte b = B10010 ("B" significa que está em formato binário)

int é a variável padrão do programa. Esta variável consegue memorizar dados de -32768 até 32767. Esta variável consegue memorizar números negativos através da propriedade matemática chamada **complemento de dois.**

Exemplo: int ledPin = 13; int nomeVariavel = valorVariavel;

string é um vetor de caracteres terminado em um caractere nulo. Você pode usar uma string para memorizar dados obtidos em forma de caracteres. Numa string você denomina o nome e a quantidade máxima de caracteres que você pretende usar mais um caractere reservado para o caractere nulo. Usando uma string você pode modificar um caractere em particular no programa.

Exemplo: char str1[8] = { 'a', 'r', 'd', 'u', 'i', 'n', 'o', '/0' };

char str2[15] = "arduino";

TIPO	TAMANHO (BITS)	INTERVALO
char	8	-128 a 127
int	16	-32768 a 32767
float	32	3,4E-38 a 3,4E+38
double	64	1,7E-308 a 1,7E+308
void	0	sem valor

FUNÇÕES I

Anteriormente, já foi explicado um pouco a respeito sobre o void **setup**() e o void **loop**(), que mesmo não utilizando-os, eles precisam ser declarados.

Uma função é uma unidade autônoma do código, que foi feita para cumprir uma atividade em particular. Um programa consiste em várias funções. Uma função se resume em:

```
tipo nome(declaração de parâmetros){
comando1;
comando2;
...
}
```

Existem dois tipos de funções: as que retornam algum valor para a função onde está inserida e as funções que não retornam nenhum valor.

Tipo int Uma função **int** retorna um valor para a função que a chamou. Há duas formas de retornar de uma função, uma delas é quando o finalizador de função (***}***) é encontrado, ou quando for feito o uso da declaração **return**.

A declaração **return** retorna um valor especificado para a função que a chamou.

Exemplo:

```
int checkSensor(){
  if (analogRead(A0) > 400) {      // se a leitura do pino A0 for maior que 400
    return 1;                      // retorna 1 (verdadeiro)
  else{
    return 0; }                    // se não for, retorne 0 (falso)
}
```

Tipo void uma função tipo **void** não retorna nenhum valor para a função que a chamou, ou seja, as ações executadas nesta função não resultaram em números binários, variáveis booleanas ou caracteres para a função principal. Com o uso de **void** não é permitido usar a declaração **return**.

Usar funções pode simplificar o código e o deixar mais organizado. No **exemplo 2** percebemos que foi criada uma função apenas para o LED piscar e que foi inserida na função *void **loop**()*, com isso, se o código fosse extenso e seria necessário o LED piscar várias vezes durante o programa, aquela parte do código não precisaria ficar sendo repetida, assim, apenas chamaria a função *piscarLED()*, então quando ela terminasse de ser executada voltaria ao mesmo ponto em que parou.

COMO PROGRAMAR EM ARDUINO

Agora que você já sabe identificar comentários, bibliotecas, variáveis, consegue compreender como lidar com o tempo, sabe criar funções, identificar o corpo do programa e enviar e receber dados de entradas digitais e analógicas, podemos começar a programar em Arduino.

OPERADORES BOOLEANOS, DE COMPARAÇÃO E INCREMENTO E DECREMENTO.

Um passo importante para podermos programar corretamente é fazer o uso adequado dos operadores booleanos e os de comparação.

OPERADORES E COMPARAÇÃO

x é igual a y: x == y

x é diferente de y; x != y

x é menor que y: x < y

x é maior que y: x > y

x é menor ou igual a y: x <= y

x é maior ou igual a y: x >= y

OPERADORES BOOLEANOS

Os operadores booleanos verificam se uma função é verdadeira ou não.

OPERADOR E (&&)

A condição apenas será verdadeira se todos os itens foram verdadeiros.

Exemplo: IF (digitalRead(2) == 1 && digitalRead(3) ==1){

... }

Se a leitura digital dos pinos 2 e 3 forem 1 (HIGH), a condição é verdadeira.

OPERADOR OU (||)

A condição será verdadeira se algum dos itens for verdadeiro.

Exemplo: IF (x>0 || y>0){

Se x ou y for maior que 0, a condição será verdadeira.

OPERADOR NÃO (!)

A condição será verdadeira se o operador for falso.

Exemplo : IF (!x){

... }

Se x for falso, a condição é verdadeira.

OPERADORES DE INCREMENTO E DECREMENTO

Em Linguagem C trabalhamos com operadores unários e binários. Operadores unários agem em apenas uma variável e a modifica ou não, enquanto os operadores binários utilizam duas variáveis, pegam seus valores sem alterá-los e retorna um terceiro valor. A soma é um exemplo de operador binário. Os operadores de incremento e decremento são operadores unários, pois alteram a variável em que estão aplicados.

Então dizemos:

x ++ x--

Que isso é equivalente a:

x++ > x = x+1

x-- > x = x-1

Estes operadores podem ser pré-fixados ou pós-fixados. Os pré-fixados eles incrementam e retornam o valor da variável já incrementada, enquanto os pós-fixados eles retornam o valor da variável sem o incremento e depois incrementam a variável.

Pré-fixado	*Pós-fixado*
Se: x=48	**Se:** x=48
y=++x	y=x++
Então:	**Então:**
x=49	x=49
y=49	y=48

Os operadores de incremento e decremento são de grande utilidade neste tipo de linguagem quando se trata de realização de testes, como veremos adiante.

OUTROS USOS:

X+=y > x=x+y

x-=y > x=x-y

x*=y > x=x*y

x/=y > x=x/y

ESTRUTURAS DE CONTROLE DE FLUXO

Podemos dizer que as estruturas de controle de fluxo são a parte mais importante da programação em Arduino, pois toda a programação é executada em torno delas.

