



Apostila de estudos Plataforma Arduino

Programação e eletrônica

Autor: Prof. Marcílio Oliveira
Versão: 1.0.05-2019

Esta apostila é um compilado dos conceitos básicos principais para iniciar com Arduino.

Não comercialize, tenha o espírito *open source*.

Contato: Prof. Marcílio Oliveira (marciliofoneto@gmail.com)

Sumário

O Arduino.....	4
Introdução.....	4
Características.....	5
O Arduino IDE.....	7
O que é IDE - <i>Integrated Development Environment</i> ?.....	7
Instalação do Arduino IDE em Linux.....	7
Instalação do Arduino IDE em Windows.....	9
Arduino IDE Online.....	10
Conhecendo o Arduino IDE.....	10
Conexão da placa Arduino - PC (Arduino IDE).....	12
Primeiro programa.....	13
Principais funções e comandos.....	14
Essenciais.....	15
Função setup().....	15
Função loop().....	15
Funções de temporização.....	15
Função delay().....	15
Função millis().....	16
Pinos digitais.....	17
Função pinMode().....	17
Função digitalWrite().....	18
Função digitalRead().....	19
Programe você mesmo.....	20
Piscando um LED.....	20
Acendendo LED com botão.....	21
Pinos analógicos.....	23
Conversor Analógico-Digital.....	24
Função analogRead().....	24
Função analogWrite().....	25
Programe você mesmo.....	26
Leitura de potenciômetro.....	26
Controlando o brilho do LED.....	27
Comunicação serial.....	28
Monitor serial.....	28
Função Serial.begin().....	29
Aguardando conexão com a porta serial.....	30
Função Serial.print().....	30
Função Serial.println().....	31
Formatando as saídas: Serial.print() e Serial.println().....	32
Função Serial.available().....	33
Função Serial.read().....	34
Função Serial.write().....	35

Programe você mesmo.....	36
Controlando o brilho do LED através da serial.....	37
Controlando o sentido de giro do motor DC.....	38
Controlando a velocidade e a rotação do motor DC.....	40
Apêndice A.....	42
Tabela ASCII.....	42
PWM (Modulação por Largura de Pulso).....	43
XCTU.....	44
XBee.....	45
Ponte H.....	47

