Fundação Universidade Federal de Mato Grosso do Sul
Faculdade de Computação - FACOM

Introdução ao Arduino

Grupo de Robótica

UFMS 2012

SUMÁRIO

1. Introdução ao Arduino

1.1 História do 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.

1.2 O que é um Arduino?

Arduino é uma placa de controle de entrada de dados (IN), como sensores, e saída de dados (OUT), como motores e leds, com cristal oscilador de 16 Mhz, um regulador de tensão de 5 V, botão de *reset*, plugue de alimentação, pinos conectores, e alguns LEDs para facilitar a verificação do funcionamento. A porta USB já fornece alimentação enquanto estiver conectado ao computador, e a tensão de alimentação quando desconectado pode variar de 7 V a 12 V, graças ao regulador presente na placa.

No Arduino, informações ou ordens são transmitidas de um computador para a placa através de *Bluetooth*, *wireless*, USB, infravermelho, etc. Essas informações devem ser traduzidas utilizando a linguagem *Wiring* baseada em C/C++.

Figura 1. Caixa do arduino

1.3 Por que usar Arduino?

- Baixo custo - Uma pessoa pode comprar um Arduino pagando em torno de R$ 50,00.
- Software para várias plataformas - Microsoft Windows, Mac OS X e Linux.
- Linguagem simples - Os desenvolvedores do Arduino tentam manter sua linguagem fácil de usar para iniciantes, mas flexível o bastante para usuários avançados.
- Software livre - O Arduino é completamente um software livre. Se quiser construir seu próprio software ou modificar um, você é livre para isso. Além disso, o *Web* site oficial do Arduino contém um wiki extensivo no qual amostras de código e exemplos são compartilhados livremente.
- Existe uma comunidade ativa para usuários, por isso, existe uma quantidade enorme pessoas que podem te ajudar.

2. Modelos de Arduino

2.1 Arduino UNO

Figura 2. Arduino UNO

É uma placa com micro controlador Atmega328. Possui 14 entradas/saídas digitais, 6 entradas analógicas, um cristal oscilador de 16MHz, conexão USB, uma entrada para fonte, soquetes para ICSP, e um botão de reset. A placa contém todo o necessário para usar o micro controlador. Simplesmente conecte-a a um computador com o cabo USB ou ligue a placa com uma fonte

AC-DC (ou bateria). O Uno seleciona automaticamente a fonte de alimentação (USB ou fonte externa).

2.2 Arduino 2009

Figura 3. Arduino 2009

Praticamente igual ao Arduino Uno.

2.3 Arduino Mega 2560

Figura 4. Arduino Mega

Possui uma considerável quantidade de portas, o que viabiliza a implementação de projetos mais complexos garantindo a eficiência e o baixo custo.

2.4 Arduino Nano

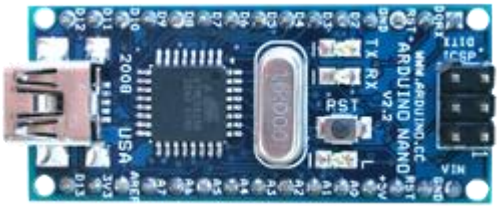

Figura 5. Arduino Nano

O Arduino Nano é uma pequena versão de Arduino parecida com o Arduino UNO, pois também possui um chip ATMega328, na versão SMD. Possui um conector para cabos Mini-USB para gravação. Uma das diferenças entre esta placa e a Arduino UNO ou Arduino 2009, é que esta placa possui 2 entradas analógicas a mais e um jumper de +5V AREF. Esta placa não possui um conector para fonte externa, mas é possível alimentá-la pelo pino Vin. O Arduino Nano automaticamente seleciona a maior alimentação fornecida.

3. **Arduino Mega**

O Arduino Mega 2560 é uma placa de micro controlador baseada no ATmega2560 (datasheet). Ele possui 54 pinos de entradas/saídas digitais, 16 entradas analógicas, 4 UARTs (portas seriais de hardware), um oscilador de cristal de 16 MHz, uma conexão USB, uma entrada de alimentação, uma conexão ICSP e um botão de reset. Ele contém tudo o que é necessário para dar suporte ao micro controlador; basta conectar a um computador com um cabo USB ou a uma fonte de alimentação e já está pronto para começar. O mega é compatível com a maioria dos shields desenhados para os Arduino Uno, Duemilanove e para o Diecimila. Possui ainda o dobro de memória do antigo Arduino Mega.

3.1 Definições:

- **Memória flash :** é capaz de preservar os dados armazenados por um longo tempo sem a presença de corrente elétrica.
- **EEPROM :** memória não volátil, pode ser programada e apagada várias vezes, eletricamente.
- **SRAM:** memória de acesso aleatório que mantém os dados armazenados desde que seja mantida sua alimentação elétrica
- **Micro controlador:** é um “computador-num-chip”, contendo um processador, memória e periféricos de entrada/saída. É um micro processador que pode ser programado para funções específicas, em contraste com outros micro processadores de propósito gerais (como os utilizados nos PCs). Eles são embarcados no interior de algum

outro dispositivo (geralmente um produto comercializado) para que possam controlar as funções ou ações do produto. Outro nome para o micro controlador, portanto, é controlador embutido.

- **Shields:** são placas de circuito impresso com uma função específica.

3.2 Especificações

Micro controlador	ATmega2560
Tensão de operação	5V
Tensão de entrada (recomendada)	7-12V
Tensão de entrada (limites)	6-20V
Pinos de entrada e saída (I/O) digitais	54 (dos quais 14 podem ser saídas PWM)
Pinos de entradas analógicas	16
Corrente DC por pino I/O	40mA
Corrente DC para pino de 3,3V	50mA
Memória Flash	256KB (dos quais 8KB são usados para o bootloader)
SRAM	8KB
EEPROM	4KB
Velocidade de Clock	16MHz

Tabela 1: Características da placa Arduino MEGA 2560

3.3 Alimentação

O Arduino Mega2560 pode ser alimentado pela conexão USB ou com uma fonte externa. A entrada de alimentação é selecionada automaticamente. Alimentação externa (não USB) pode ser tanto de uma fonte como de baterias. A fonte pode ser conectada plugando um conector de 2,1mm, positivo no centro, na entrada de alimentação. Cabos vindos de uma bateria podem ser

inseridos nos pinos terra (Gnd) e entrada de voltagem (Vin) do conector de energia.