➔ IF

IF testa se certa condição foi alcançada, como por exemplo, se uma entrada analógica obteve um número específico. Se o resultado da condição for 0 (falsa), ela não será executada, caso o resultado seja 1 (verdadeira), a função será executada. O formato desta estrutura de controle de fluxo é:

```
IF (certaCondicao) {
    // comandos ... }
```

➔ **IF... ELSE**

Usar IF/else permite um controle maior sobre a estrutura, sendo que poderá ser realizado vários testes sem sair de uma única função. Podemos pensar no else como sendo um complemento ao comando IF.

```
if (certaCondição) {
    // comando A... }
else {
    // comando B... }
```

Podemos interpretar a estrutura da seguinte maneira: "**se** a **certa Condição** for verdadeira (1) execute o comando A, **senão** (se for falsa, 0), execute o comando B".

OBSERVAÇÃO!

Quando você estiver trabalhando com várias condições, onde não se restringe a apenas duas hipóteses, dentro da estrutura de controle **else,** você pode colocar mais funções **if/else** e assim por diante. Veja:

```
if (Condicao1) {
  // comando A... }
else if (Condicao2) {
  // comando B...  }
        else {
        // comando C }
```

➔ **FOR**

A estrutura FOR é uma das estruturas que se trabalha com loops de repetição. Essa estrutura normalmente é usada para repetir um bloco de informações definidas na função. Enquanto a condição for verdadeira, as informações contidas na função serão executadas. Sua forma geral é:

For (inicialização; condição; incremento)

{

//instrução(ou instruções) ; }

Vamos entender os blocos desta estrutura:

INICIALIZAÇÃO: inicialização é o que ocorre primeiro e apenas uma vez.

CONDIÇÃO: o bloco for trabalha com loops de repetição, cada vez que a função for repetida, a condição será testada, se ela for verdadeira (1), executará o que se encontra dentro das chaves.

INCREMENTO: se a condição for verdadeira, o incremento será executado, caso contrário, o loop é terminado.

Caso ainda não conseguiu entender a estrutura for equivale a mesma coisa que o seguinte código:

Inicialização; // primeiro: realiza-se a inicialização

```
if (condição) {          // segundo: verifica se a condição é verdadeira
        declaração;    // terceiro: executa a declaração contida na função
        incremento;    // realiza o incremento
        "Volta para o comando if"
        }
```

➔ SWITCH CASE

O switch case permite programar diferentes blocos de instruções para diferentes condições. Por exemplo: dizemos que você possui um sensor analógico que emite diferentes valores e cada valor indica que você necessita tomar uma ação específica, ou seja, uma instrução diferente. Sua forma geral é:

```
switch (valor) {
  case 1:
    //fazer algo quando valor é igual a 1
  break;

  case 2:
    //fazer algo quando valor é igual a 2
  break;

  default:
    // se nenhum caso se encaixa, fazer algo como padrão
}
```

Quando uma condição for verdadeira, break é responsável por interromper o código e seguir a programação adiante. Usa-se default caso nenhuma das condições especificadas forem verdadeiras, e então o bloco contido será executado.

➔ WHILE

A estrutura WHILE funciona como um loop de reptição. Enquanto a condição contida dentro dos parênteses permanecer verdadeira, a estrutura será executada. A partir do momento que a condição for falsa, o programa seguirá normalmente. Sua forma geral é:

```
while (condição) {
        // instrução 1...
        // intrução 2...
}
```

PROJETOS BÁSICOS

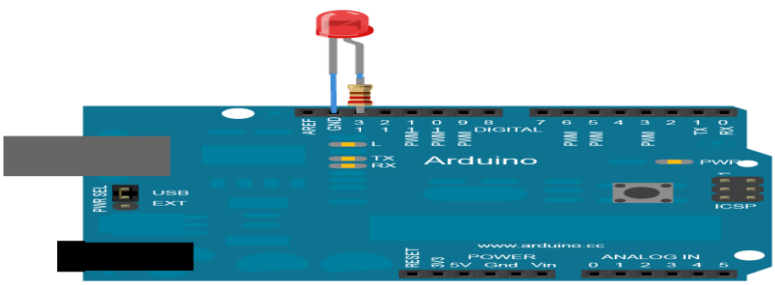
Projeto 1: LIGANDO UM LED

Exemplo piscar mostra a coisa mais simples que você pode fazer com um Arduino para ver a saída física: ela pisca um LED.

- Hardware necessário:
  - Arduino Board
  - LED
- Circuito: Para construir o circuito, coloque um resistor de 220 ohm no pino 13. Em seguida, anexar o terminal longo de um LED (o terminal positivo, o chamado ânodo) para o resistor. Anexar o terminal curto (o terminal negativo, chamado de cátodo) para o solo. Em seguida, conecte sua placa Arduino em seu computador, inicie o programa Arduino, e digite o código abaixo.