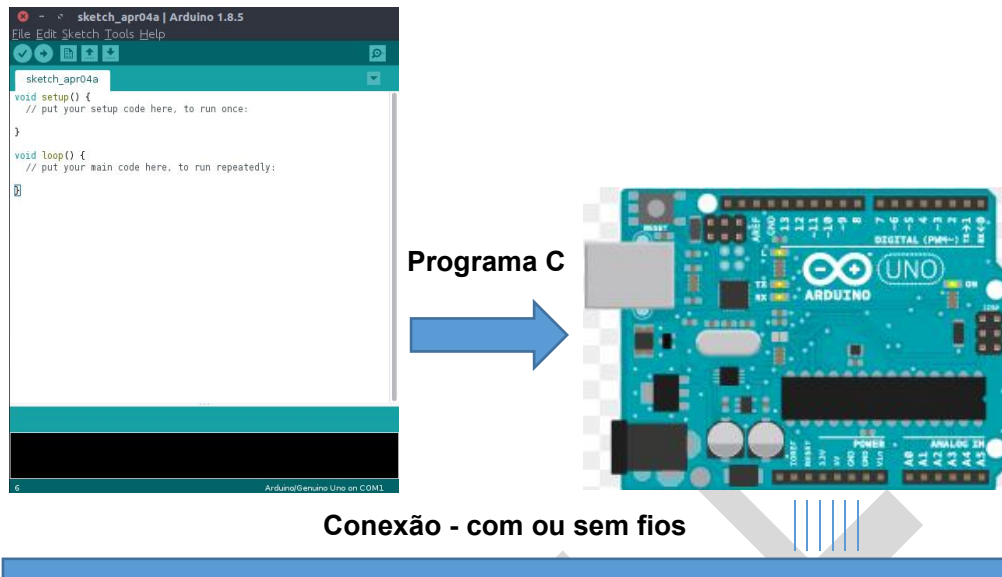
O Arduino

Introdução

Arduino é uma plataforma eletrônica *open-source* que proporciona a construção de várias aplicações envolvendo *hardware* e *software* de maneira “fácil”. Com sua “simplicidade”, proporcionou a programação para microcontroladores fácil e divertida. Entenda-se “fácil” por ter uma curva de aprendizado alta, contendo diversas bibliotecas que auxiliam na construção de várias aplicações, sejam estas simples ou robustas, sem a necessidade de conhecer a fundo a arquitetura ou fundamentos mais detalhados. Além disso, a plataforma Arduino aceita diversos *shields* (módulos de hardware que se acoplam na plataforma original), e isso propicia a construção de projetos complexos sem a necessidade de construir circuitos-integrados e/ou outros *hardwares* específicos, tais como: Ethernet, Módulos WI-FI, SD-card, entre outros.

Com uma placa Arduino, que é confeccionada com um microcontrolador ATmega, você é capaz de ler *inputs* de diversas fontes, como: sensores de luz, humidade, botões, ou uma mensagem vinda da rede; e transformá-los em *outputs* para acionar atuadores que realizarão atividades como: ativar um ou mais motores, ligar LED, controlar um robô, e muitas outras. Ao longo do tempo o Arduino tem sido figura principal em projetos de entusiastas, programadores profissionais, estudantes, pesquisas científicas, etc.

Assim sendo, para iniciar a conversa entre o Arduino e um conjunto de sensores, motores ou módulos do seu projeto, você envia um conjunto de instruções ao microcontrolador da placa através da linguagem de programação C.



No esquemático acima você pode ver que é necessário programar o código C em uma IDE própria do Arduino e, em seguida, o código compilado é carregado na plataforma, via serial. A IDE é responsável por identificar a porta serial na qual o Arduino está conectado, além disso, é necessário configurar a plataforma para reconhecer o modelo correto do Arduino utilizado. Nas próximas seções você saberá como preparar o ambiente e iniciar seus projetos com o Arduino.

Características

O modelo de Arduino utilizado nos exemplos deste documento é o UNO. Portanto, é importante saber de suas características e conhecer o hardware utilizado.

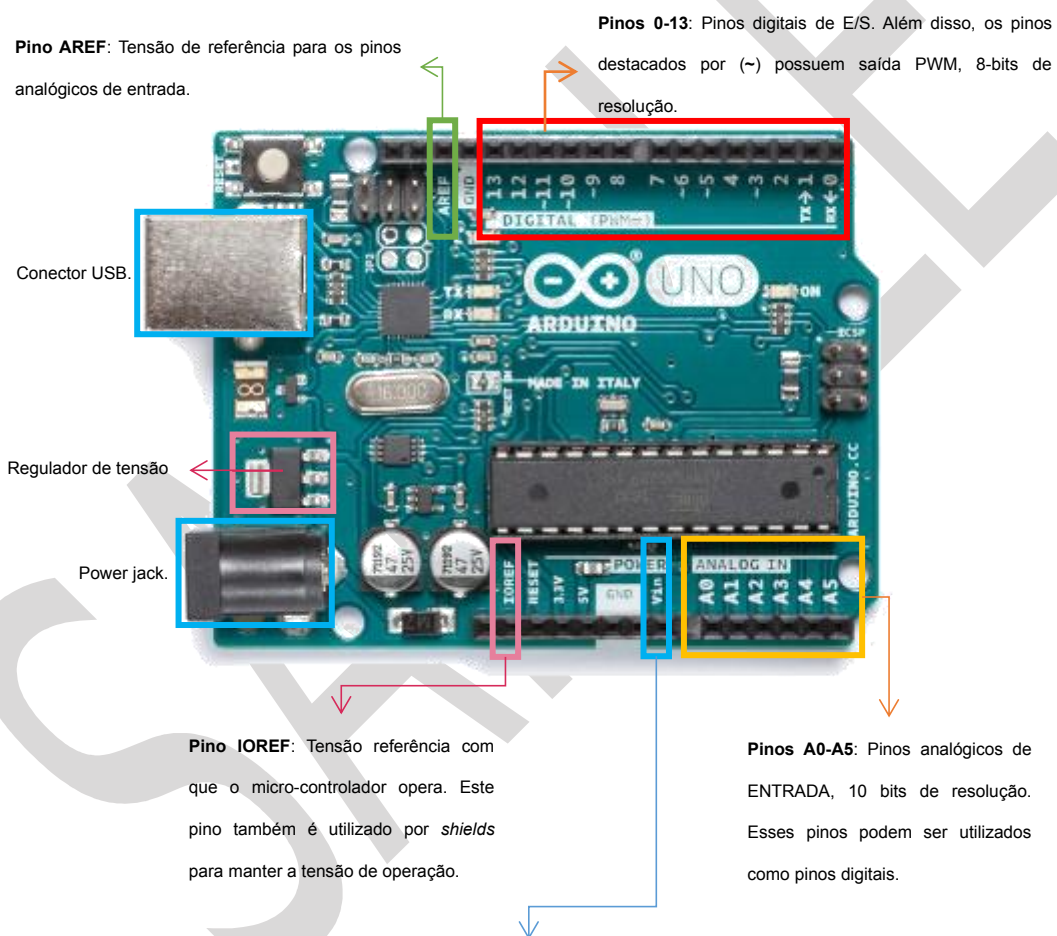
As características da plataforma Arduino UNO são:

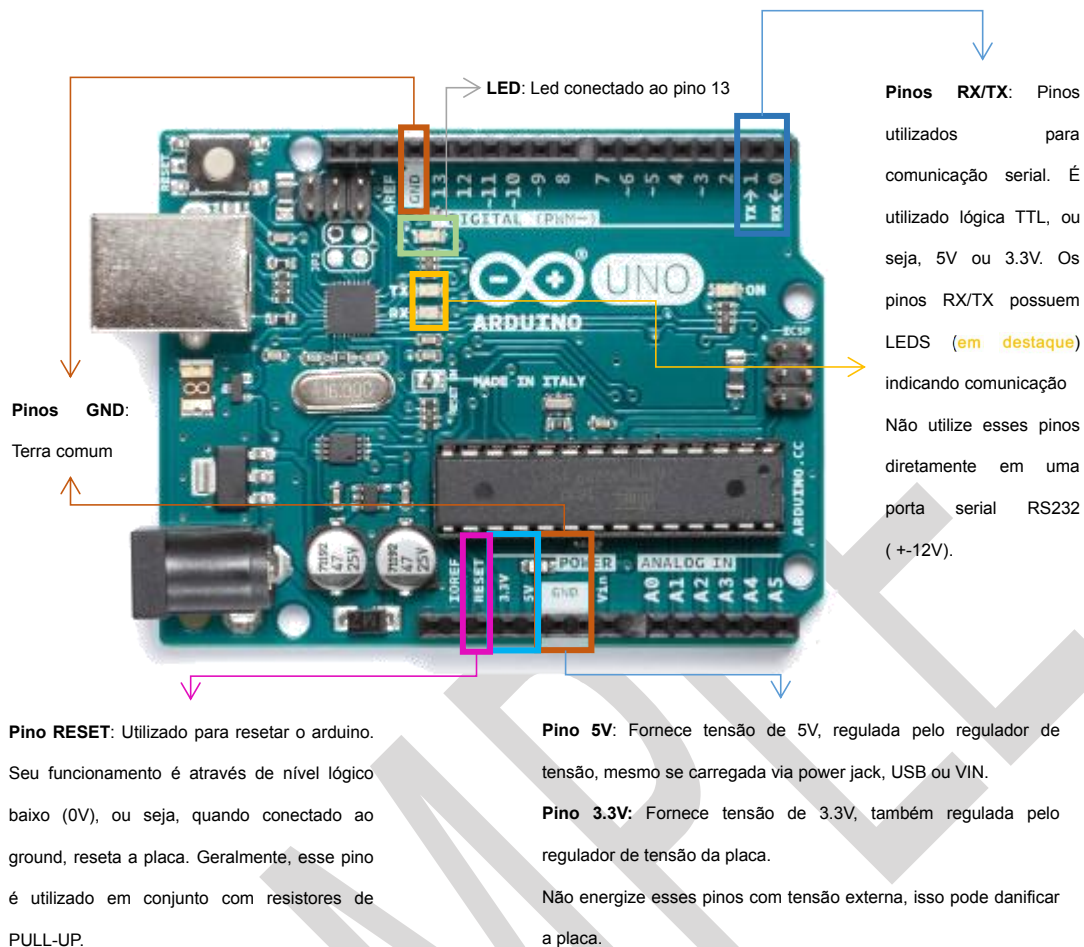
- Microcontrolador ATmega328;
- **Tensão de operação: 5V;**
- Tensão de entrada recomendada (alimentação DC externa): 7-12V;
- **14 pinos digitais de I/O - Sendo que 6 desses pinos possuem saída PWM (~);**
- **6 pinos de entrada analógicos;**
- **Conexão USB;**
- **Conector power jack**

