

1. INTRODUÇÃO

O objetivo deste manual é apresentar, de uma forma simples e rápida, o básico de programação, sendo assim iremos focar apenas nos conceitos que são importantes para Arduino e sistemas embarcados em geral.

COMPUTADOR

Um computador é, de forma simplificada, uma máquina que processa instruções. Essas instruções são processadas no "cérebro" do computador, que se chama microprocessador. Todo computador possui pelo menos um microprocessador. O Arduino, por exemplo, nada mais é do que um computador muito pequeno, e ele utiliza um microprocessador do modelo ATmega. Alguns microprocessadores, como o ATmega, também são chamados de microcontroladores.

PROGRAMA DE COMPUTADOR

Um programa de computador, ou software, é uma sequência de instruções que são enviadas para o computador. Cada tipo de microprocessador (cérebro) entende um conjunto de instruções diferente, ou seja, o seu próprio "idioma". Também chamamos esse idioma de linguagem de máquina.

As linguagens de máquina são, no fundo, as únicas linguagens que os computadores conseguem entender, só que elas são muito difíceis para os seres humanos entenderem. É por isso nós usamos uma coisa chamada linguagem de programação. No caso de sistemas como o Arduino (os chamados sistemas embarcados), o software que roda no microprocessador é também chamado de firmware.

LINGUAGEM DE PROGRAMAÇÃO

Nós seres humanos precisamos converter as nossas idéias para uma forma que os computadores consigam processar, ou seja, a linguagem de máquina. Os computadores de hoje (ainda) não conseguem entender a linguagem natural que nós usamos no dia a dia, então precisamos de um outro "idioma" especial para instruir o computador a fazer as tarefas que desejamos. Esse "idioma" é uma linguagem de programação, e na verdade existem muitas delas.

Essas linguagens de programação também são chamadas de linguagens de programação de alto nível. A linguagem de programação utilizada no Arduino é a linguagem C++ (com pequenas modificações), que é uma linguagem muito tradicional e conhecida.

Para converter um programa escrito em uma linguagem de alto nível para linguagem de máquina, nós utilizamos uma coisa chamada compilador. A ação de converter um programa para linguagem de máquina é chamada compilar. Para compilar um programa, normalmente se utiliza um ambiente de desenvolvimento (ou IDE, do inglês Integrated Development Environment), que é um aplicativo de computador que possui um compilador integrado, onde você pode escrever o seu programa e compilá-lo. No caso do Arduino, esse ambiente de desenvolvimento é o Arduino IDE.

O texto contendo o programa em uma linguagem de programação de alto nível também é conhecido como o código fonte do programa.

ALGORITMO (PROGRAMA)

Um algoritmo, ou simplesmente programa, é uma forma de dizer para um computador o que ele deve fazer, de uma forma que nós humanos conseguimos entender facilmente. Os algoritmos normalmente são escritos em linguagens de programação de alto nível. Isso se aplica a praticamente qualquer computador, inclusive o Arduino, onde um algoritmo também é conhecido como sketch. Para simplificar, a partir de agora nós vamos nos referir aos algoritmos, programas ou sketches simplesmente como "programas". Um programa é composto de uma seqüência de comandos, normalmente escritos em um arquivo de texto.

VARIÁVEL

Uma variável é um recurso utilizado para armazenar dados em um programa de computador. Todo computador possui algum tipo de memória, e uma variável representa uma região da memória usada para armazenar uma determinada informação. Essa informação pode ser, por exemplo, um número, um caráter ou uma seqüência de texto. Para podermos usar uma variável em um programa Arduino nós precisamos fazer uma declaração de variável, como por exemplo:

INT LED;

Nesse caso estamos declarando uma variável do tipo **INT** chamada **LED**. Em seguida nós falaremos mais sobre o tipo de dado de uma variável.

TIPO DE DADO

O tipo de dado de uma variável significa, como o próprio nome diz, o tipo de informação que se pode armazenar naquela variável. Em muitas linguagens de programação, como C++, é obrigatório definir o tipo de dado no momento da declaração da variável, como vimos na declaração da variável led acima. No caso dos Arduino que usam processador ATmega, os tipos mais comuns de dados que utilizamos são:

- boolean: valor verdadeiro (true) ou falso (false)
- char: um caracter
- byte: um byte, ou seqüência de 8 bits
- int: número inteiro de 16 bits com sinal (-32768 a 32767)
- unsigned int: número inteiro de 16 bits sem sinal (0 a 65535)
- long: número inteiro de 16 bits com sinal (-2147483648 a 2147483647)
- unsigned long: número inteiro de 16 bits sem sinal (0 a 4294967295)
- float: número real de precisão simples (ponto flutuante)
- double: número real de precisão dupla (ponto flutuante)
- string: seqüência de caracteres
- void: tipo vazio (não tem tipo)

ATRIBUIÇÃO

Atribuir um valor a uma variável significa armazenar o valor nela para usar posteriormente. O comando de atribuição em C++ é o =. Para atribuirmos o valor 13 à variável led que criamos acima, fazemos assim:

```
LED = 13;
```

Quando se armazena um valor em uma variável logo na sua inicialização, chamamos isso de inicialização de variável. Assim, no nosso programa de exemplo temos:

```
INT LED = 13;
```

O objetivo dessa linha de código é dizer que o pino 13 do Arduino será utilizado para acender o LED, e armazenar essa informação para usar depois ao longo do programa.

Os valores fixos usados no programa, como o valor 13 acima, são chamados de constantes, pois, diferentemente das variáveis, o seu valor não muda.

OPERADOR

Um operador é um conjunto de um ou mais caracteres que serve para operar sobre uma ou mais variáveis ou constantes. Um exemplo muito simples de operador é o operador de adição, o +. Digamos que queremos somar dois números e atribuir a uma variável x. Para isso, fazemos o seguinte:

x = 2 + 3;

Após executar o comando acima, a variável x irá conter o valor 5.

Cada linguagem de programação possui um conjunto de operadores diferente. Alguns dos operadores mais comuns na linguagem C++ são:

=> Operadores aritméticos:

=> +: adição ("mais")

=> -: subtração ("menos")

=> *: multiplicação ("vezes")

=> /: divisão ("dividido por")

=> Operadores lógicos:

=> &&: conjunção ("e")

=> ||: disjunção ("ou")

=> ==: igualdade ("igual a")

=> !=: desigualdade ("diferente de")

=> !: negação ("não")

=> >: "maior que"

=> <: "menor que"

=> >=: "maior ou igual a"

=> <=: "menor ou igual a"

=> Operadores de atribuição:

=> =: atribui um valor a uma variável, como vimos acima.

Ao longo do desenvolvimento dos seus projetos, aos poucos você se familiarizará com todos esses operadores.

FUNÇÃO

Uma função é, em linhas gerais, uma seqüência de comandos que pode ser reutilizada várias vezes ao longo de um programa. Para criar uma função e dizer o que ela faz, nós precisamos fazer uma declaração de função. Veja como uma função é declarada no nosso programa de exemplo:

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

Aqui estamos declarando uma função com o nome `setup()`. O que ela faz é executar os comandos de uma outra função `pinMode()`. A ação de executar os comandos de função previamente declarada é denominada chamada de função. Nós não precisamos declarar a função `pinMode()` porque ela já é declarada automaticamente no caso do Arduino.

CHAMADA DE FUNÇÃO

Chamar uma função significa executar os comandos que foram definidos na sua declaração. Uma vez declarada, uma função pode ser chamada várias vezes no mesmo programa para que seus comandos sejam executados novamente. Para chamarmos a nossa função `setup()`, por exemplo, nós usariammos o seguinte comando:

```
setup();
```

No entanto, no caso do Arduino, nós não precisamos chamar a função `setup()`, porque ela é chamada automaticamente. Quando compilamos um programa no Arduino IDE, ele chama a função `setup()` uma vez e depois chama a função `loop()` repetidamente até que o Arduino seja desligado ou reiniciado.

VALOR DE RETORNO

A palavra chave que vem antes do nome da função na declaração define o tipo do valor de retorno da função. Toda vez que uma função é chamada, ela é executada e devolve ou retorna um determinado valor - esse é o valor de retorno, ou simplesmente retorno da função. O valor de retorno precisa ter um tipo, que pode ser qualquer um dos tipos de dados citados anteriormente. No caso da nossa função `setup()`, o tipo de retorno é `void`, o que significa que a função não retorna nada.

Para exemplificar, vamos criar uma função que retorna alguma coisa, por exemplo um número inteiro. Para retornar um valor, nós utilizamos o comando `return`:

```
int f() {  
    return 1;  
}
```

Quando chamada, a função `f()` acima retorna sempre o valor 1. Você pode usar o valor de retorno de uma função para atribuí-lo a uma variável. Por exemplo:

```
x = f();
```

Após declarar a função `f()` e chamar o comando de atribuição acima, a variável `x` irá conter o valor 1.

PARÂMETROS

Um outro recurso importante de uma função são os parâmetros. Eles servem para enviar algum dado para a função quando ela é chamada. Vamos criar por exemplo uma função que soma dois números.

```
int soma(int a, int b) {  
    return a + b;  
}
```

Aqui acabamos definir uma função chamada `soma()`, que aceita dois números inteiros como parâmetros. Nos precisamos dar um nome para esses parâmetros, e nesse caso escolhemos `a` e `b`. Esses parâmetros funcionam como variável que você pode usar dentro da função. Sempre que chamarmos a função `soma()`, precisamos fornecer esses dois números. O comando `return a + b;` simplesmente retorna a função com a soma dos dois números. Vamos então somar 2 + 3 e atribuir o resultado para uma variável `x`:

```
x = soma(2, 3);
```

Após a chamada acima, a variável `x` irá conter o valor 5.