A placa pode operar com alimentação externa entre 6 e 20 volts. No entanto, se menos de 7 volts forem fornecidos o pino de 5V pode fornecer menos de 5 volts e a placa pode ficar instável. Com mais de 12V o regulador de voltagem pode superaquecer e danificar a placa. A faixa recomendável é de 7 a 12 volts.

Os pinos de alimentação são os seguintes:

• **VIN**. Relacionado à entrada de voltagem da placa Arduino quando se está usando alimentação externa (em oposição aos 5 volts fornecidos pela conexão USB ou outra fonte de alimentação regulada). É possível fornecer alimentação através deste pino ou acessá-la se estiver alimentando pelo conector de alimentação.

• **5V.** Fornecimento de alimentação regulada para o micro controlador e outros componentes da placa.

• **3V3.** Uma alimentação de 3,3 volts gerada pelo chip FTDI. A corrente máxima é de 50 mA.

• **GND.** Pinos terra.

3.4 Entrada e Saída

Cada um dos 54 pinos digitais do Mega2560 pode ser usado como entrada ou saída. Eles operam a 5 volts. Cada pino pode fornecer ou receber um máximo de 40 mA e possui um resistor interno de 20-50 KΩ.

3.5 LED 13

Há um LED conectado ao pino digital 13. Quando o pino está em HIGH o led se acende.

3.6 Memória

O Atmega2560 tem 256 KB de memória flash para armazenamento de código(dos quais 8KB são usados pelo bootloader), 8 KB de SRAM e 4 KB de EEPROM.

3.7 Características Físicas do Arduino MEGA 2560

Na figura 6 está representada a localização dos pinos de Entrada/Saída Digital, as entradas analógicas, as portas de alimentação, entradas de comunicação serial, entre outros.

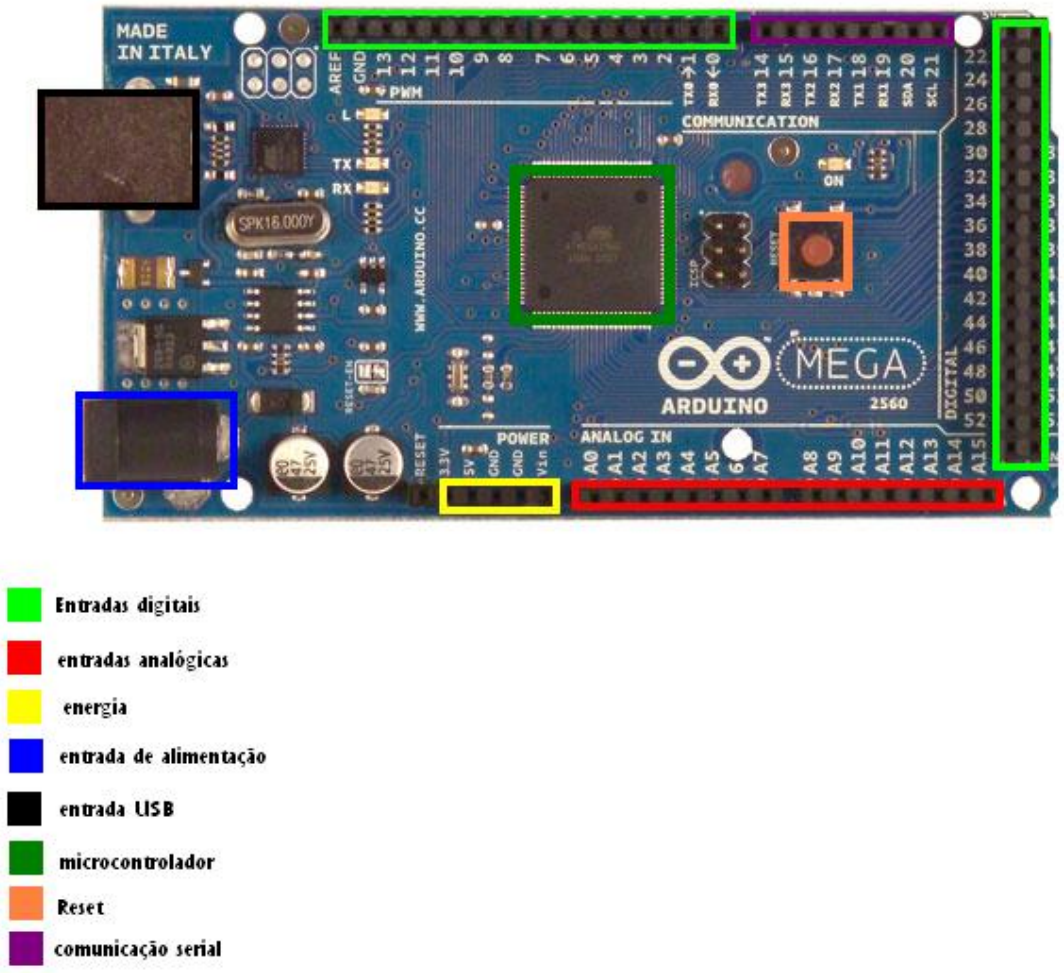

Figura 6. Caracterização do Arduino Mega

4. Descrição do funcionamento de uma Protoboard

A *protoboard* (ou *breadboard*) é uma base de construção de protótipos eletrônicos. Ela é muito utilizada, pois não requer solda, ou seja, é reutilizável. Isto torna mais fácil de usar para criação de protótipos temporários. A utilização de uma *protoboard* em montagem com projetos de Arduino torna possível a construção de circuitos mais complexos.

A ligação de circuitos é feita através de *jumpers* (basicamente pequenos fios), que são utilizados para ligar temporariamente componentes eletrônicos na *protoboard*. A faixa central tem o tamanho especifico para componentes eletrônicos pequenos. Normalmente, uma *protoboard* possui quatro matrizes, mas este número pode variar.