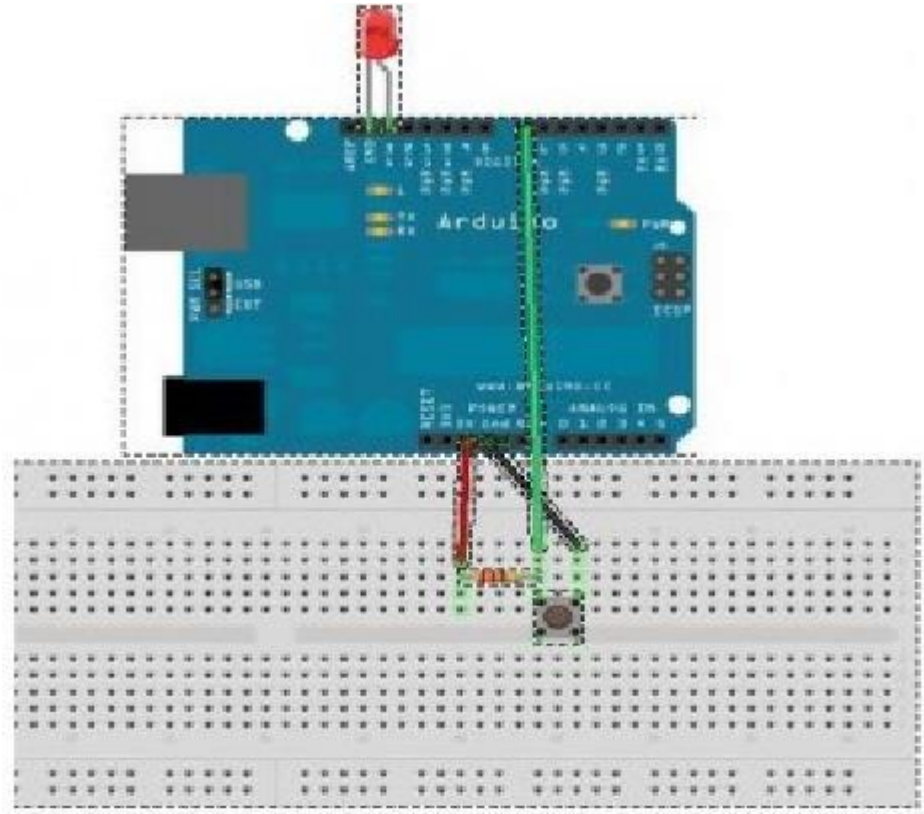
```
/ *
Piscar
Acende um LED por um segundo, então fora por um segundo,
repetidamente.
Este exemplo de código é de domínio público.
* /

void setup () {
/ / Inicializa o pino digital como uma saída.
/ / Pin 13 tem um LED conectado na maioria das placas Arduino:
pinMode (13, OUTPUT);
}

void loop () {
digitalWrite (13, HIGH); / / set o LED
delay (1000); / esperar / por um segundo
digitalWrite (13, LOW); / / definir o off LED
delay (1000); / esperar / por um segundo
}
```

A maioria das placas Arduino já tem um LED conectado ao pino 13 da placa em si. Se você executar este exemplo sem hardware conectado, você verá piscar um LED.



Projeto 2: Ligando/Desligando LED através de botão

Primeiramente conectar três fios à placa Arduino, um no pino Digital 7, um na alimentação de 5V e o outro no pino GND. O fio correspondente ao pino 7 será ligado em um terminal do pushButton, o que representa o GND será ligado em um terminal oposto ao pino 7 do pushButton. Na mesma trilha do pino 7, conectar um resistor de 220ohms, e o outro terminal do resistor no pino de alimentação equivalente à 5V. Como na imagem a seguir:



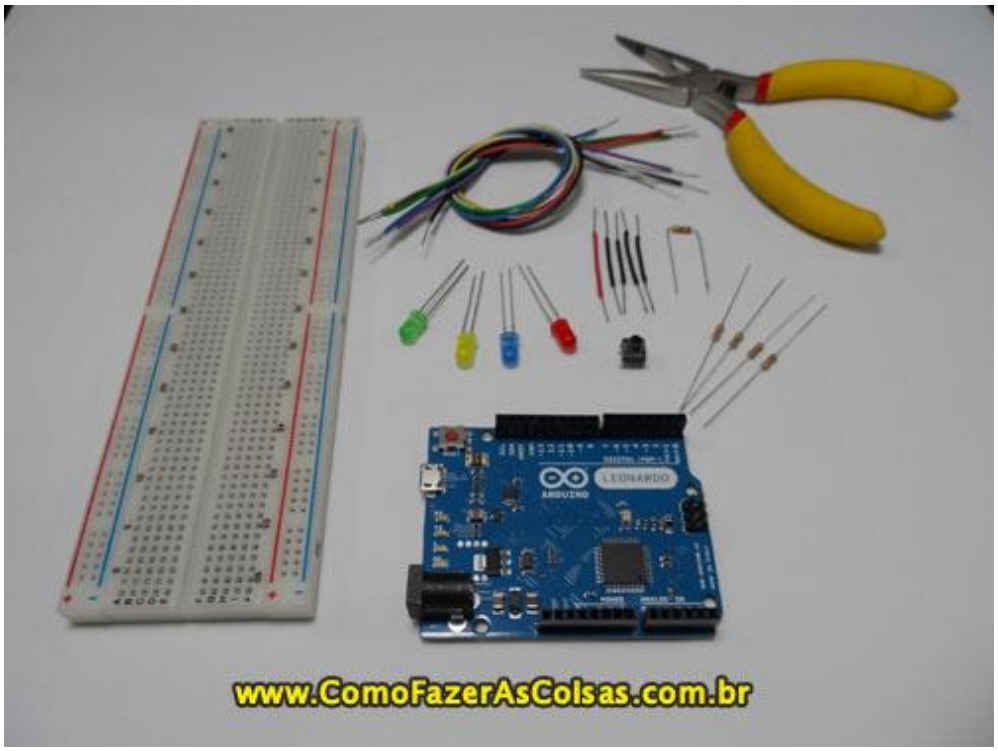
A seguir, abrir o Arduino IDE e escrever o seguinte código:

```
int ledPin = 13; // porta 13 em output para o LED
int inPin = 7; // porta 7 em input (para o push button)
int val = 0; // variável para ler o status do pino
void setup() {
pinMode(ledPin, OUTPUT); // declare LED como output
pinMode(inPin, INPUT); // declare pushbutton como input
}
void loop(){
val = digitalRead(inPin); // ler a entrada de valor
if (val == HIGH) {// verificar se a entrada é alta
digitalWrite(ledPin, LOW); // LED OFF
} else {
digitalWrite(ledPin, HIGH); // LED ON
}
}
```