COMENTÁRIOS

Um comentário é um trecho de texto no seu programa que serve apenas para explicar (documentar) o código, sem executar nenhum tipo de comando no programa. Muitas vezes, os comentários são usados também para desabilitar comandos no código. Nesse caso, dizemos que o código foi comentado.

Na linguagem C++, um comentário pode ser escrito de duas formas:

- Comentário de linha: inicia-se com os caracteres //, tornando todo o resto da linha atual um comentário.
- Comentário de bloco: inicia-se com os caracteres /* e termina com os caracteres */. Todo o texto entre o início e o término se torna um comentário, podendo ser composto de várias linhas.

Para facilitar a visualização, os ambientes de desenvolvimento geralmente mostram os comentários em uma cor diferente. No caso do Arduino IDE, por exemplo, os comentários são exibidos na cor cinza. Vamos então explicar o que o programa de exemplo faz, inserindo nele vários comentários explicativos:

```
/*
Programação para Arduino - Primeiros Passos
Programa de exemplo: Blink
*/



/*
Declaração da variável "led"
Indica que o LED está conectado no pino digital 13 do Arduino (D13).
*/
int led = 13;

/*
Declaração da função setup()
Esta função é chamada apenas uma vez, quando o Arduino é ligado ou
reiniciado.
*/
void setup() {

    // Chama a função pinMode() que configura um pino como entrada ou saída
    pinMode(led, OUTPUT); // Configura o pino do LED como saída
}
```

```
/*
```

Declaração da função loop()

Após a função setup() ser chamada, a função loop() é chamada repetidamente até

o Arduino ser desligado.

```
*/
```

```
void loop() {
```

```
    // Todas as linhas a seguir são chamadas de função com passagem de parâmetros
```

```
    // As funções são executadas em seqüência para fazer o LED acender e apagar
```

```
        digitalWrite(led, HIGH); // Atribui nível lógico alto ao pino do LED, acendendo-o
```

```
        delay(1000);           // Espera 1000 milissegundos (um segundo)
```

```
        digitalWrite(led, LOW); // Atribui nível lógico baixo ao pino do LED, apagando-o
```

```
        delay(1000);           // Espera 1000 milissegundos (um segundo)
```

```
    // Após terminar a função loop(), ela é executada novamente repetidas vezes,
```

```
    // e assim o LED continua piscando.
```

```
}
```

ESTRUTURAS DE CONTROLE

Estruturas de controle são blocos de instruções que alteram o fluxo de execução do código de um programa. Com elas é possível fazer coisas como executar comandos diferentes de acordo com uma condição ou repetir uma série de comandos várias vezes, por exemplo.

A seguir nós veremos algumas das estruturas de controle mais comuns usadas nas linguagens de programação em geral. Vamos também modificar o nosso programa de teste para exemplificar melhor como essas estruturas funcionam.

WHILE

O while é uma estrutura que executa um conjunto de comandos repetidas vezes enquanto uma determinada condição for verdadeira. While em inglês quer dizer "enquanto", e pronuncia-se "uái-ou". Ele segue o seguinte formato:

```
while(condição) {  
    ...  
}
```

Vamos então fazer uma modificação no nosso programa para exemplificar melhor como o while funciona. O nosso objetivo agora é fazer o LED piscar três vezes, depois esperar cinco segundos, piscar mais três vezes e assim por diante. Nós vamos mudar o conteúdo da função loop() para o seguinte:

```
// Variável para contar o número de vezes que o LED piscou  
  
int i = 0;  
  
// Piscar o LED três vezes  
while(i < 3) {  
    digitalWrite(led, HIGH); // Atribui nível lógico alto ao pino do LED,  
    acendendo-o  
  
    delay(1000); // Espera 1000 milissegundos (um segundo)  
  
    digitalWrite(led, LOW); // Atribui nível lógico baixo ao pino do LED,  
    apagando-o  
  
    delay(1000); // Espera 1000 milissegundos (um segundo)  
  
    i = i + 1; // Aumenta o número de vezes que o LED piscou  
}
```

```
delay(5000);           // Espera 5 segundos para piscar o LED de novo
```

Primeiro nós declaramos uma variável *i*. Essa variável vai contar quantas vezes o LED já piscou desde o início do programa ou desde a última pausa de cinco segundos. Nós vamos inicializar essa variável com zero porque no início da função *loop()* o LED ainda não piscou nenhuma vez sob essas condições.

Em seguida nós inserimos o comando *while*, que deve ser seguido de uma condição definida entre parênteses. Enquanto essa condição for verdadeira, todo o bloco de comandos entre os caracteres { e } é executado repetidamente. No caso do nosso programa, enquanto o número de "piscadas" do LED (representado pela variável *i*) for menor do que três, nós continuamos a executar os comandos que fazem o LED piscar. Isso é representado pela expressão *i < 3* dentro dos parênteses.

Entre os caracteres { e } nós colocamos o código que faz o LED piscar, como anteriormente, mas não podemos nos esquecer de somar 1 à variável que conta o número de "piscadas". Isso é feito na seguinte linha de código:

```
i = i + 1;           // Aumenta o número de vezes que o LED piscou
```

Veja que após executar todos os comandos entre { e }, sempre teremos na variável *i* o número de vezes que o LED piscou desde o início da função *loop()*. Vamos percorrer a seqüência de passos executada cada vez que a função *loop()* é chamada:

1. Atribuímos 0 à variável *i*: o LED ainda não piscou nenhuma vez.
2. Comparamos se *i < 3*: como 0 é menor do que 3, executamos os comandos entre{ e }:
 1. Executamos os comandos para acender e apagar o LED.
 2. Somamos 1 à variável *i*, tornando-a 1: sabemos que o LED piscou uma vez.
 3. Voltamos ao início do while e comparamos se *i < 3*: como 1 é menor do que 3, executamos os comandos entre { e } novamente:
 1. Executamos os comandos para acender e apagar o LED.
 2. Somamos 1 à variável *i*, tornando-a 2: sabemos que o LED piscou duas vezes.
 4. Voltamos ao início do while e comparamos se *i < 3*: como 2 é menor do que 3, executamos os comandos entre { e } novamente:
 1. Executamos os comandos para acender e apagar o LED.

2. Somamos 1 à variável i, tornando-a 3: sabemos que o LED piscou três vezes.

5. Voltamos ao início do while e comparamos se $i < 3$: como 3 não é menor do que 3, não executamos mais os comandos entre { e } e prosseguimos à próxima instrução.

6. Esperamos cinco segundos por meio da chamada `delay(5000)`.

Após esses passos, chegamos ao final da função `loop()`, e como já sabemos, ela é chamada novamente pelo sistema do Arduino. Isso reinicia o ciclo, executando os passos acima indefinidamente.

Rode o programa modificado com as instruções acima no seu Arduino e tente variar o número de "piscadas" e o número

FOR

Agora que nós já aprendemos o comando `while`, fica muito fácil aprender o comando `for`, pois ele é quase a mesma coisa. Vamos modificar o conteúdo da função `loop()` como fizemos acima, porém usando o `for` no lugar do `while`:

```
// Variável para contar o número de vezes que o LED piscou  
int i;  
  
// Pisca o LED três vezes  
for(i = 0; i < 3; i++) {  
    digitalWrite(led, HIGH); // Atribui nível lógico alto ao pino do LED,  
    // acendendo-o  
    delay(1000); // Espera 1000 milissegundos (um segundo)  
    digitalWrite(led, LOW); // Atribui nível lógico baixo ao pino do LED,  
    // apagando-o  
    delay(1000); // Espera 1000 milissegundos (um segundo)  
}  
delay(5000); // Espera 5 segundos para piscar o LED de novo
```

A primeira modificação que fizemos foi declarar a variável i sem inicializá-la com o valor 0. Nós podemos fazer isso porque o comando for fará isso para a gente. Ele segue o seguinte formato:

```
for(inicialização; condição; finalização) {  
    ...  
}
```

IF

É uma das estruturas mais básicas de programação em geral. If significa "se" em inglês, e é exatamente isso que ele faz: ele verifica uma expressão e, apenas se ela for verdadeira, executa um conjunto de comandos. Em linguagem natural, ele executa uma lógica do tipo: "se isso for verdadeiro, então faça aquilo"

Para ilustrar, vamos modificar o nosso programa de exemplo para que ele faça a mesma coisa que fizemos com o while e o for acima, porém vamos fazer isso usando um if, que segue o seguinte formato:

```
if(condição) {  
    ...  
}
```

A lógica é muito simples: sempre que a condição for verdadeira, os comandos entre { e } são executados, caso contrário o programa prossegue sem executá-los. Vamos ver então como fica a função loop():

```
// Variável para contar o número de vezes que o LED piscou  
int i = 0;  
  
void loop() {  
    digitalWrite(led, HIGH); // Atribui nível lógico alto ao pino do LED, acendendo-o  
    delay(1000);           // Espera 1000 milissegundos (um segundo)  
    digitalWrite(led, LOW); // Atribui nível lógico baixo ao pino do LED, apagando-o  
    delay(1000);           // Espera 1000 milissegundos (um segundo)
```

```

i++;           // Incrementa o número de "piscadas"

if(i == 3) {

    delay(5000);      // Espera 5 segundos para piscar o LED de novo

    i = 0;           // Reinicia o contador de número de "piscadas"

}

}

```

Aqui a lógica é um pouco diferente: nós vamos manter a função loop() piscando o LED como no programa original, porém vamos inserir uma espera adicional de 5 segundos após cada 3 piscadas. Para isso, criamos uma variável *i* fora da função loop(); ela precisa ser declarada de fora da função para poder reter o seu valor entre cada execução da função loop(). Chamamos isso de variável global. Quando a variável é declarada dentro do corpo da função, ela não retém o valor entre cada execução, sendo reiniciada a cada vez que a função é re-executada. Chamamos isso de variável local.