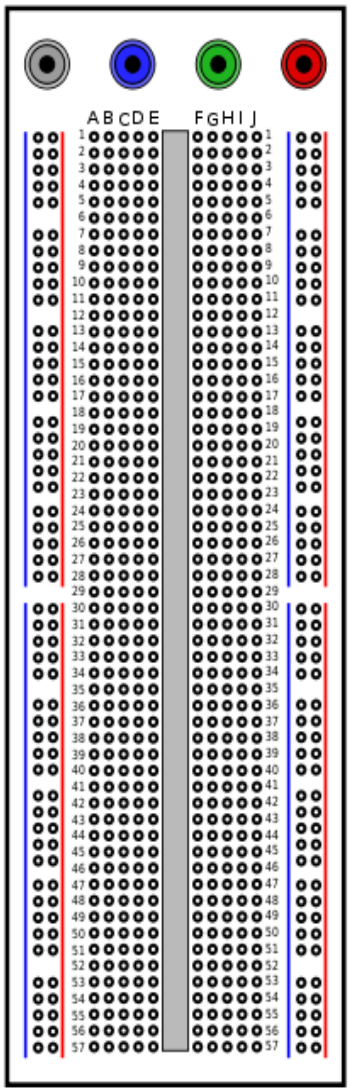

Figura 7. Exemplo de uma *protoboard*

Nos exemplos de *protoboard* já com circuito (figuras 8 e 9), pode-se notar que o encaixe dos circuitos torna-se bastante complexo com acréscimo de muitos *jumpers* (fios) e circuitos**.**