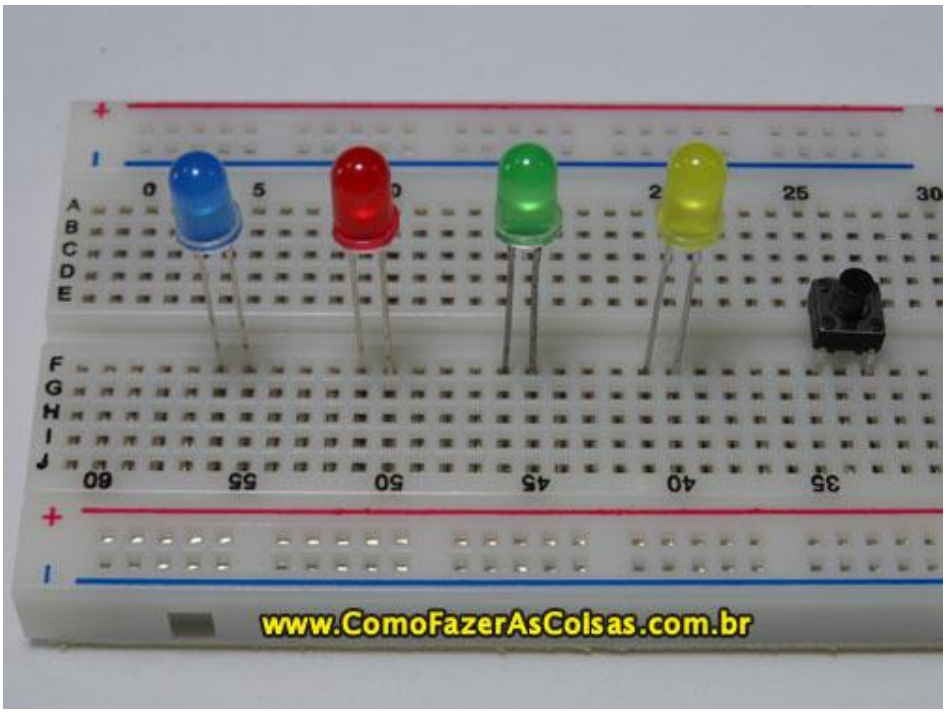
Projeto 3 – LEDs com efeito

Para desenvolver o projeto arduino botão que acende vários leds e com efeito você vai precisar de:

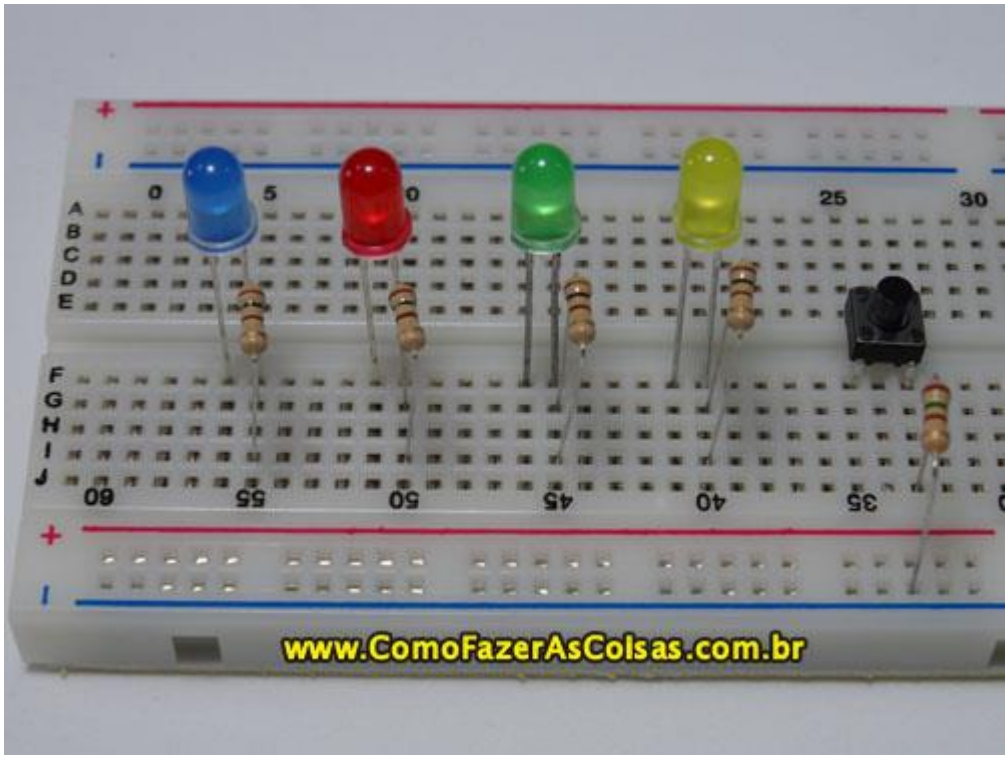
- um arduino uno ou leonardo;
- uma protoboard;
- quatro leds de cores diferentes;
- quatro resistores de 100 ohms;
- um resistor de 150 ohms;
- um botão do tipo push button;
- sete fios jumper grandes;
- cinco fios jumper pequenos;
- a IDE do arduino instalada no seu computador;
- um cabo USB para conectar o arduino ao computador;
- um alicate de bico (opcional).



A primeira coisa que vamos fazer é colocar os leds e o botão na protoboard. Neste caso as pernas grandes dos leds foram conectadas nas colunas 40, 45, 50, 55. As pernas pequenas dos leds foram conectadas nas colunas 41, 46, 51, 56. O botão teve suas perninhas conectadas as colunas 33 e 35.