Nós usaremos então essa variável global *i* para contar, novamente, o número de vezes que o LED acendeu e apagou. Na declaração da variável, nós a inicializamos com o valor 0 para indicar que o LED não acendeu nenhuma vez ainda. A função loop() então começa a ser executada, acendendo e apagando o LED. Para contar o número de vezes que o LED piscou, nós adicionamos a seguinte linha de código:

```
i++;           // Incrementa o número de "piscadas"
```

Em seguida utilizamos o if para verificar se acabamos de acender o LED pela terceira vez. Para isso, usamos a expressão *i == 3* na condição do if. Se essa expressão for verdadeira, isso que dizer que o LED já acendeu 3 vezes, então inserimos uma pausa adicional de 5 segundos com a chamada *delay(5000)* e reiniciamos a contagem do número de "piscadas" novamente com o seguinte comando:

```
i = 0;           // Reinicia o contador de número de "piscadas"
```

A partir daí a função loop() continua sendo chamada e o ciclo se inicia novamente.

IF-ELSE

O if-else, também conhecido como if-then-else, pode ser visto como uma extensão do comando if. Else em inglês significa "caso contrário", e ele faz exatamente o que o nome diz: "se isso for verdadeiro, então faça aquilo, caso contrário, faça outra coisa". Ele segue o seguinte formato:

```
if(condição) {  
    ...  
} else {  
    ...  
}
```

Para exemplificar, vamos usar o programa do for que mostramos acima, mas vamos dessa vez fazer o LED acender e apagar quatro vezes antes de dar uma pausa de cinco segundos. Depois vamos fazer com que na terceira de cada uma dessas quatro "piscadas", o LED acenda por um período mais curto. Dentro da função loop(), teremos o seguinte:

```
// Variável para contar o número de vezes que o LED piscou  
  
int i;  
  
// Pisca o LED três vezes  
  
for(i = 0; i < 3; i++) {  
    if(i == 2) {  
        digitalWrite(led, HIGH); // Atribui nível lógico alto ao pino do LED,  
        acendendo-o  
        delay(200); // Espera 200 milissegundos (um segundo)  
        digitalWrite(led, LOW); // Atribui nível lógico baixo ao pino do LED,  
        apagando-o  
        delay(1800); // Espera 1800 milissegundos (um segundo)  
    } else {  
        digitalWrite(led, HIGH); // Atribui nível lógico alto ao pino do LED,  
        acendendo-o  
        delay(1000); // Espera 1000 milissegundos (um segundo)
```

```
    digitalWrite(led, LOW); // Atribui nível lógico baixo ao pino do LED,  
    apagando-o  
  
    delay(1000); // Espera 1000 milissegundos (um segundo)  
}  
  
}  
  
delay(5000); // Espera 5 segundos para piscar o LED de novo
```

BIBLIOTECAS

As coisas que aprendemos nas seções anteriores são importantes para implementar a lógica do seu programa no Arduino, mas normalmente você vai querer fazer mais coisas além de apenas acender um LED. Quando se faz tarefas mais complexas ou se utiliza algum outro circuito conectado ao seu Arduino, um recurso muito importante são as bibliotecas.

Uma biblioteca é basicamente composta de código fonte adicional que você adiciona ao seu projeto por meio do comando `include`. Vejamos como adicionar, por exemplo, uma biblioteca para controle de um display de cristal líquido (LCD):

#INCLUDE <LIQUIDCRYSTAL.H>

Uma biblioteca do Arduino se apresenta normalmente como uma ou mais classes que possuem funções, os métodos, para acionar dispositivos, configurá-los ou executar alguma outra tarefa. Continuando com o exemplo do display de cristal líquido, para usá-lo no seu programa, primeiro é preciso inicializá-lo. O que fazemos nesse caso é criar um objeto para acessar o LCD (teoricamente isso se chama instanciar um objeto). Vejamos como isso é feito:

```
LIQUIDCRYSTAL lcd(12, 11, 5, 4, 3, 2);
```

Quando fazemos isso, `lcd` se torna um objeto da classe `LiquidCrystal`. Isso é o equivalente a criar uma variável do tipo `LiquidCrystal`. Os parâmetros que são passados entre parênteses servem para inicializar a configuração desse objeto, e nesse caso correspondem aos números dos pinos que foram utilizados para conectar o LCD ao Arduino.

Quase sempre as bibliotecas de Arduino possuem um método `begin()`, que serve para fazer a configuração inicial do dispositivo que está sendo controlado. Para chamar a função `begin()` do objeto `lcd` que criamos, fazemos o seguinte:

```
lcd.begin(16, 2);
```

Normalmente esse método begin() é chamado de dentro da função setup(), ou seja, durante a inicialização do programa. Os parâmetros do método begin(), nesse caso, correspondem ao número de colunas e o número de linhas do LCD, respectivamente.

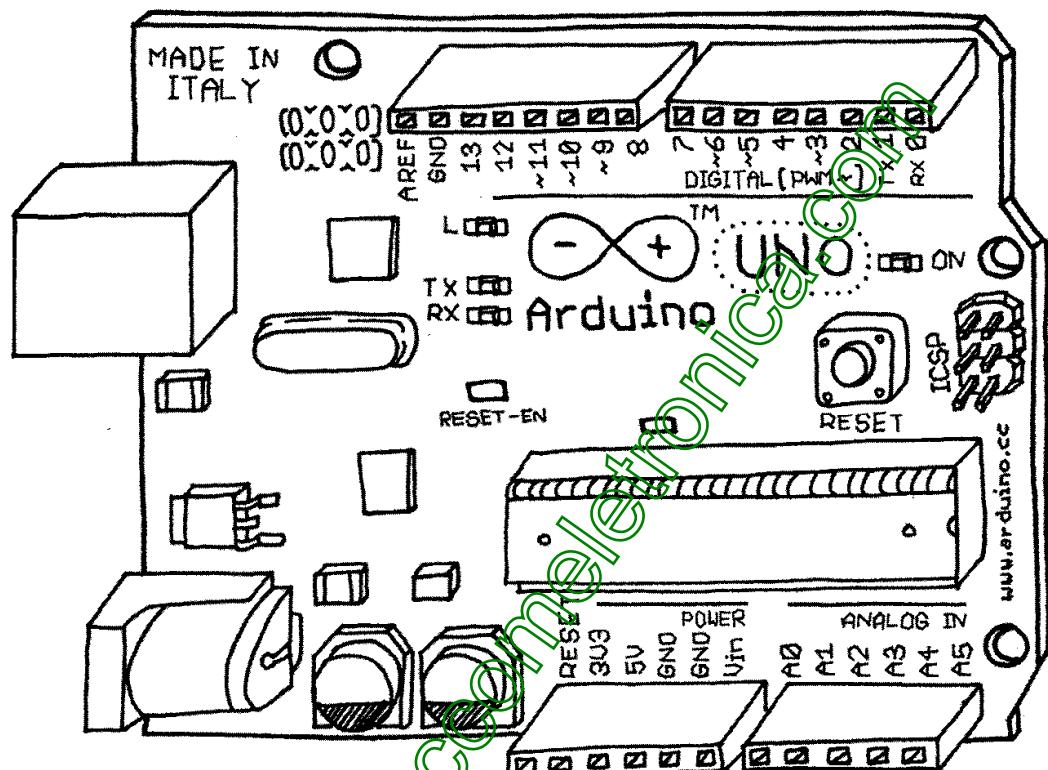
Feitos esses passos, já podemos escrever texto no LCD. Fazemos isso usando o método print() do objeto lcd, sempre que precisarmos ao longo do programa:

```
lcd.print("Oi!");
```