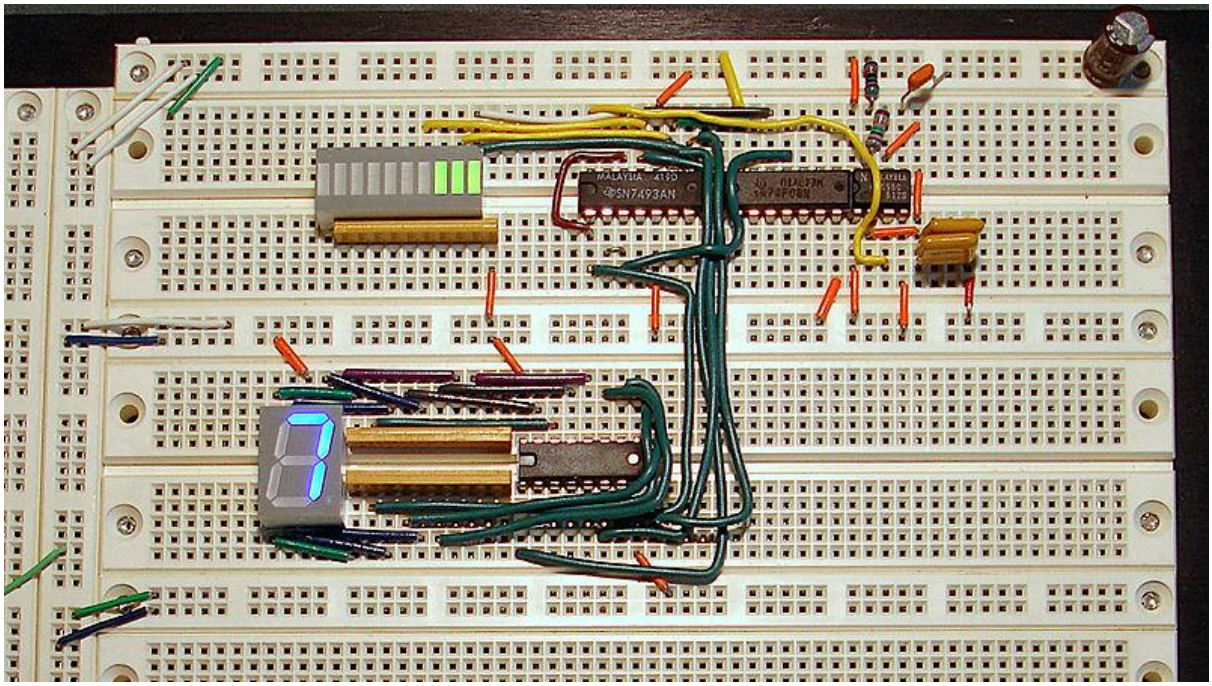

Figura 8. Circuito simples – poucos jumpers

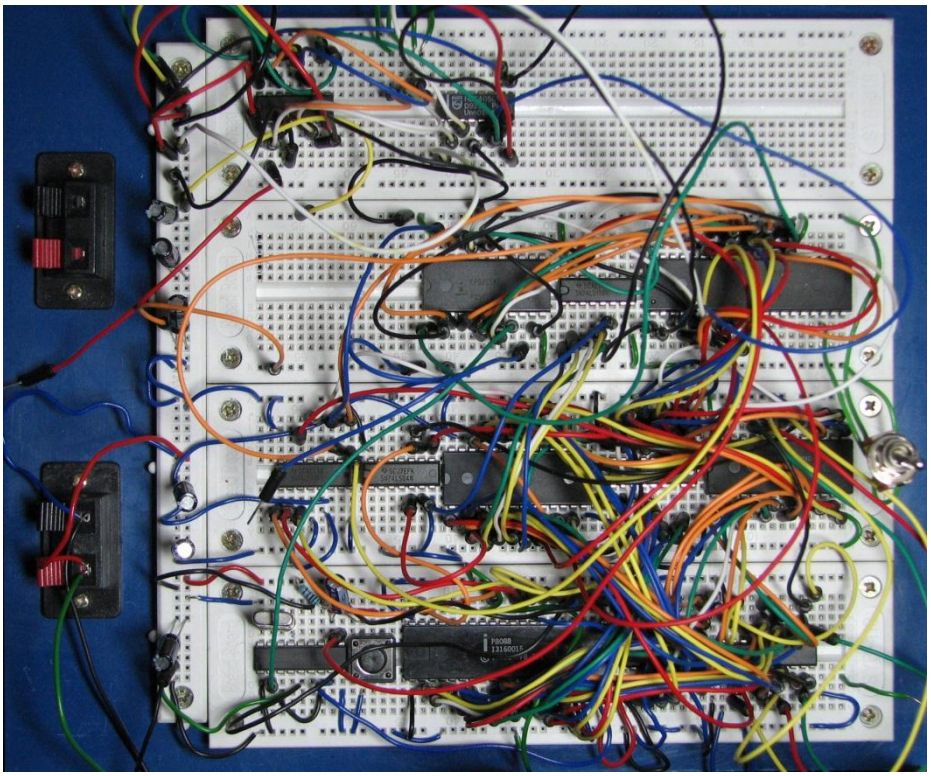

Figura 9. Circuito complexo – muitos jumpers

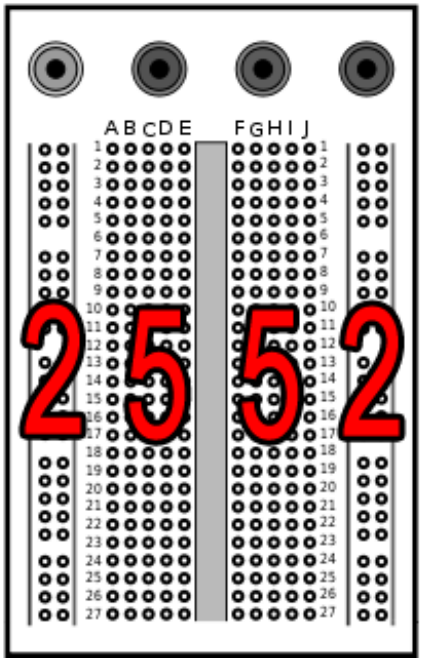

Figura 10. Protoboard – divisão por matrizes

A *protoboard* é composta de dois tipos de matrizes principais: uma com duas colunas, chamada de matriz 2 e outra com cinco colunas, chamada de matriz 5. Elas se diferem no modo de transmissão de energia e dados.

A matriz 2 geralmente é usada para ligação inicial de energia, e a 5 é usada na utilização de componentes no circuito.A matriz de 2 tem sua transmissão de coluna em coluna, enquanto a matriz 5 tem a transmissão de linha em linha.

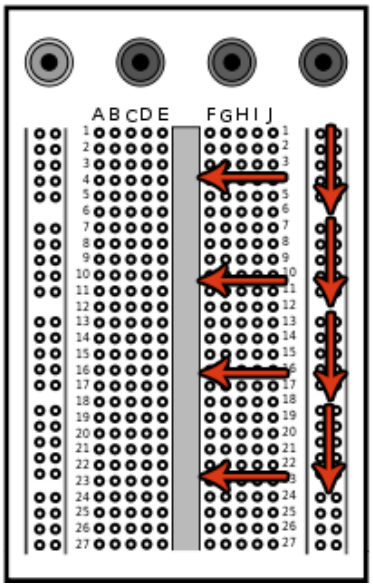

Figura 11. Protoboard – transmissão nas matrizes

Um exemplo de energização da *protoboard*, na figura 12 a cor vermelha corresponde a parte energizada da *protoboard* e a parte azul são jumpers (fios). Como se pode ver, a matriz 2 esquerda tem ligação com a fonte de energia. Toda essa coluna que está interligada na fonte por um jumper está energizada, característica da matriz 2. Ao colocarmos um *jumper* dessa matriz 2 interligando-a com a linha cinco da matriz esquerda de cinco colunas, energizamos esta linha. Ao colocarmos um *jumper* da linha cinco da matriz esquerda de cinco colunas para a linha cindo da matriz direita de cinco colunas, também energizamos essa linha. E assim por diante.

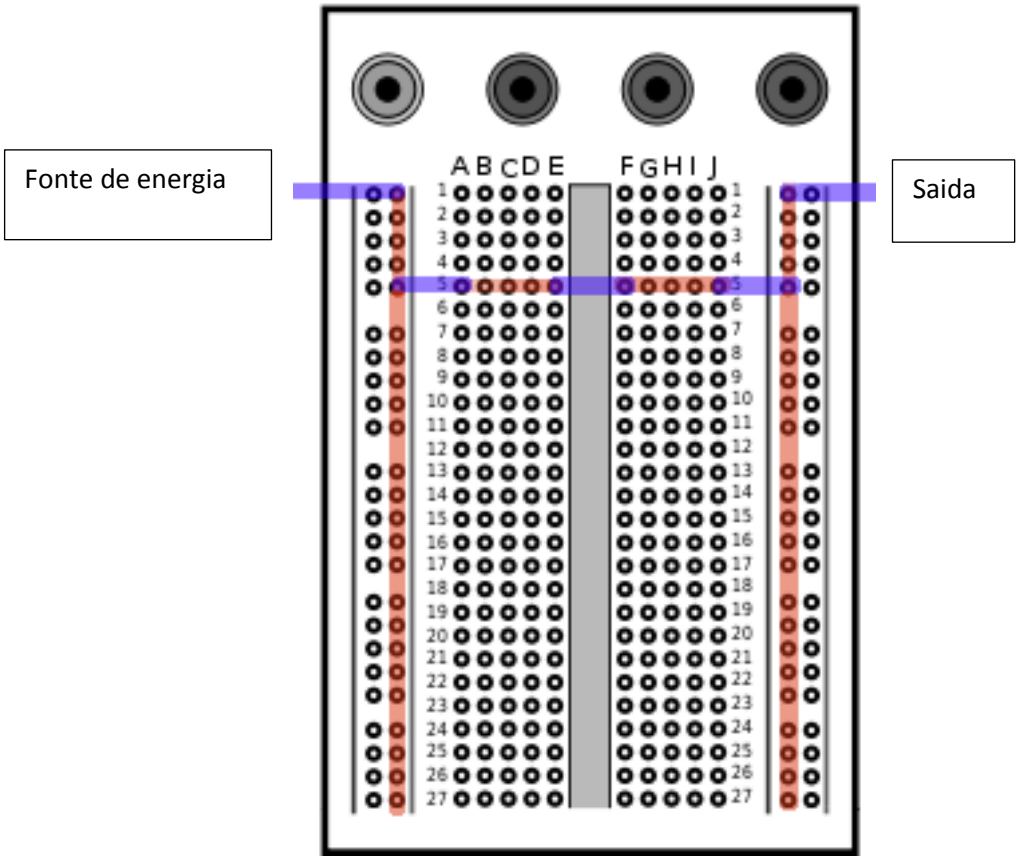

Figura 12. Exemplo do funcionamento das matrizes em uma *protoboard*

5. Resistores

5.1 O que são resistores?

São elementos que apresentam resistência à passagem de corrente elétrica, quanto maior a sua resistência, menor é a corrente elétrica que passa num condutor. Os resistores possuem um formato cilíndrico e faixas coloridas que definem o seu valor em Ohms. Servem para opor-se a passagem de corrente, ficando assim certa tensão retida no mesmo.