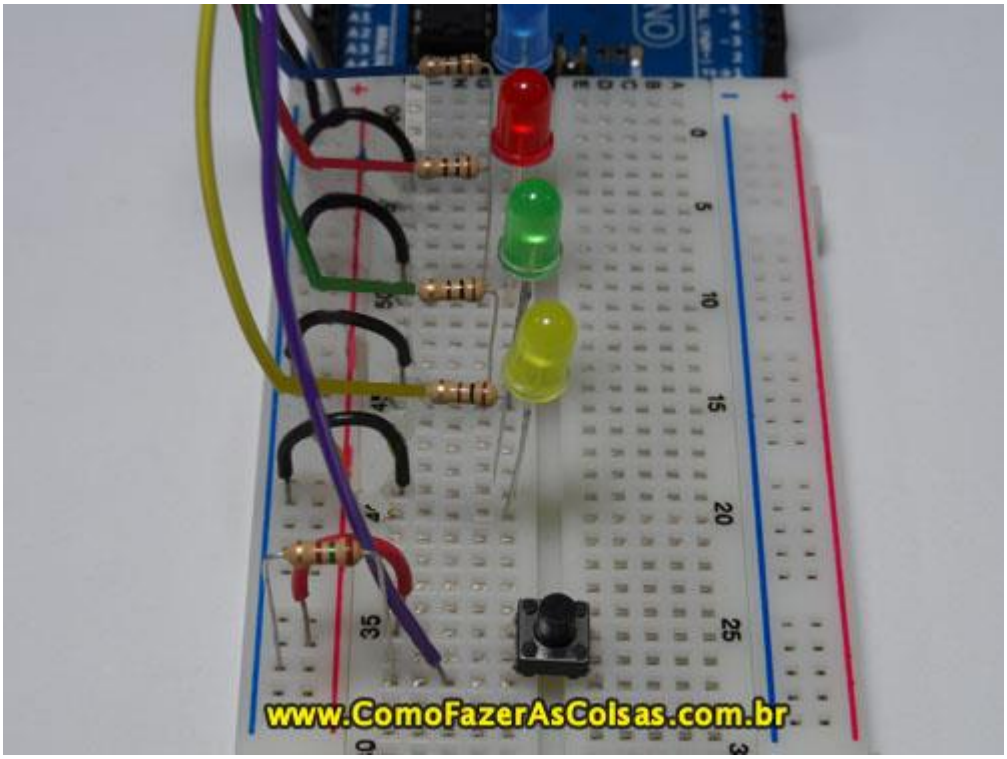
Agora chegou a vez dos resistores. Conecte os quatros resistores de 100 ohms nas colunas que estão conectadas as pernas grandes dos leds (colunas 40, 45, 50 e 55) conforme mostra a imagem abaixo. Conecte o resistor de 150 ohms na coluna 33 e na linha horizontal negativa da protoboard (linha azul neste caso) como na imagem abaixo.



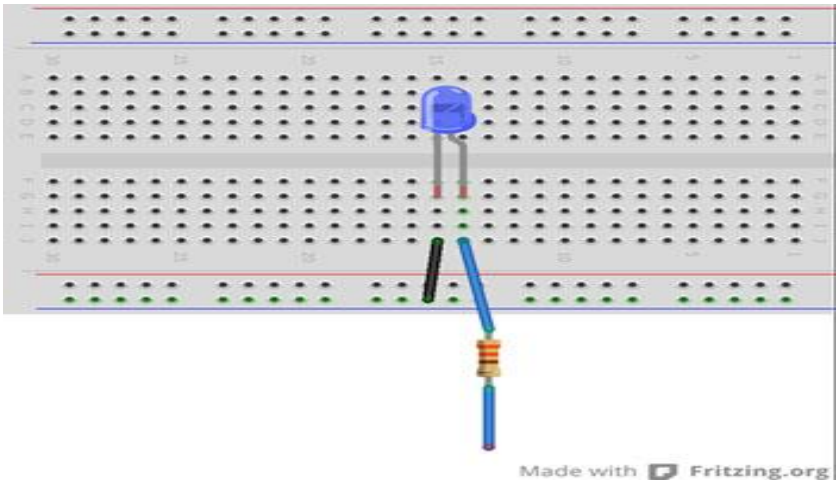
Conecte os fios jumper pequenos na linha horizontal negativa da protoboard e nas colunas das pernas pequenas dos leds, no caso as colunas 41, 46, 51, 56. Conecte a coluna 35, que esta a perninha do botão um fio jumper ligado a linha horizontal positiva da protoboard, linha vermelha neste caso.

Chegou a hora de conectar os fios jumper maiores na protoboard. Para cada perna grande de um led você vai ligar um fio jumper, neste caso o fio azul esta ligado no resistor da coluna 55, o fio vermelho no resistor da coluna 50, o fio verde no resistor da coluna 45 e o fio amarelo no resistor da coluna 40. Na coluna 33 entre o botão e o resistor você vai ligar outro fio jumper, neste caso é o fio jumper roxo. Para finalizar conecte um fio jumper na linha horizontal negativa azul, neste caso foi o fio preto, e outro fio na linha horizontal positiva vermelha, neste caso foi o fio cinza.

Veja na imagem abaixo outro ponto de vista da protoboard com todos os componentes do projeto arduino com um botão que acende os leds com efeito. O próximo passo é conectar os fios jumper grandes no arduino.



Abaixo está melhor exemplificado como deve ser feita a ligação entre cada um dos led, resistores e fios jumper.



Os fios jumpers grandes serão conectados da seguinte forma:

- fio azul no pino digital 11;
- fio vermelho no pino digital 10;
- fio verde no pino digital 9;
- fio amarelo no pino digital 8;
- fio roxo no pino digital 2;
- fio preto no pino GND;
- fio cinza no pino 5V.

Os pinos digitais 8, 9, 10, e 11 do arduino serão responsáveis por enviar a energia para acender os leds conectados na protoboard. O pino digital 2 será um pino de entrada, que receberá ou não energia, conforme o presionamento do botão, possibilitando a leitura do estado do botão (pressionado ou não pressionado). O pino GND é o terra do arduino, onde estão conectadas todas as perninhas menores dos leds. E o pino 5V é o responsável por mandar energia para o botão possibilitando o controle do seu estado, juntamente com o pino digital 2 do arduino.

Pronto! o seu projeto físico está totalmente pronto, mas ainda falta a parte mais legal, que é programar o arduino para que ele faça o que desejamos. Então vamos conectar o arduino ao computador abrir a IDE do arduino e mãos a obra.