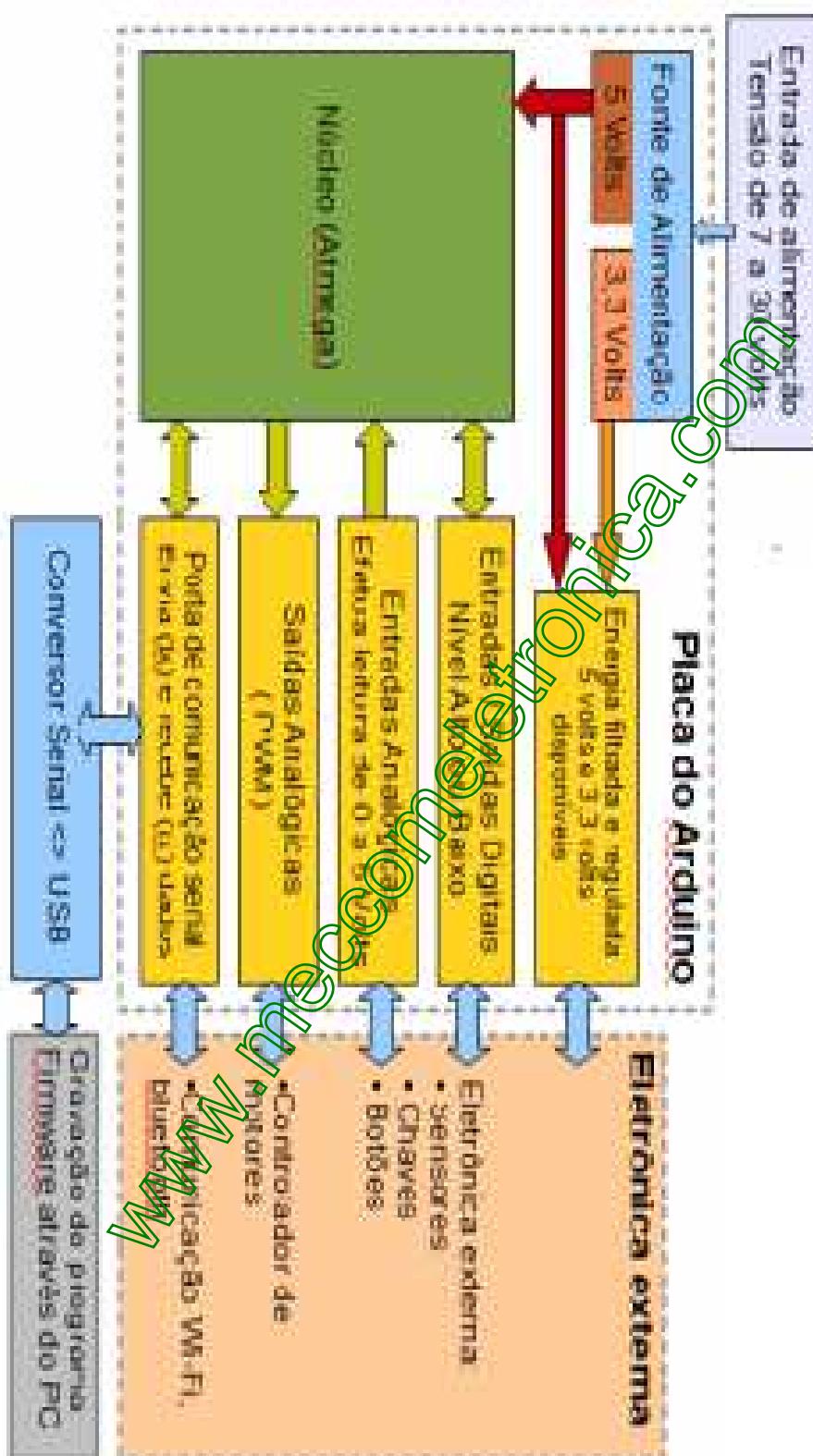
O método print() é apenas um dos vários métodos disponíveis na biblioteca LiquidCrystal. Para saber todos os métodos fornecidos por uma determinada biblioteca, é preciso consultar a documentação fornecida com ela. Este é o processo básico de utilização de bibliotecas no Arduino.

O QUE É O ARDUINO?

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

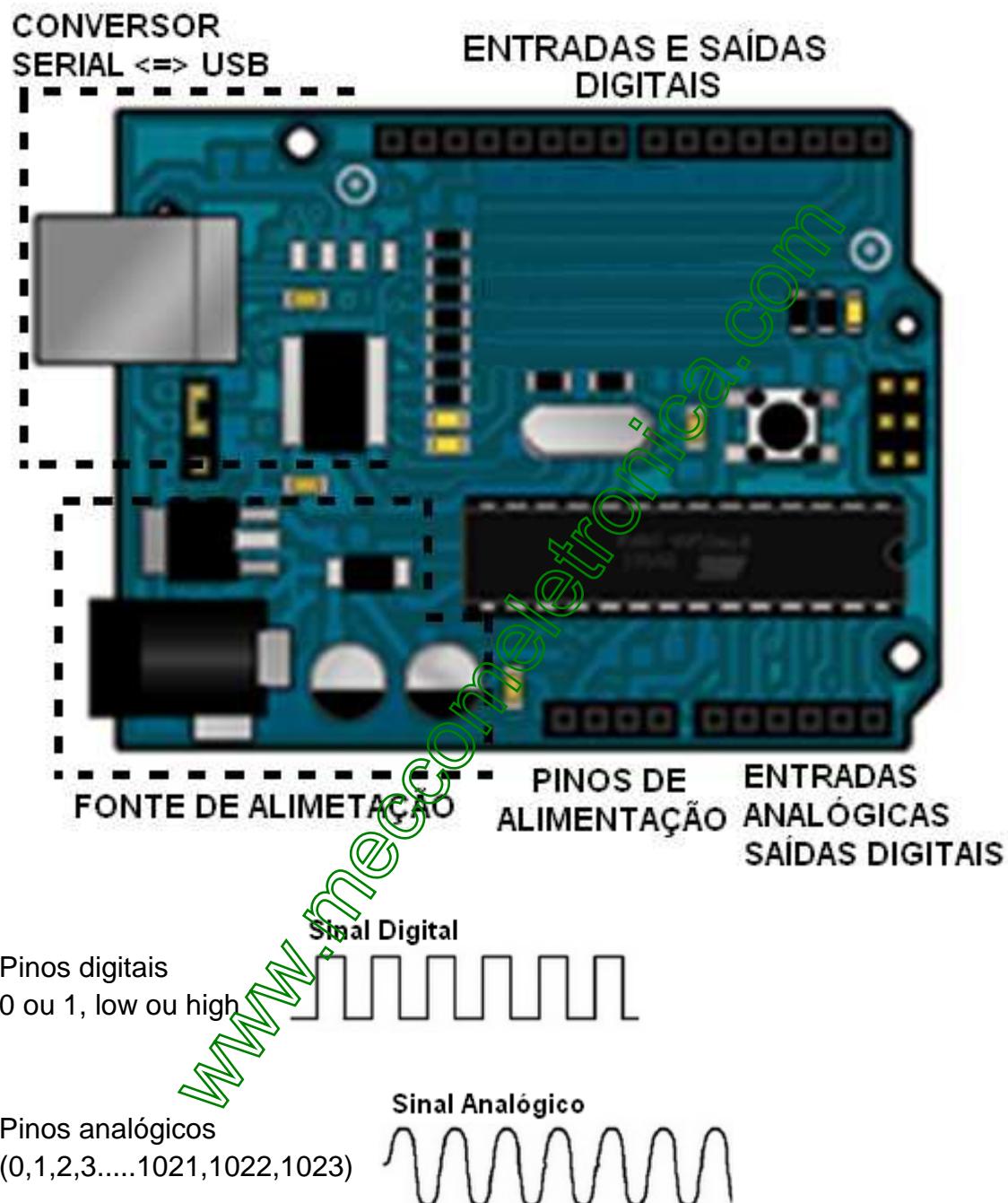


2. O Hardware



Arquitetura do Arduino

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



2.1 Fonte de Alimentação

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

2.2 Núcleo CPU:

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

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

2.3 Entradas e Saídas

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



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

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

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

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

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

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

2.3.1 Entradas Digitais

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

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

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

2.3.2 Entradas Analógicas

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

2.3.3 Saídas Digitais

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

2.4 Pinos com funções especiais

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

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

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

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

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

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

2.5 Firmware

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

3. O Software

Quando tratamos de software na plataforma do Arduino podemos referir-nos: ao ambiente de desenvolvimento integrado do Arduino e ao software desenvolvido por nós para enviar para a nossa placa. O ambiente de desenvolvimento do Arduino é um compilador (C e C++) que usa uma interface gráfica construída em Java.

Basicamente se resume a um programa IDE muito simples de se utilizar e de estender com bibliotecas que podem ser facilmente encontradas. As funções da IDE do Arduino são basicamente duas. Permitir o desenvolvimento de um software e enviá-lo à placa para que possa ser executado.

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



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

3.1 Estrutura do programa

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

```
File Edit Sketch Tools Help
Blink
/*
Blink

Pisca um led de dois em dois segundos
*/
//Função de inicialização
void setup() {
    // inicializa o pino digital como saída
    // O pino 13 possui um led integrado em muitas das placas Arduino
    pinMode(13, OUTPUT);
}

//looping principal
void loop() {

    digitalWrite(13, HIGH);      // acende o led conectado no pino 13
    delay(1000);                // espera um segundo (1000 mS)
    digitalWrite(13, LOW);       // apaga o led
    delay(1000);                // espera um segundo (1000 mS)
}

Done Saving
X-Duino ATmega168 on COM3
```

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

- ⇒ **A função setup** serve para inicialização da placa e do programa. Esta sessão é executada uma vez quando a placa é ligada ou resetada através do botão. Aqui, informamos para o hardware da placa o que vamos utilizar dele. No exemplo, vamos informar para a placa que o pino 13 será uma saída digital onde está conectado um LED (no Arduino UNO o pino 13 possui um led integrado).
- ⇒ **A função loop** é como se fosse a main () da placa. O programa escrito dentro da função loop é executado indefinidamente, ou seja, ao terminar a execução da última linha desta função, o programa inicia novamente a partir da primeira linha da função loop e continua a executar até que a placa seja desligada ou o botão de reset seja pressionado.