Um resistor ideal é um componente com uma resistência elétrica que permanece constante independentemente da tensão ou corrente elétrica que circular pelo dispositivo.

O valor de um resistor pode ser facilmente identificado analisando as cores que apresenta em torno dele ou então usando um ohmímetro (instrumento de medição de resistores).

O material do resistor é uma película fina de carbono (filme), depositada sobre um pequeno tubo de cerâmica. O filme resistivo é enrolado em hélice por fora do tubinho até que a resistência entre os dois extremos fique tão próxima quanto possível do valor que se deseja. São acrescentados terminais (um em forma de tampa e outro em forma de fio) em cada extremo. Em seguida o resistor é recoberto com uma camada isolante, e no fim suas faixas coloridas transversais são pintadas para indicar o valor da sua resistência.

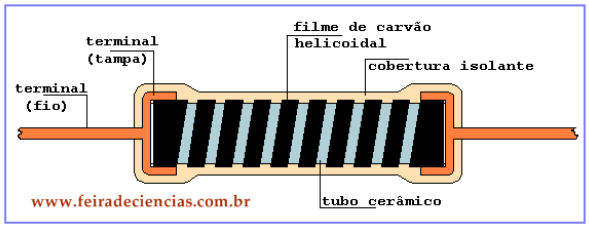

Figura 13. O resistor internamente

5.2 Tipos de resistores

Os resistores podem ser de resistência fixa ou variável.

- Resistor fixo: É um resistor que possibilita um único valor de resistência.
- Resistor variável: Seus valores podem ser ajustados por um movimento mecânico, ou seja, rodando manualmente.

O reostato é um resistor variável com dois terminais, sendo um fixo e o outro deslizante. Geralmente são utilizados com altas correntes.

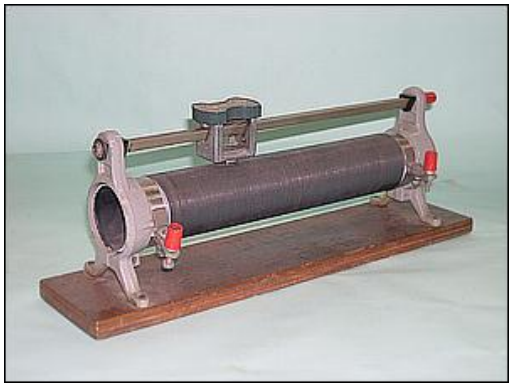

Figura 14. Reostato

O potenciômetro é um tipo de resistor variável comum, sendo comumente utilizado para controlar o volume em amplificadores de áudio. Geralmente, é um resistor de três terminais onde a conexão central é deslizante e manipulável. Se todos os três terminais são usados, ele atua como um divisor de tensão.

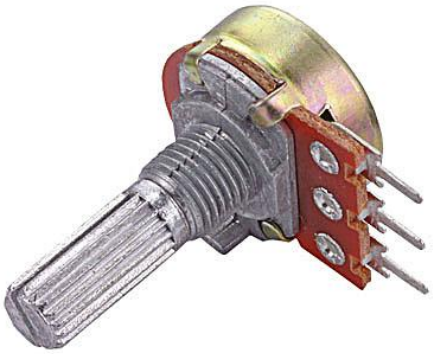

Figura 15. Potenciômetro

Varistor é um tipo especial de resistor que tem dois valores de resistência muito diferentes, um valor muito alto em baixas voltagens. É usado geralmente para proteção contra curtos-circuitos em extensões ou para-raios usados nos postes de ruas, ou como "trava" em circuitos eletromotores. No caso de picos de tensão de maior duração, a alta corrente que circula pelo componente faz com que o dispositivo de proteção, disjuntor ou fusível, desarme, desconectando o circuito da fonte de alimentação.

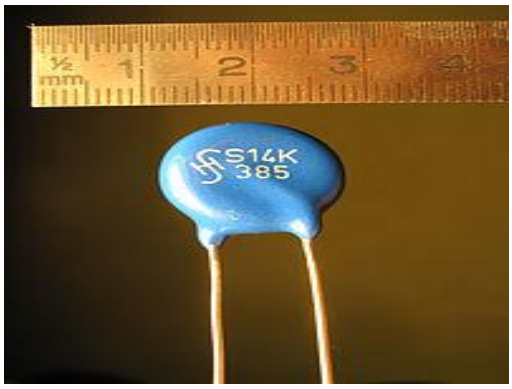

Figura 16. Metal Oxido Varistor 385 volt

5.3 Resistores em série e em paralelo

Em um circuito série constatam-se as seguintes propriedades:

- A **corrente** que passa por todos os componentes é a mesma;
- A soma das **tensões** sobre todos os componentes deve ser igual à tensão total aplicada;
- A resistência total da associação em serie é igual à soma das resistências dos componentes individuais.

Em um circuito paralelo constatam-se as seguintes propriedades:

- Todos os componentes suportam a mesma **tensão** elétrica;
- A soma de todas **correntes** nos componentes individuais deve ser igual à corrente total;
- A resistência total da associação é calculada pelo quociente entre o produto das resistências individuais e a soma delas (CUIDADO: isso vale só para 2 resistores em paralelo)

5.4 Códigos de cores

Por ter um tamanho muito reduzido, é inviável imprimir nos resistores as suas respectivas resistências. Optou-se então pelo código de cores, que consiste em faixas coloridas no corpo do resistor.

As primeiras três faixas servem para indicar o valor nominal de sua resistência e a última faixa, a porcentagem na qual a resistência pode variar seu valor nominal (tolerância).