- 32k de memória flash;
- 16Mhz de frequência de clock;
- **Corrente contínua por pino de E/S: 20mA;**
- Corrente contínua para o pino de 3.3V: 50mA.

Nas características acima estão destacadas as informações essenciais que você deve se atentar, saber e respeitar, para operar o Arduino de maneira correta. Para mais detalhes sobre o hardware, acesse: <https://store.arduino.cc/usa/arduino-uno-rev3>

Além disso, abaixo é destacado os principais pinos utilizados nos projetos e suas funções, respectivamente.





O Arduino IDE

O que é IDE - *Integrated Development Environment*?

Um ambiente de desenvolvimento integrado - IDE *Integrated Development Environment* - é um programa que reuni ferramentas de apoio para desenvolvimento de software. Assim sendo, para programar para o Arduino deve-se ter a IDE Arduino, e essa IDE tem como vantagem ser *cross-plataform*, isto é, independe de ambiente, podendo ser executado em Windows, MacOS e Linux.

Instalação do Arduino IDE em Linux

Para instalar o Arduino IDE em seu computador basta acessar o link <https://www.arduino.cc/en/Main/Software> e escolher o instalador compatível com o seu sistema operacional. Esta apostila utilizará o sistema operacional Linux, portanto,

Autor: Prof. Márcilio Oliveira

Versão 1.0.05-2019 - Não comercializar.

ilustrará a instalação nesse ambiente. A instalação em Windows é bem mais prática, seguindo o famoso *Next-Next-Finish*.

Ao acessar o link anterior, você terá várias opções, escolha a opção compatível com seu ambiente. Suponha que você tenha Windows, clique em “Windows installer”.



Uma vez realizado o *download* do IDE, siga os passos abaixo para conseguir utilizá-lo.

1. Acesse o terminal do sistema e navegue até a pasta que contém o download do IDE. No caso, o download está na pasta “Desktop”, para acessá-la, digite “cd Desktop/”.

```
marcilio@ubuntu-laptop: ~/Desktop
marcilio@ubuntu-laptop:~$ cd Desktop/
marcilio@ubuntu-laptop:~/Desktop$ ls
a0000666_front.jpg  arduino-1.8.5-linux64.tar.xz
marcilio@ubuntu-laptop:~/Desktop$
```

2. Descompacte o arquivo utilizando o comando “tar -xf arduino-<versão-download>”. Onde <versão-download> corresponde ao nome completo do arquivo baixado no site.

```
marcilio@ubuntu-laptop: ~/Desktop
marcilio@ubuntu-laptop:~/Desktop$ ls
a0000666_front.jpg  arduino-1.8.5-linux64.tar.xz  DesktopLinux  Untitled
marcilio@ubuntu-laptop:~/Desktop$ tar -xf arduino-1.8.5-linux64.tar.xz
```