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

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

3.2 Serial Monitor

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



The screenshot shows the Arduino IDE interface. The top menu bar includes File, Edit, Sketch, Tools, and Help. Below the menu is a toolbar with various icons. The main workspace displays a sketch named "TesteSerial.s". The code in the sketch is:

```
/*
 * Teste Serial
 */

//Função de inicialização
void setup() {
    Serial.begin( 9600 ); //inicializa a comunicação na velocidade 9600
    Serial.println("Teste de comunicação serial"); //envia texto
}

int count = 0;

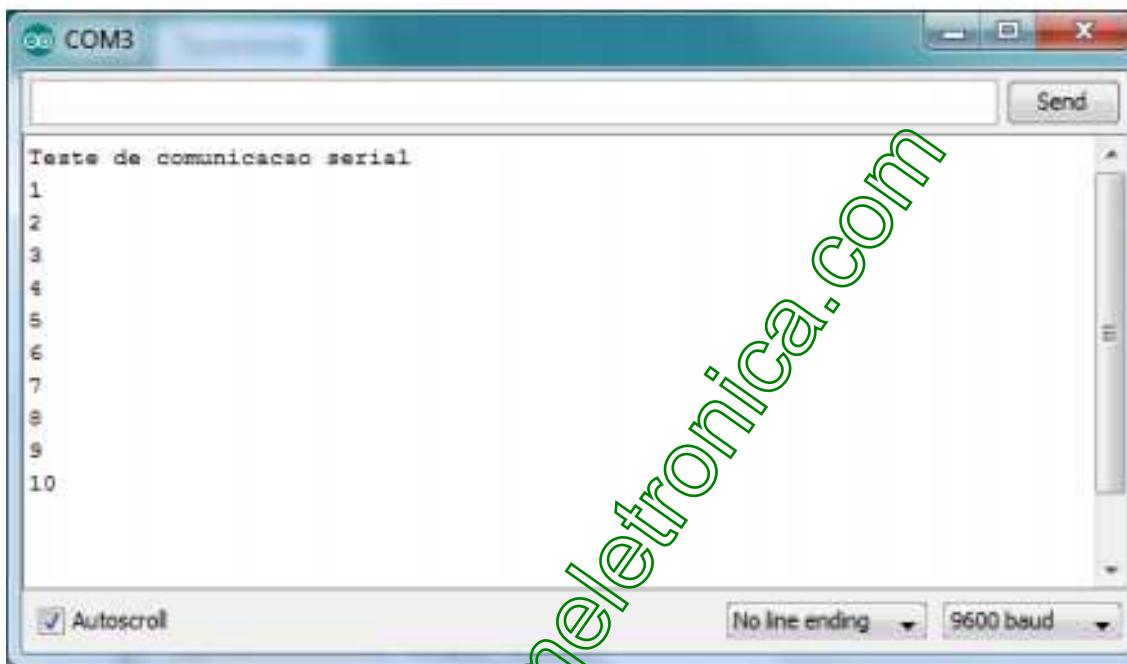
//looping principal
void loop() {

    count++;
    Serial.println( count, DEC ); //envia o valor da contagem
    delay(1000); //espera um segundo (1000 ms)
}
```

A red box highlights the "Serial Monitor" button in the top right corner of the IDE window. A watermark for "www.meccomeletronica.com" is diagonally across the image.

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

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



3.3 Outros comandos

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

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

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

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

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

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

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

4. UTILIZANDO UM PROTOBOARD

Até aqui já acendemos um Led interno do arduíno, utilizando o Led já incluso do pino 13. Agora vamos acender outro Led, agora utilizando o protoboard. Primeiro devemos saber qual a tensão necessária para acender o Led. Para isso utilizamos a tabela abaixo, sendo que cada cor tem a sua tensão específica Vermelho Laranja Amarelo Verde Azul Branco

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

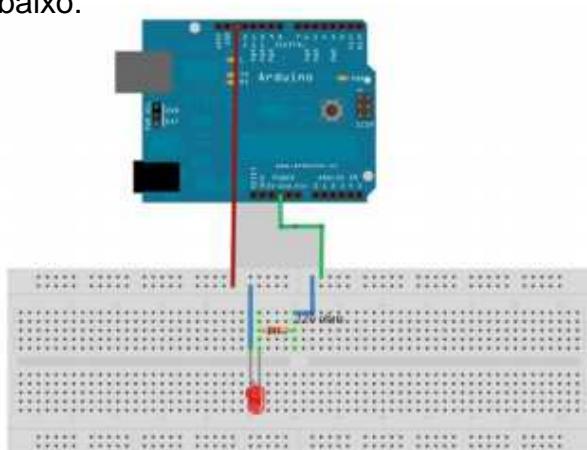
Sabemos que a porta digital escreve HIGH (5v) ou LOW (0v). Logo, se colocarmos 5v o Led irá queimar. Portanto devemos colocar um resistor em série com o Led para limitarmos a corrente que passa por ele. Essa corrente deve ser de algo em torno de 20mA. Assim, utilizando um pouquinho de teoria de circuitos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

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

Assim teremos a tensão do Led mais a tensão do resistor igual a 5v (tensão da fonte), e nosso Led estará seguro contra sobre corrente. Nem sempre vamos conseguir encontrar um resistor igual ao valor calculado, pois ele pode ser um valor não comercial. Quando isso ocorre basta associar resistores ou utilizar algum outro um pouco acima do valor encontrado.

Outro ponto importante é a polaridade do Led (ou de qualquer outro componente). Observe que este possui uma perna maior que a outra. A maior é positivo e a menor negativo. Portanto na hora de ligar, não podemos confundir caso contrário isso também queimarará o Led. Aqui o exemplo foi um Led, mas poderia ser um sensor, um motor ou algum outro componente. Portanto é preciso atenção para não queimarmos nossos equipamentos.

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



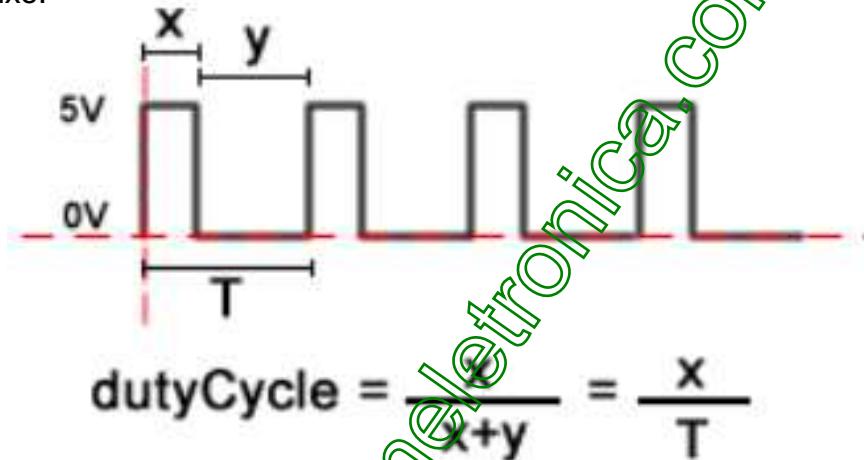
Ligado o circuito conforme a figura 8, podemos passar para o Arduino o código da figura 5 (o exemplo de como acender Led) que o nosso Led externo vai acender assim como o interno.

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

www.meccomeletronica.com

5. Utilizando o PWM

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



Portanto, o que controlamos através do software é justamente o duty cycle, ou seja, o percentual de tempo em que o sinal fica em 5v. Dessa forma, podemos utilizar essa técnica para limitar a potência de algum circuito, como por exemplo, controlar um servomotor.

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

```

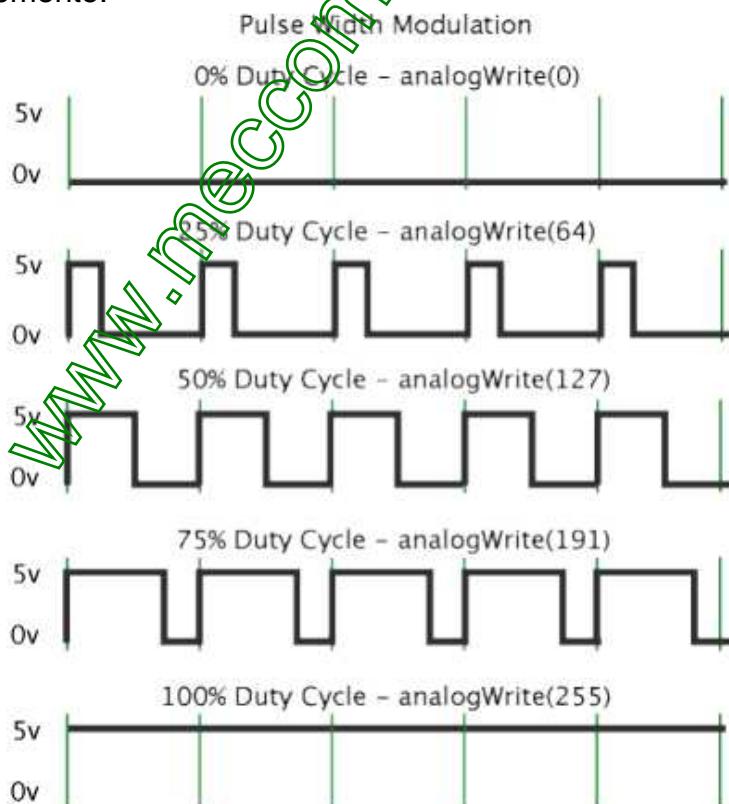
#define LED 11

void setup () {
    pinMode(LED, OUTPUT); //pino 11 ajustado como saída
}

void loop () {
    int i;
    for (i=0; i<255; i++){// variando i de 0 a 255
        analogWrite(LED,i);// escrevendo o valor de i no pino 11
        delay(30); // esperando 30 milésimos de segundo
    }
}