Para encontrar a resistência de um resistor e sua tolerância, usa-se a equação e a tabela abaixo:

*Restistência = (10*faixa 1 + faixa 2) * 10^(faixa 3) ± % de tolerância*

Obs. 1: Na faixa 3, são permitidos valores somente até 7, o dourado passa a valer -1 e o prateado -2.

Obs. 2: A ausência de uma quarta faixa, indica uma tolerância de 20%.

Cores	Valores			Multiplicadores	Tolerência
	Faixa 1	Faixa 2	Faixa 3		
Prata	-	-	-	0,01	10%
Ouro	-	-	-	0,1	5%
Preto	0	0	0	1	-
Marrom	1	1	1	10	1%
Vermelho	2	2	2	100	2%
Laranja	3	3	3	1000	-
Amarelo	4	4	4	10000	-
Verde	5	5	5	100000	-
Azul	6	6	6	1000000	-
Violeta	7	7	7	-	-
Cinza	8	8	8	-	-
Branco	9	9	9	-	-
Nenhuma	-	-	-	-	20%

Tabela 2. Cores e valores das faixas de resistores

Alguns websites disponibilizam uma página que podemos inserir as cores de um resistor (em sua devida ordem) e ele nos mostra qual a resistência imposta por ele.

Figura 17. Página da web <http://www.areaseg.com/sinais/resistores.html>

6. Ambiente de Desenvolvimento Arduino

O ambiente de desenvolvimento Arduino contém um editor de texto para a escrita de código, uma área de mensagens, um *console* de texto, uma barra de ferramentas com botões para variadas funções, e uma série de menus. Ele se conecta ao *hardware* do Arduino, permitindo o *upload* de programas e a comunicação com eles.

Os programas escritos usando Arduino são chamados *sketches*. Essas *sketches* são escritas no editor de texto, e são salvas com a extensão de "*arquivo*.*ino*". Elas permitem funcionalidades como recortar/colar e procurar/substituir texto. Na área de mensagem, temos retorno de ações como salvar e exportar, além da exibição de erros. O *console* mostra a saída de texto do Arduino, incluindo mensagens de erro completas e outras informações. O canto inferior direito mostra a *board* atual e a portal serial. Os botões da barra de ferramentas permitem que você verifique e faça *upload* de programas, crie, abra e salve *sketches*, e abra o monitor serial.

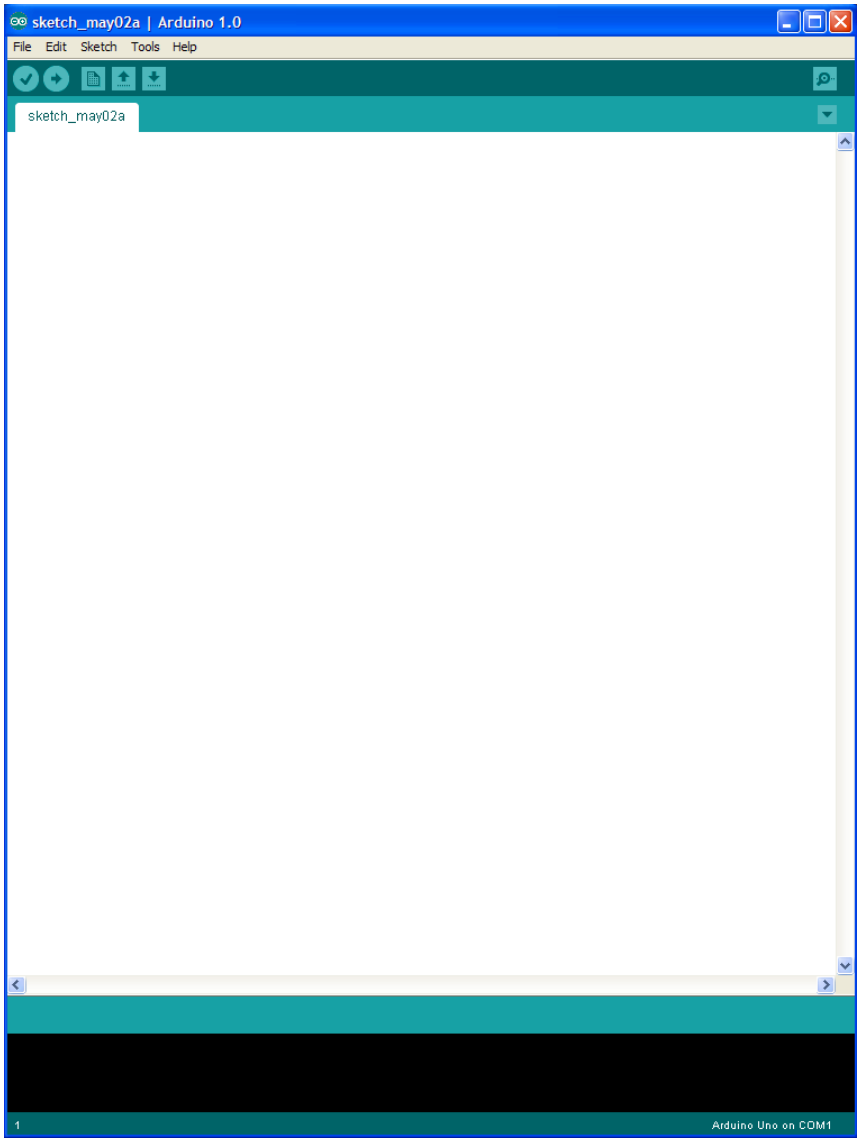

Figura 18: Ambiente de desenvolvimento (IDE) do Arduino.

Principais comandos disponíveis através dos botões:

(a) **Verify**: Verifica se o código contém erros.

(b) **Upload**: Compila o código a faz o upload para o Arduino.

(c) **New**: Cria uma nova sketch.

(d) **Open**: Exibe um menu com todas as sketches no seu sketchbook. Ao clicar em uma, a mesma será aberta na janela atual.

(e) **Save**: Salva sua sketch.

(f) **Serial Monitor**: Abre o monitor serial.