3. Acesse a pasta descompactada pelo passo anterior e execute o comando “ls” para visualizar os arquivos contidos dentro da pasta. O arquivo de interesse é o “install.sh”

```
marcilio@ubuntu-laptop: ~/Desktop/arduino-1.8.5
marcilio@ubuntu-laptop:~/Desktop$ ls
a000066_front.jpg  arduino-1.8.5-linux64.tar.xz  DesktopLinux  Untitled
marcilio@ubuntu-laptop:~/Desktop$ tar -xvf arduino-1.8.5-linux64.tar.xz
marcilio@ubuntu-laptop:~/Desktop$ cd arduino-1.8.5/
marcilio@ubuntu-laptop:~/Desktop/arduino-1.8.5$ ls
arduino      hardware    lib         revisions.txt  uninstall.sh
arduino-builder  install.sh  libraries  tools          tools-builder
examples      java       reference  tools-builder
marcilio@ubuntu-laptop:~/Desktop/arduino-1.8.5$
```

4. Após o passo 3, execute o comando “./install.sh” no terminal e aguarde a instalação do Arduino IDE em seu Linux. Em seu desktop será criado um ícone para acessar o IDE.

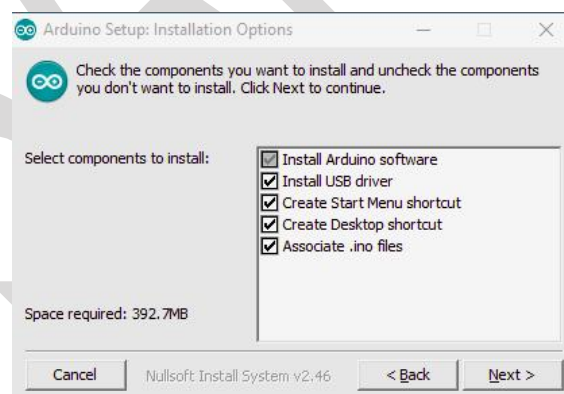


Para mais informações, consulte o site do projeto Arduino (www.arduino.cc).

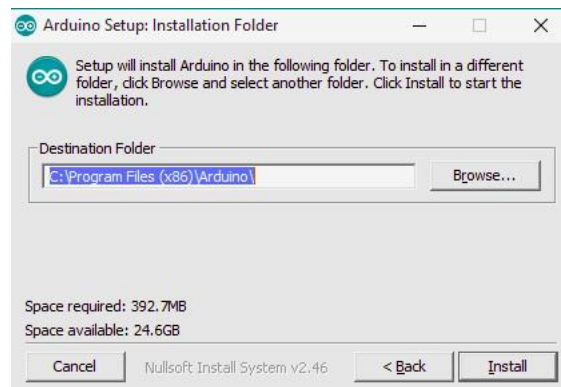
Instalação do Arduino IDE em Windows

O processo de instalação do Arduino IDE foi extraído do próprio site do projeto Arduino. Link: <https://www.arduino.cc/en/Guide/Windows>

1. Após baixar o Arduino IDE para Windows, dê duplo clique e aguarde a exibição da tela abaixo.



2. Dê Avançar (Next) e escolha o local desejado para a instalação.



3. Após escolher o local adequado, clique em Instalar (Install) e aguarde o ícone do ambiente aparecer na área de trabalho (Desktop).

Para outras informações, consulte o site: <https://www.arduino.cc>

Arduino IDE Online

Com o crescimento de serviços disponíveis na rede e a explosão do conceito de *cloud computing*, ferramentas que até então eram executadas apenas localmente (instaladas nos PCs), hoje são entregues ao cliente via *web*. O Arduino também teve essa ideia e é possível programar via *browser*.

Nesta apostila não focaremos no acesso ao Arduino IDE online, mas para saber mais sobre a plataforma, acesse: <https://create.arduino.cc/>.

Conhecendo o Arduino IDE

O Arduino IDE é um programa simples que suporta programação em C e é capaz de reconhecer a placa conectada ao PC. Com isso, é capaz de compilar o código escrito pelo usuário - você - e carregá-lo na placa.

Como mencionado anteriormente, projetos para Arduino são escritos em linguagem C, portanto, toda e qualquer regra da linguagem utilizada em outra IDE, ex.: Code::Blocks, é válida para o Arduino IDE.

A imagem abaixo é mostra uma típica janela do ambiente de desenvolvimento Arduino. Nota-se que ao abrir o ambiente, duas funções são automaticamente declaradas. Mas

primeiro, vamos conhecer um pouco sobre os recursos do ambiente.