```

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



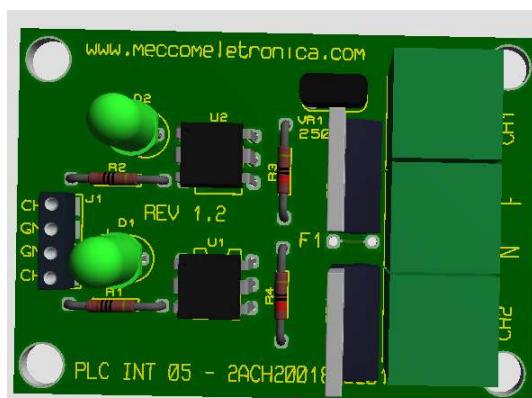
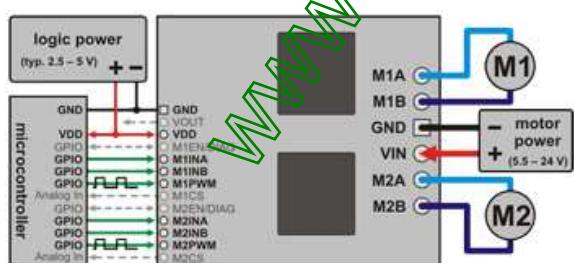
6. SHIELDS

O Arduino possui o que chamamos de Shields, que são nada mais, nada menos do que outras plaquinhas que se acoplam à placa original, agregando diversas outras funcionalidades. Existe diversos Shields, com diversas funções. Alguns servem como entrada, como saída e outros como entradas e saída. Com eles conseguimos, por exemplo, fazer com que o Arduino se comunique numa rede Ethernet ou via USB com um celular Android.

- ⇒ Funcionalidades de alto nível
- ⇒ Encaixados em cima do Arduino
- ⇒ Utilizam alguns pinos encaminham os outros



Alguns Shields possuem circuitos integrados prontos para controlarmos motores sem que precisemos nos preocupar com complicações eletrônicas envolvidas.



7. CONTROLANDO UM ULTRASSOM

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



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

```
teste_ultrassom.S

#define echoPin 13 //Pino 13 recebe o pulso do echo
#define trigPin 12 //Pino 12 envia o pulso para gerar o echo
void setup()
{
    Serial.begin(9600); //inicia a porta serial
    pinMode(echoPin, INPUT); // define o pino 13 como entrada (recebe)
    pinMode(trigPin, OUTPUT); // define o pino 12 como saída (envia)
}
void loop()
{
    //seta o pino 12 com um pulso baixo "LOW" quando desligado ou ainda 0
    digitalWrite(trigPin, LOW);
    // delay de 2 microssegundos
    delayMicroseconds(2);
    //seta o pino 12 com pulso alto "HIGH" quando ligado ou ainda 1
    digitalWrite(trigPin, HIGH);
    //delay de 10 microssegundos
    delayMicroseconds(10);
    //seta o pino 12 com pulso baixo novamente
    digitalWrite(trigPin, LOW);
    //pulseInt é o tempo entre a chamaada e o pino entrar em high
    long duration = pulseIn(echoPin,HIGH);
    //Esse calculo é baseado em s = v . t, lembrando que o tempo vem dobrado
    //porque é o tempo de ida e volta do ultrassom
    long distancia = duration /29 / 2 : |
    Serial.print("Distancia em CM: ");
    Serial.println(distancia);
    delay(1000); //espera 1 segundo para fazer a leitura novamente
}
```

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

para que o resultado saia em centímetros, temos que dividir o tempo (duration) por 29. Por fim o resultado é impresso no serial monitor.

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

www.meccomeletronica.com

8. Utilizando a interrupção

Uma interrupção de entrada e saída (IO) é uma função que é executada quando existe uma mudança estado no pino correspondente, independente do ponto em que se encontra a execução do programa. O Arduino UNO possui dois pinos que podem ser ligados como interrupção, os pinos 2 e 3 (no Arduino mega temos 6 interrupções).

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

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

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

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

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

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

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

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

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

9. Adicionando uma biblioteca

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

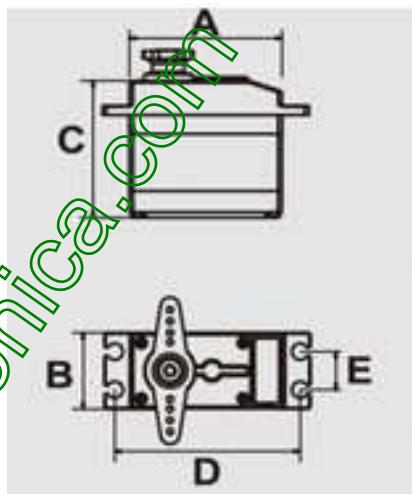
Para mostrar como proceder, vamos adicionar a biblioteca MsTimer2.h como exemplo. Primeiro vamos baixá-la na página do Arduino (<http://arduino.cc/playground/Main/MsTimer2>).

Feito isso, descompacte o arquivo.zip que foi baixado. Agora vá até a pasta onde você “instalou” o seu IDE para o Arduino e procure pela pasta libraries. Dentro deste diretório copie a pasta que foi extraída anteriormente. Por fim, vamos verificar se a biblioteca foi mesmo detectada pelo IDE. Vá em Files > Examples e verifique se a biblioteca que acabamos de adicionar está ali.

Se sim, a instalação ocorreu tudo bem e você já pode começar a utilizar a sua nova biblioteca. Agora é só “chamá-la” no seu código, que neste caso ficaria: `#include <MsTimer2.h>`. Vale a pena destacar que na própria página onde você baixou a biblioteca, possui as instruções de como utilizar a mesma.

10. Explorando um servomotor

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



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

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

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

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

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

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

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

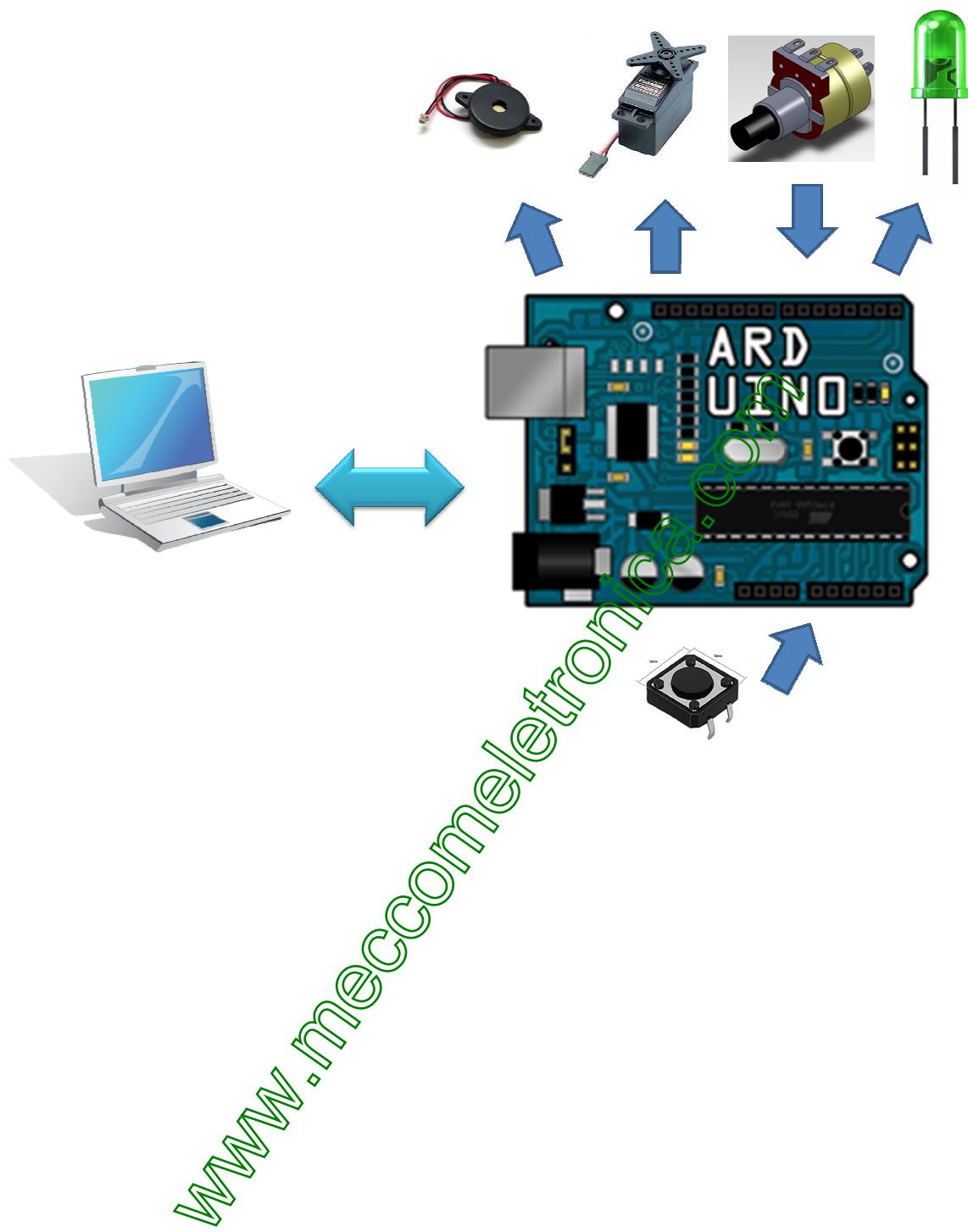
5 - Podemos utilizar também as funções **motor1.attached()** para checar se o servo está associado a algum pino e também a **motor1.detach()** para desassociar o servo ao seu pino declarado do item 2 acima