Agora chegou a hora da programação do arduino. Segue abaixo, após a imagem da IDE, o código fonte completo e testado deste projeto. Copie e cole ou faça você mesmo a sua versão do programa de controle do arduino, compile, e posteriormente faça o upload do programa para o arduino.


sketch_botao_led_efeitos | Arduino 1.0.1
File Edit Sketch Tools Help
sketch_botao_led_efeitos
/*
Projeto Arduino acendendo com efeito e apagando os leds com botão.
Por Jota

--=<| www.ComoFazerAsCoisas.com.br |>=--
---
*/
//Declaração das constantes referentes aos pinos digitais.
const int ledAmarelo = 8;
const int ledVerde = 9;
const int ledVermelho = 10;
const int ledAzul = 11;
const int botao = 2;
//Declaração da variável que possuirá os estados do botão.
int estadoBotao = 0;
//Método setup, executado uma vez assim que o Arduino é ligado.
void setup() {
pinMode(ledAmarelo,OUTPUT); //Definindo pino 8 como saída.
Done Saving.
www.ComoFazerAsCoisas.com.br
3
Arduino Leonardo on COM4

Código fonte comentado do projeto arduino com botão que acende leds com efeito.

```
/*
Projeto Arduino acendendo com efeito e apagando os leds com botão.

----------------------------------------
*/

//Declaração das constantes referentes aos pinos digitais.
const int ledAmarelo = 8;
const int ledVerde = 9;
const int ledVermelho = 10;
const int ledAzul = 11;
const int botao = 2;

//Declaração da variável que possuirá os estados do botão.
int estadoBotao = 0;

//Método setup, executado uma vez assim que o Arduino é ligado.
void setup() {
  pinMode(ledAmarelo,OUTPUT);  //Definindo pino 8 como saída.
  pinMode(ledVerde,OUTPUT);    //Definindo pino 9 como saída.
  pinMode(ledVermelho,OUTPUT); //Definindo pino 10 como saída.
  pinMode(ledAzul,OUTPUT);     //Definindo pino 11 como saída.

  pinMode(botao,INPUT);        //Definindo pino 2 como entrada.
}

//Método loop, executado enquanto o Arduino estiver ligado.
void loop() {
  estadoBotao = digitalRead(botao);

  if (estadoBotao == HIGH) {
    //Acendendo os leds caso o botão esteja pressionado, com
    //um intervalo de tempo (delay) entre os acendimentos para
    //criar um pequeno efeito.
    digitalWrite(ledAmarelo,HIGH);
    delay(200);
    digitalWrite(ledVerde,HIGH);
    delay(200);
    digitalWrite(ledVermelho,HIGH);
    delay(200);
    digitalWrite(ledAzul,HIGH);
    delay(200);
  } else {
    //Apagando os leds caso o botão não esteja pressionado.
    digitalWrite(ledAmarelo,LOW);
    digitalWrite(ledVerde,LOW);
    digitalWrite(ledVermelho,LOW);
    digitalWrite(ledAzul,LOW);
  }}
```

Em resumo, o objetivo deste programa é verificar se o botão está pressionado, e caso esteja acende os leds. Caso não esteja pressionado apaga os leds.

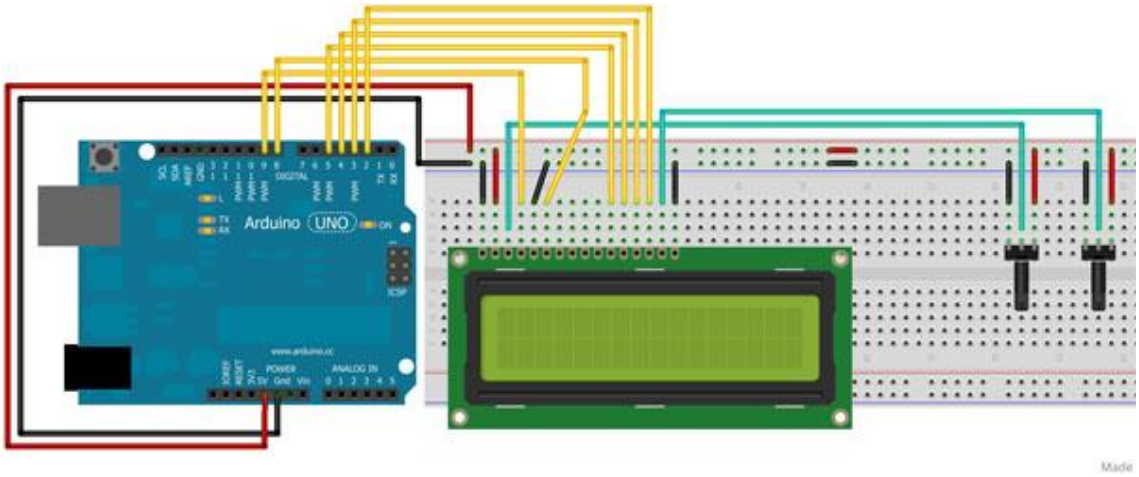
Projeto 4 – Arduino com display LCD

Desenvolver um projeto arduino com display LCD 16x2 do tipo JHD 162A. Um display LCD eleva o seu projeto a outro nível, pois possibilita uma saída mais amigável e a possibilidade de interação por parte dos usuários. Usar displays LCDs em projetos arduino ao contrário do que parece é bem fácil, pois todas as funcionalidades já estão codificadas na biblioteca **LiquidCrystal.h** que será incluída, é terá suas funções chamadas a partir do nosso programa desenvolvido para este projeto. Outro ponto importante é o uso de dois potenciômetros que servem para controlar o brilho e o contraste da tela LCD permitindo que você defina um nível adequado de visibilidade para o display LCD, como poder ser visto no vídeo ao final deste post, que mostra este projeto concluído e em funcionamento.