Comandos adicionais são encontrados através dos menus: File, Edit, Sketch, Tools, Help. As funções disponíveis pelos menus File, Edit e Help são semelhantes a outros programas bem conhecidos e, por isso, não iremos detalhá-las aqui.

menu Sketch

- **Verify/Compile** - Verifica se seu código tem erros
- **Import Library** - Adiciona bibliotecas ao seu programa
- **Show sketchfolder** - Abre a pasta onde o programa está salvo
- **Add File...** - Adiciona um arquivo fonte ao programa. O novo arquivo aparece em uma nova aba

menu Tools

- **Auto format** - Formata o código para uma melhor leitura, alinhando as chaves e indentando seu conteúdo.
- **Board** - Seleciona o kit de desenvolvimento onde se deseja realizar o projeto.
- **Serial Port** - Mostra todas as portas seriais que o computador possui.
- **Burn Bootloader** - Permite gravar um *bootloader* no kit de desenvolvimento do Arduino.

7. Programando para o Arduino: Conceitos e Sintaxe da Linguagem de Programação

A plataforma de implementação dos programas em Arduino é baseada nas linguagens C/C++, preservando sua sintaxe na declaração de variáveis, na utilização de operadores, na manipulação de vetores, na conservação de estruturas, bem como é uma linguagem sensível ao caso (*case-sensitive*) Contudo, ao invés de uma função *main( )*, o Arduino necessita de duas funções elementares: *setup( )* e *loop( ).*

7.1 Setup e Loop

Pode-se dizer que todo código criado para o Arduino deve obrigatoriamente possuir duas funções para que o programa funcione corretamente: a função *setup( )* e a função *loop( )*. Essas duas funções não utilizam parâmetros de entrada e são declaradas como void. Não é necessário invocar a função setup() ou a função loop(). Ao compilar um programa para o Arduino, o compilador irá, automaticamente, inserir uma função main que invocará ambas as funções.

setup( )

A função setup é utilizada para inicializar variáveis, configurar o modo dos pinos e incluir bibliotecas. Esta função é executada automaticamente uma única vez, assim que o kit Arduino é ligado ou resetado.

Sintaxe:

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

loop( )

A função loop faz exatamente o que seu nome sugere: entra em *looping* (executa sempre o mesmo bloco de código), permitindo ao seu programa executar as operações que estão dentro desta função. A função *loop( )* deve ser declarada após a função *setup( )*

Sintaxe:

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

7.2 HIGH e LOW

Quando estamos lendo ou escrevendo em um pino digital há apenas dois valores que um pino pode ter: HIGH (alto) e LOW (baixo).

- **HIGH** O significado de HIGH (em referência a um pino) pode variar um pouco dependendo se este pino é uma entrada (INPUT) ou saída (OUTPUT). Quando um pino é configurado como INPUT com a função pinMode, e lido com a função digitalRead, o micro controlador considera como HIGH se a voltagem for de 3 Volts ou mais. Um pino também pode ser configurado como um INPUT com o pinMode, e posteriormente receber um HIGH com um digitalWrite, isto vai "levantar" o resistor interno de 20 KOhms que vai manter a leitura do pino como HIGH a não ser que ela seja alterada para LOW por um circuito externo. Quando um pino é configurado como OUTPUT com o pinMode, e marcado como HIGH com o digitalWrite, ele está a 5 Volts. Neste estado ele pode enviar corrente para, por exemplo, acender um LED que está conectado com um resistor em série ao terra, ou a outro pino configurado como OUTPUT e marcado como LOW.

- **LOW** O significado de LOW também pode variar dependendo do pino ser marcado como INPUT ou OUTPUT. Quando um pino é configurado como INPUT com a função pinMode, e lido com a função digitalRead, o micro controlador considera como LOW se a voltagem for de 2 Volts ou menos. Quando um pino é configurado como OUTPUT com a função pinMode, e marcado como LOW com a função digitalWrite, ele está a 0 Volts. Neste estado ele pode "drenar" corrente para, por exemplo, acender um LED que está conectado com um resistor em série ao +5 Volts, ou a outro pino configurado como OUTPUT e marcado como HIGH.

7.3 INPUT e OUTPUT

Pinos digitais podem ser tanto de INPUT como de OUTPUT. Mudar um pino de INPUT para OUTPUT com pinMode() muda drasticamente o seu comportamento elétrico.

- **INPUT** Os pinos do Arduino (Atmega) configurados como INPUT com a função pinMode() estão em um estado de alta impedância. Pinos de entrada são válidos para ler um sensor, mas não para energizar um LED.

- **OUTPUT** Pinos configurados como OUTPUT com a função pinMode() estão em um estado de baixa impedância. Isto significa que eles podem fornecer grandes quantidades de corrente para outros circuitos. Os pinos do Atmega podem fornecer corrente positiva ou drenar corrente negativa até 40 mA (milliamperes)

de/para outros dispositivos ou circuitos. Isto faz com que eles sejam úteis para energizar um LED mas disfuncionais para a leitura de sensores. Pinos configurados como OUTPUT também podem ser danificados ou destruídos por curto-circuitos com o terra ou com outros pontos de 5 Volts. A quantidade de corrente fornecida por um pino do Atmega também não é suficiente para ativar muitos relês e motores e, neste caso, algum circuito de interface será necessário.

7.4 Funções

7.4.1 Funções de Entrada e Saída Digital

pinMode( ) Configura o pino especificado para que se comporte ou como uma entrada ou uma saída. Deve-se informar o número do pino que se deseja configurar e em seguida, se o pino será determinado como uma entrada(INPUT) ou uma saída(OUTPUT).

Sintaxe:

pinMode(pino, modo);

digitalWrite( ) Escreve um valor HIGH ou LOW em um pino digital. Se o pino foi configurado como uma saída, sua voltagem será determinada ao valor correspondente: 5V para HIGH e 0V para LOW. Se o pino está configurado como uma entrada, HIGH levantará o resistor interno de 20KOhms e LOW rebaixará o resistor.

Sintaxe:

digitalWrite(pino, valor);

digitalRead( ) Lê o valor de um pino digital especificado e retorna um valor HIGH ou LOW.