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

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

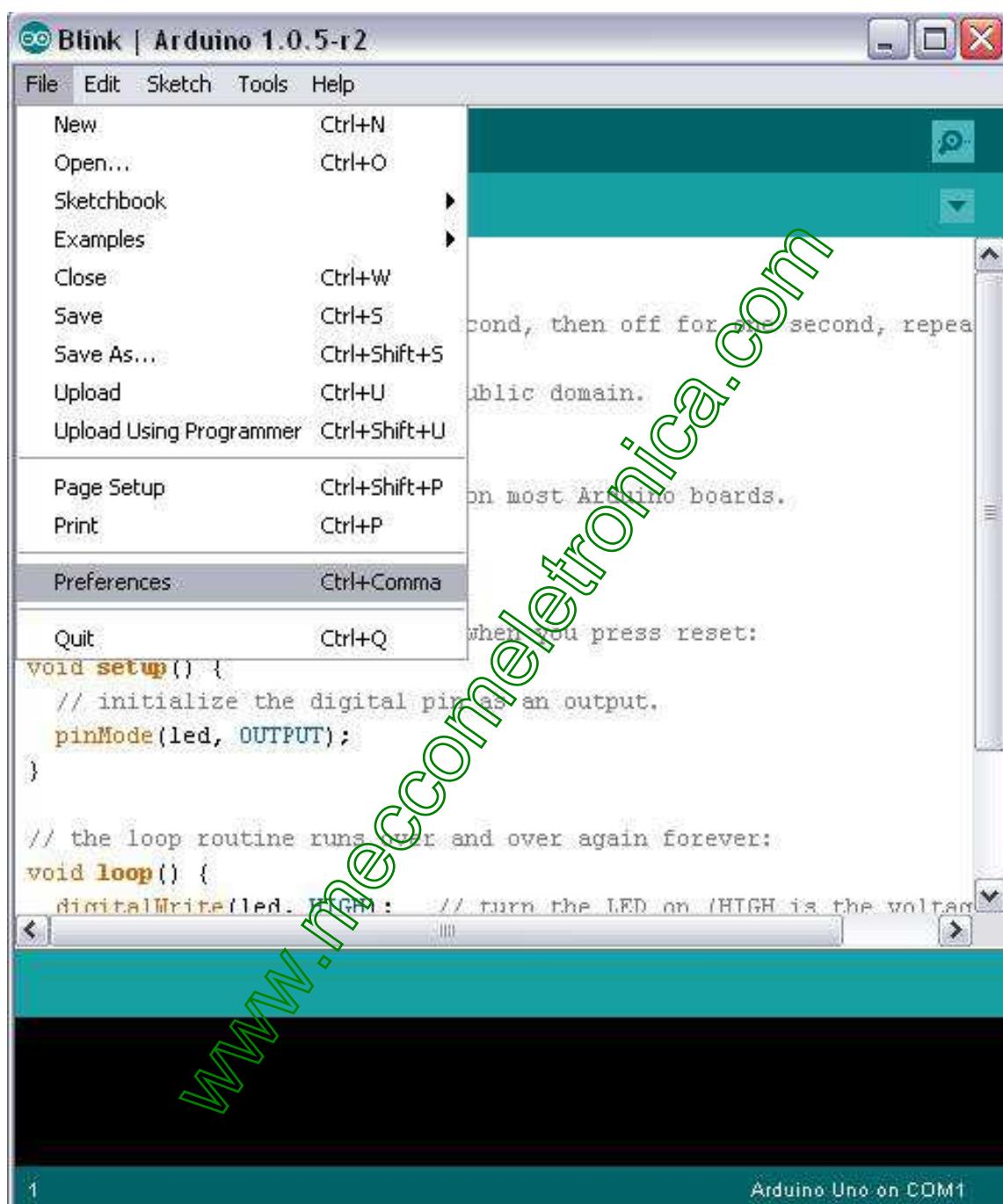
void setup()
{
    motor1.attach(8); //associando o pino 8 ao servo
}

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



LOCALIZANDO O ARQUIVO .HEX

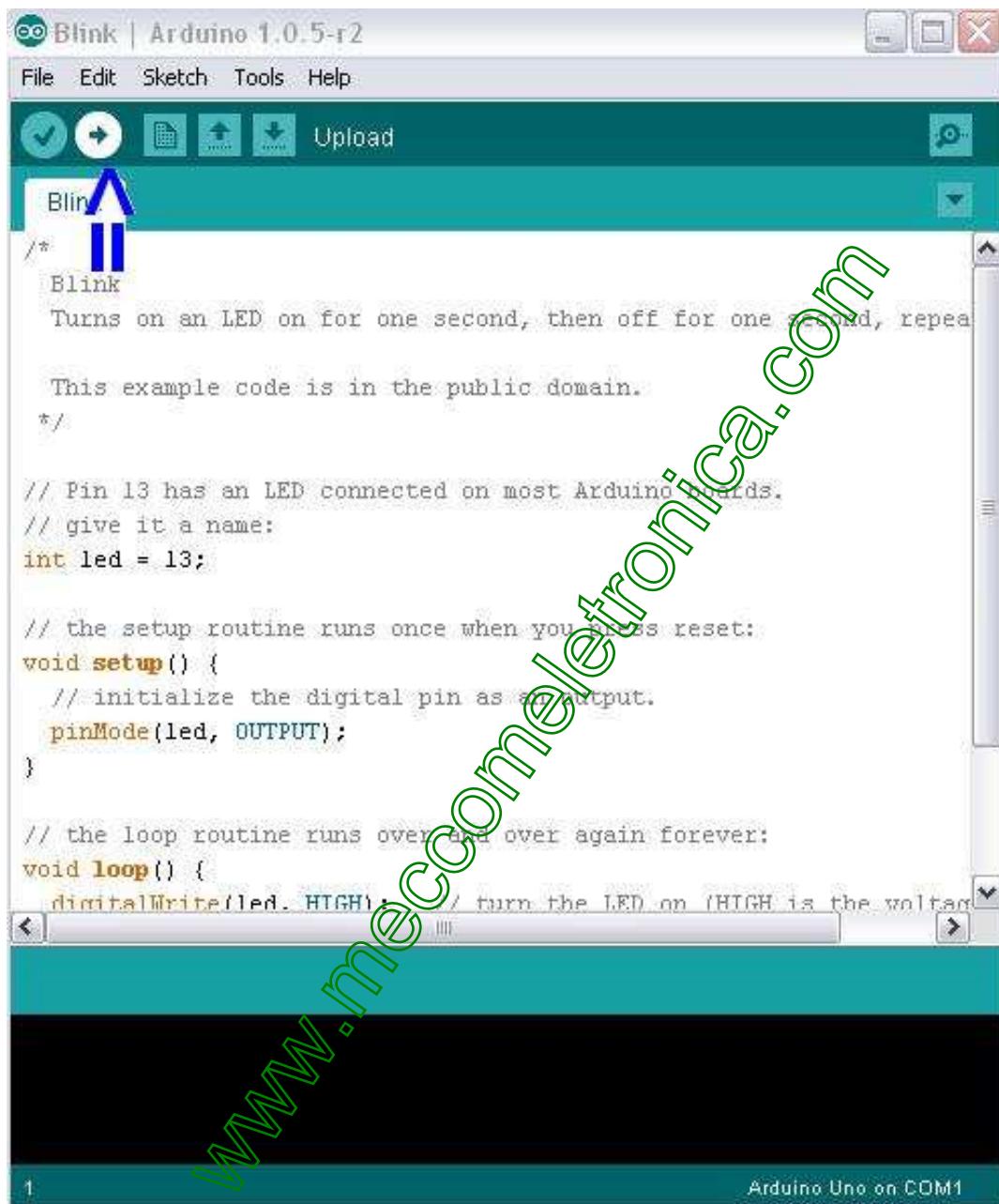
Abrir a IDE do Arduino, selecionar File, Preferências.