Para desenvolver o projeto arduino com display lcd você vai precisar de:

- um arduino, poder ser qualquer versão (Uno, Leonardo, Mega...);
- uma protoboard;
- um display LCD do tipo JHD 162A de 16 colunas e 2 linhas;
- dois potenciômetros de 10 K ohms;
- fios jumper.

Segue abaixo o esquema de montagem do projeto, que servirá como referência de ligação entre os componentes ao longo do projeto.



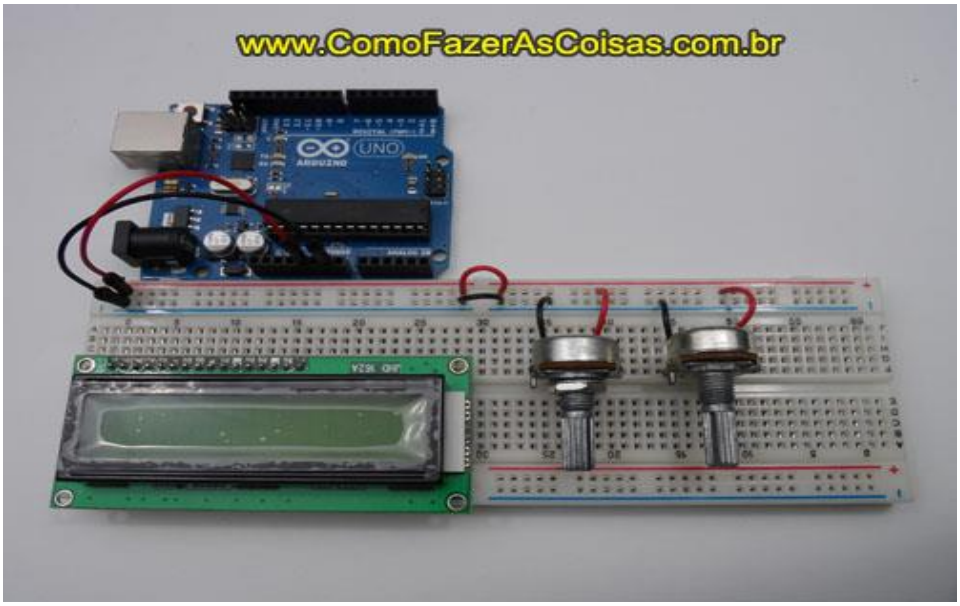
Segue abaixo o *datasheet*, detalhando a funcionalidade de cada pino, do display LCD JHD 162A que foi usado neste projeto.

PIN No	Name	Function
1	VSS	Ground voltage
2	VCC	+5V
3	VEE	Contrast voltage
4	RS	Register Select 0 = Instruction Register 1 = Data Register
5	R/W	Read/ Write, to choose write or read mode 0 = write mode 1 = read mode
6	E	Enable 0 = start to latch data to LCD character 1= disable
7	DB0	Data bit 0 (LSB)
8	DB1	Data bit 1
9	DB2	Data bit 2
10	DB3	Data bit 3
11	DB4	Data bit 4
12	DB5	Data bit 5
13	DB6	Data bit 6
14	DB7	Data bit 7 (MSB)
15	BPL	Back Plane Light +5V or lower (Optional)
16	GND	Ground voltage (Optional)

A imagem abaixo mostra o verso do display LCD JHD 162A. No canto inferior direito vemos o código deste display.

Vamos iniciar a montagem do nosso projeto conectando o display LCD na protoboard. Neste caso o pino 1 do display LCD foi conectado a coluna 0 da protoboard.

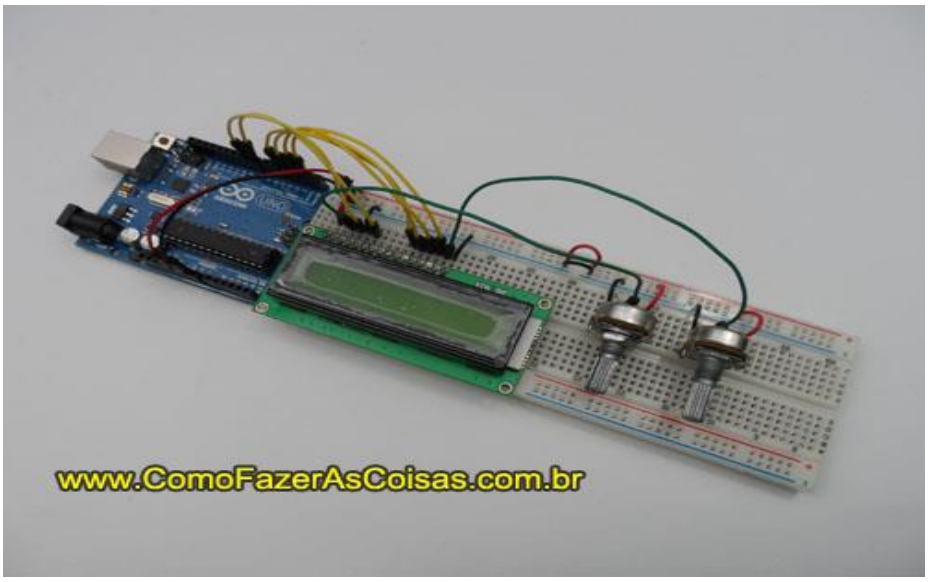
Agora conecte o 5V e o GND do arduino na protoboard. Conecte os dois potenciômetros e também os ligue ao 5V e GND. Neste caso o fio vermelho é o 5V e o preto e o GND.



Agora chegou a hora de ligar o display LCD ao arduino. A ligação será feita da seguinte forma.