Sintaxe:

Int digitalRead(pino);

```
/* Exemplo de função sobre de Entrada e Saída Digital */
int ledPin = 13; // LED conectado ao pino digital 13
int bot = 7; // botão conectado ao pino digital 7
int val = 0; // variável para armazenar o valor lido
void setup()
{
      pinMode(ledPin, OUTPUT); // seta o pino digital 13 como uma saída
      pinMode(bot, INPUT); // seta o pino digital 7 como uma entrada
}
void loop()
{
      val = digitalRead(bot); // lê o pino de entrada
      digitalWrite(ledPin, val); // acende o LED de acordo com o botão
```

```
}
```
Essa função transfere para o pino 13, o valor lido no pino 7 que é uma entrada.

7.4.2 Funções de Entrada e Saída Analógica

analogWrite( ) - PWM Pulse Width Modulation ou Modulação por Largura de Pulso (MLP) é um método para obter resultados analógicos com meios digitais. Essa função basicamente escreve um sinal analógico. Ela pode ser usada para acender um LED variando o seu brilho, ou girar um motor com velocidade variável. Depois de realizar um *analogWrite(*), o pino gera uma onda quadrada estável com o ciclo de rendimento especificado até que um *analogWrite( )*, um *digitalRead( )* ou um *digitalWrite( )* seja usado no mesmo pino. Em kits Arduino com o chip ATmega168, esta função está disponível nos pinos 3,5,6,9,10 e 11. Kits Arduino mais antigos com um ATmega8 suportam o *analogWrite( )* apenas nos pinos 9,10 e 11. As saídas PWM geradas pelos pinos 5 e 6 terão rendimento de ciclo acima do esperado. Isto se deve às interações com as funções *millis( )* e *delay( )*, que compartilham o mesmo temporizador interno usado para gerar as saídas PWM. Para usar esta função deve-se informar o pino ao qual deseja escrever e em seguida informar um valor entre 0 (pino sempre desligado) e 255 (pino sempre ligado).

Sintaxe:

analogWrite(pino, valor);

analogRead( ) Lê o valor de um pino analógico especificado. O kit Arduino contém um conversor analógico-digital de 10 bits com 6 canais. Com isto ele pode mapear voltagens de entrada entre 0 e 5 Volts para valores inteiros entre 0 e 1023. Isto permite uma resolução entre leituras de 5 Volts / 1024 unidades ou 0,0049 Volts (4.9 mV) por unidade

Sintaxe:

int analogRead(pino);

```
/* Exemplo de função sobre Entrada e Saída Analógica */
int ledPin = 9; // LED conectado ao pino digital 9
int analogPin = 3; // potenciômetro conectado ao pino analógico 3
int val = 0; // variável para armazenar o valor lido
void setup()
{
pinMode(ledPin, OUTPUT); // pré-determina o pino como saída
}
void loop()
{
val = analogRead(analogPin); // lê o pino de entrada
analogWrite(ledPin, val/4); //
}
```

Torna o brilho de um LED proporcional ao valor lido em um potenciômetro.

7.4.3 Tempo

millis( ) Retorna o número de milisegundos desde que o kit Arduino começou a executar o programa. Este número extrapolará (voltará a zero) depois de aproximadamente 50 dias.

Sintaxe:

```
unsigned long tempo;
void loop
{
.
:
tempo = millis()
}
```

delay( ) Suspende a execução do programa pelo tempo (em milisegundos) especificado. Em um segundo há 1.000 milisegundos.

Sintaxe:

```
delay(tempo);
```

7.4.4 Comunicação serial

Serial.begin( ) Ajusta o taxa de transferência em bits por segundo para uma transmissão de dados pelo padrão serial. Para comunicação com um computador use uma destas taxas: 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 57600, 115200. Pode-se, entretanto, especificar outras velocidades por exemplo, para comunicação através dos pinos 0 e 1 com um componente que requer uma taxa específica.

Sintaxe:

```
Serial.begin(taxa);
```

```
/* Este programa mostra uma aplicação das funções millis( ) e delay( )  e Comunicação Serial */
unsigned long time;
void setup()
{
      Int Time = 0;       // variável
      Serial.begin(9600); // ajusta a taxa de transferência
}
void loop()
{
      Serial.print("O tempo eh: ");  //imprime O tempo eh: na tela
      Time = millis(); // variável Time guarda o valor retornado por millis()
      Serial.println(time); //imprime o tempo desde que o programa começou
      delay(1000); // espera 1 segundo
}
```

REFERÊNCIAS BIBLIOGRÁFICAS

Ambiente de Desenvolvimento Arduino
Descrição do Funcionamento de uma Protoboard
Programando para o Arduino: Conceitos e Sintaxe da Linguagem de Programação

http://lscad.facom.ufms.br/wiki/index.php/Downloads
http://arduino.cc/
http://en.wikipedia.org/wiki/Breadboard

Responsáveis: Riccieli Minakawa, Luiz Eduardo Tiosso, Wellington Oliveira dos Santos, Lucas Tsutsui da Silva

Introdução
Modelos de Arduino
Resistores

http://www.sabereletronica.com.br/secoes/leitura/1307
http://arduino.cc/en/Guide/Environment
http://www.garotascpbr.com.br/2011/10/04/introducao-ao-arduino-primeira-parte/
http://projeto39.wordpress.com/o-arduino/
http://www.arduino.cc/
http://pt.wikipedia.org/wiki/Arduino
http://www.labdegaragem.com.br/wiki/index.php?title=Qual_Arduino_%C3%A9_o_certo_para_mim%3F
http://4.bp.blogspot.com/_pGsJFsc5Buc/TNePByaIHGI/AAAAAAAAA1w/q2DtWB-DynU/s1600/Arduino+Uno+Unboxing.JPG
https://sites.google.com/site/marceloboeirajr/tutoriais/eletronica-senai/arduino/hardware/1-3-tipos-de-arduino
http://www.electronica-pt.com/
http://leandrocodorna.vilabol.uol.com.br/segunda.html
http://pt.wikipedia.org/wiki/Resistor
http://www.feiradeciencias.com.br/
http://www.sabereletronica.com.br/
http://brasilrobotics.blogspot.com.br/2011/02/pisca-led-o-primeiro-exemplo-para.html

Responsáveis: Jefferson Rios, Rodrigo Santiago, Angelo Maggioni e Paulo Tiene

Arduino Mega

http://lscad.facom.ufms.br/wiki/index.php/Downloads

Responsáveis: Lucas Rodrigues, Eliseu Sartori, Bruno Gouveia