Marque a opção Compilation



Selecione a opção Upload, o programa nesse momento irá conferir o código, estando tudo certo, fará o processo de envio do arquivo .hex, neste caso como não há uma placa conectada ele apontará um erro.



```
eo Blink | Arduino 1.0.5-r2
File Edit Sketch Tools Help
Upload
Blink
/*
Blink
  Turns on an LED on for one second, then off for one second, repeating
  This example code is in the public domain.
*/
// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output:
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000); // wait for a second
  digitalWrite(led, LOW); // turn the LED off by making the voltage LOW
  delay(1000); // wait for a second
}
```

1 Arduino Uno on COM1

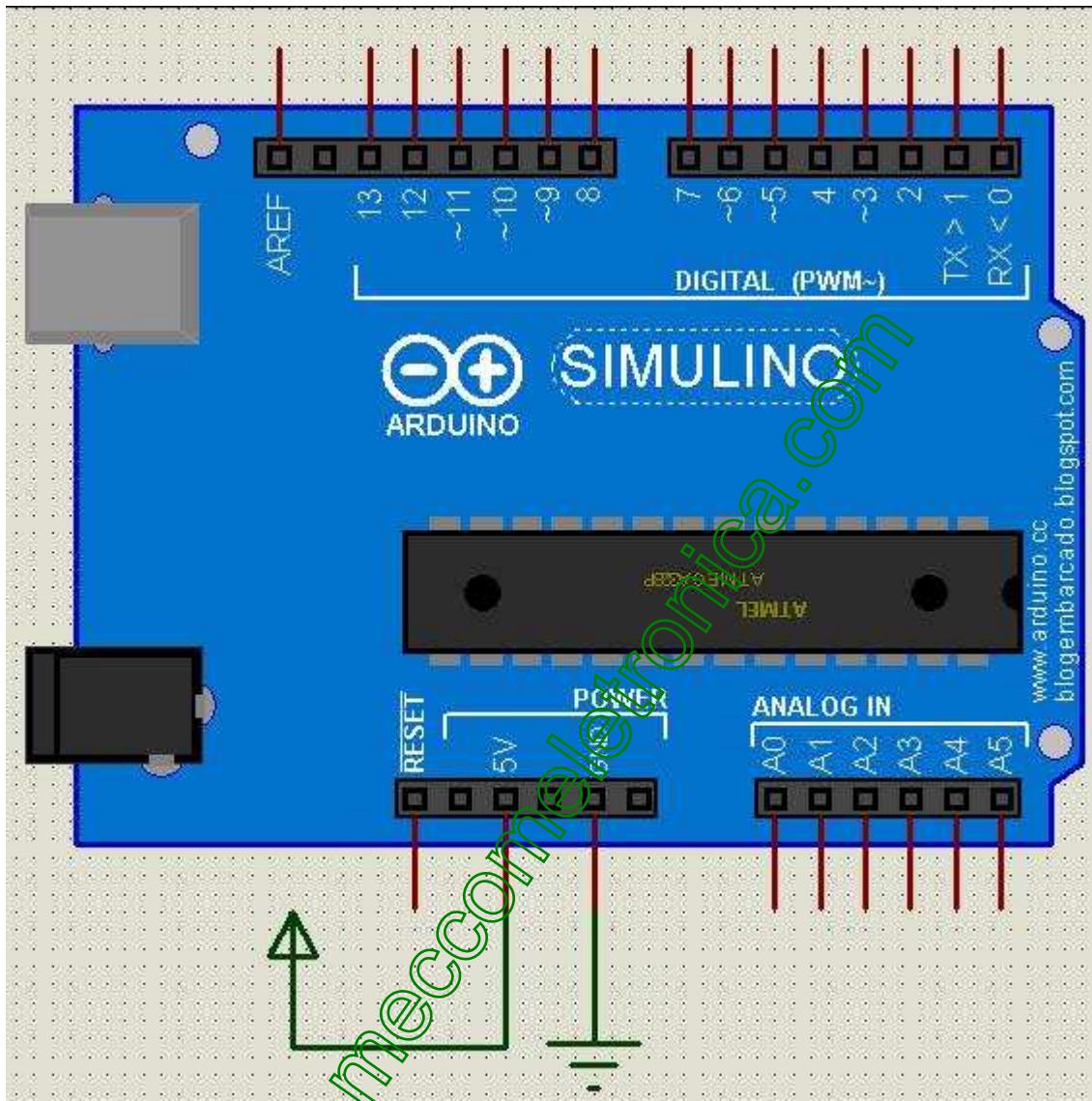
Use a barra de rolagem e encontre na antepenúltima linha o caminho da pasta temporária com o nome completo e o seu arquivo .hex gerado.

The screenshot shows the Arduino IDE interface. The top menu bar includes File, Edit, Sketch, Tools, Help, and a language selection dropdown. Below the menu is a toolbar with icons for upload, download, and other functions. The main workspace displays the 'Blink' sketch code. A large watermark 'www.meccomeletronica.com' is diagonally across the code area. The terminal window at the bottom shows the output of the upload process:

```
Done uploading.
C:\DOCUME~1\ADMINI~1\CONFIG~1\Temp\build2437400235558994325.tmp\Blink.cpp.eir
C:\DOCUME~1\ADMINI~1\CONFIG~1\Temp\build2437400235558994325.tmp\Blink.cpp.hex
Binary sketch size: 1.084 bytes (of a 32.256 byte maximum)
avrduude: stk500_getsync(): not in sync: resp=0x00
```

A blue arrow points to the file path 'C:\DOCUME~1\ADMINI~1\CONFIG~1\Temp\build2437400235558994325.tmp\Blink.cpp.hex' in the terminal window.

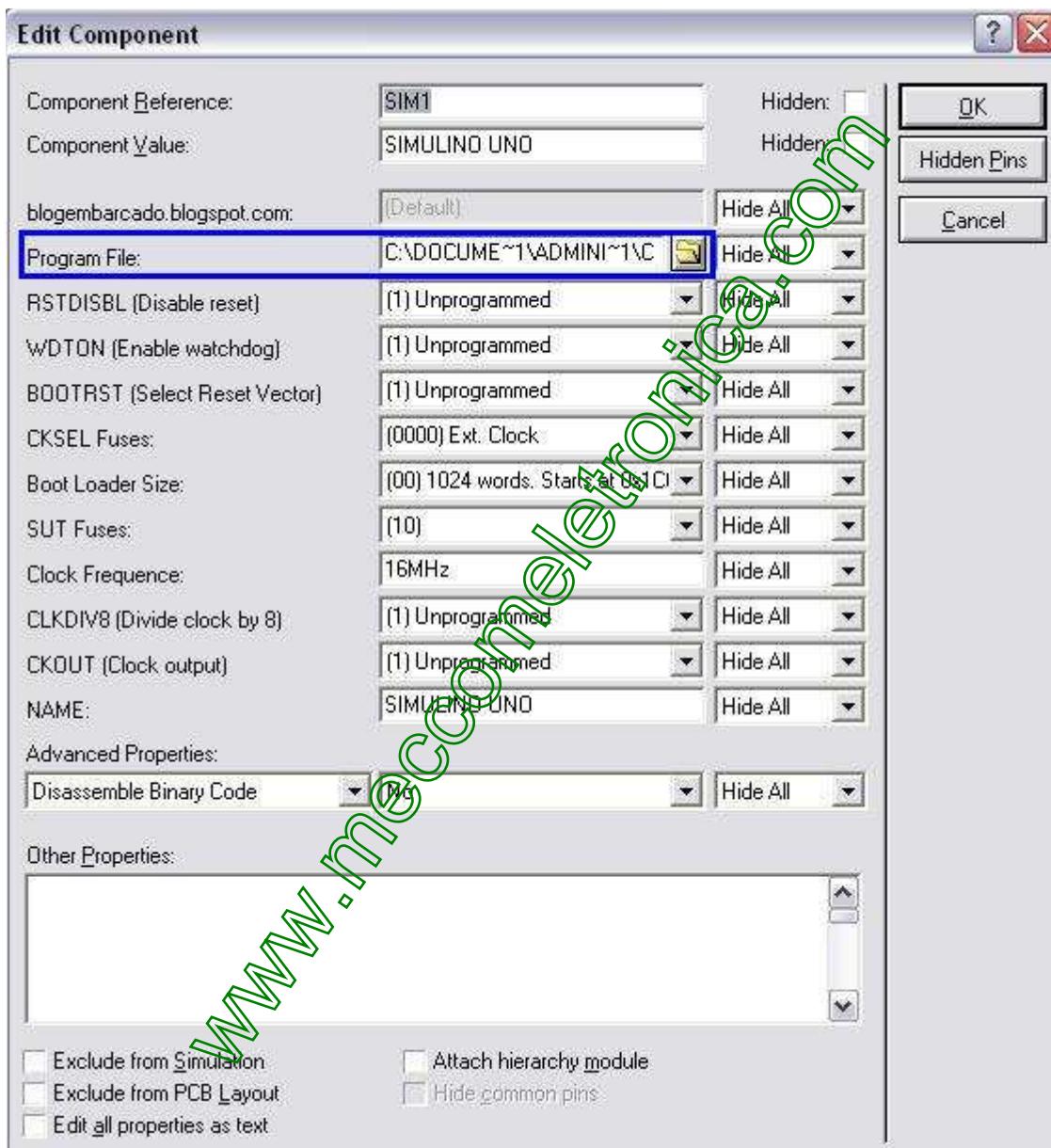
Abra o proteus, selecione o modelo de arduino. Sempre insira os pinos de fonte na placa do arduino.



PARA SELECIONAR O ARQUIVO .HEX

Dê um duplo clique no ATmega da placa arduino, após isso aparecerá a janela Edit Component.

Verifique as configurações de acordo com a imagem abaixo e não altere, a não ser o caminho do arquivo .hex



Basta colar o caminho da encontrado na antepenúltima linha do código em Program File, e pressionar a tecla OK.

www.meccomelettronica.com