- pino 1 do LCD ligado ao GND do arduino;
- pino 2 do LCD ligado ao 5V do arduino;
- pino 3 do LCD ligado ao pino central do primeiro potênciômetro (controle de contraste);
- pino 4 do LCD ligado ao pino digital 9 do arduino;
- pino 5 do LCD ligado ao GND do arduino;
- pino 6 do LCD ligado ao pino digital 8 do arduino;
- pino 11 do LCD ligado ao pino digital 5 do arduino;
- pino 12 do LCD ligado ao pino digital 4 do arduino;
- pino 13 do LCD ligado ao pino digital 3 do arduino;
- pino 14 do LCD ligado ao pino digital 2 do arduino;
- pino 15 do LCD ligado ao pino central do segundo potênciômetro (controle do brilho);
- pino 16 do LCD ligado ao GND do arduino.

Lembrando que você poderá verificar o esquema de montagem do projeto no início deste post, que mostra com detalhes como as ligações entre os componentes foram feitas.

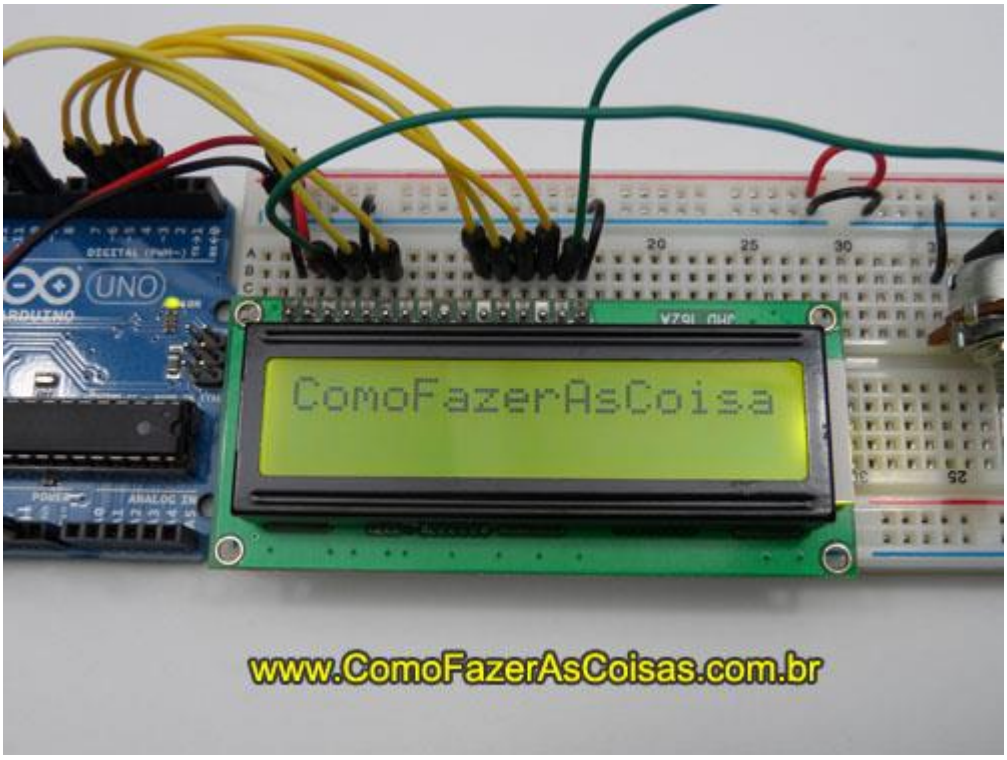


Agora chegou a hora de programar o arduino. Segue abaixo o código fonte comentado deste projeto, lembrando que as mensagens exibidas no display LCD, através do comando lcd.print poderão ser alteradas.

Código fonte do projeto.

```
/*
Projeto Arduino com LCD.

---------------------------------------
*/

//Define a utilização da biblioteca para controle de telas LDCs.
 #include <LiquidCrystal.h>

//Criando um objeto da classe LiquidCrystal e
//inicializando com os pinos da interface.
LiquidCrystal lcd(9, 8, 5, 4, 3, 2);

void setup() {
  //Inicializando o LCD e informando o tamanho de 16 colunas e 2 linhas
  //que é o tamanho do LCD JHD 162A usado neste projeto.
  lcd.begin(16, 2);
}

void loop() {
  lcd.clear();                  //limpa o display do LCD.
  lcd.print("Oi!!! ");          //imprime a string no display do LCD.
  delay(2000);

  lcd.setCursor(0,1);           //posiciona o cursor na coluna 0 linha 1
do LCD.
  lcd.print("Tudo Bem???");     //imprime a string no display do LCD.
  delay(2000);

  lcd.clear();
  lcd.print("Quer aprender");
  lcd.setCursor(0,1);
  lcd.print("este projeto?");
  delay(4000);

  lcd.clear();
  lcd.print("www.ComoFazerAsCoisas.com.br");
  delay(1000);

  //Rolando o display para a esquerda 12 vezes
  for (int i = 0; i < 12; i++) {
    lcd.scrollDisplayLeft();
    delay(600);
  }

  delay(1000);
}
```

Após transferir o código fonte para a IDE do arduino compile-o e depois faça o upload do mesmo. Ajuste os potenciômetros para que as mensagens fiquem visíveis e pronto, seu projeto foi concluído.



REFERÊNCIAS BIBLIOGRÁFICAS

http://ordemnatural.com.br/pdf-files/CartilhadoArduino_ed1.pdf <Acesso dia: 14/01/2014>

http://arduinoprojetoscaseiros.blogspot.com.br/2011/11/ligando-um-led.html <Acesso dia: 14/01/2014>

http://blog.samuelcavalcante.com/?p=326 <Acesso dia: 14/01/2014>

http://arduinotecbr.wordpress.com/2012/11/09/apostila-de-arduino-para-iniciantes/ <Acesso dia: 14/01/2014>

http://pt.slideshare.net/Miojex360/apostila-para-programar-arduino <Acesso dia: 14/01/2014>

http://arduinoifsul.blogspot.com.br/2012/08/exemplo-basico-led-piscando.html#!/2012/08/exemplo-basico-led-piscando.html <Acesso dia: 14/01/2014>