A barra principal do IDE é exibida na imagem abaixo, ela é essencial para a verificação do código escrito, bem como realizar o carregamento do código no microcontrolador. Em destaque, será mostrado as principais funções do IDE.



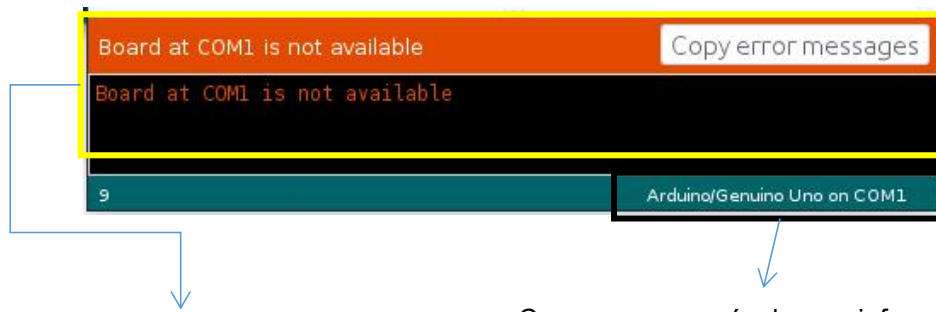
1 - Botão *Verify*: Responsável por verificar se há erros de codificação e sintaxe. Esse botão é muito útil, uma vez que as facilidades encontradas em IDEs como Netbeans e Eclipse, tais como: auto-completar e sugestões de métodos; não aparecem no Arduino IDE, um ambiente mais simplista.

2 - Botão *Upload*: Responsável por compilar o programa codificado e enviar ao microcontrolador.

3 - Botão *Serial Monitor*: Responsável por abrir o monitor serial, janela que exibe as mensagens trocadas através da comunicação serial entre o microcontrolador e o PC. Para acionar o monitor serial, o Arduino deve estar conectado ao PC, via USB. Na próxima seção será descrito o processo de conexão, reconhecimento e primeiro upload de programa no Arduino.

Os botões 4, 5 e 6 são utilizados para criar um novo documento de código, abrir um documento de código e salvar um documento, respectivamente.

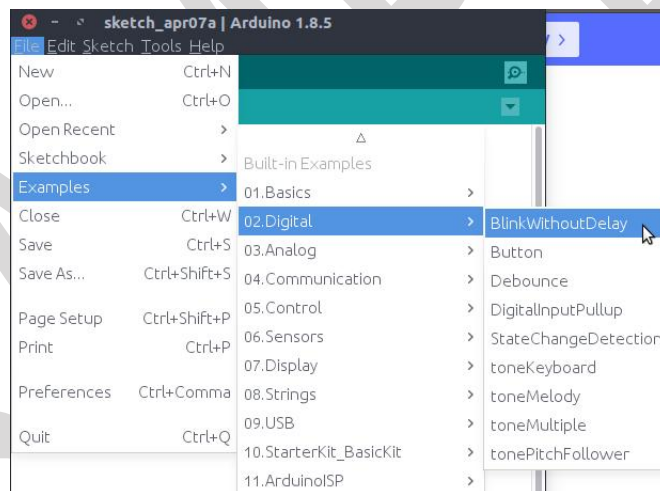
Na parte inferior do Arduino IDE tem-se uma área específica para notificar erros e/ou avisos relacionados à conexão, reconhecimento do Arduino, erros de codificação, entre outros. Veja um exemplo na imagem abaixo.



Área responsável por reportar ao usuário os eventuais erros de conexão, programação, entre outros.

Campo responsável por informar ao usuário qual PORTA (COM) o Arduino está conectado. Esse detalhe é importante para que o Arduino seja reconhecido na porta correta e configurado para “conversar” com o PC através desta porta.

O Arduino IDE, assim que instalado, disponibiliza ao usuário vários códigos de exemplo para que o aprendizado seja mais efetivo. Os códigos de exemplos podem ser visualizados através do menu superior “File --> Examples --> <Módulo_de_Escolha> --> <Exemplo_desejado>”. Veja na imagem abaixo a ilustração do acesso.



Nas próximas seções serão apresentados alguns dos códigos exemplares junto com comentários e explicações relevantes.

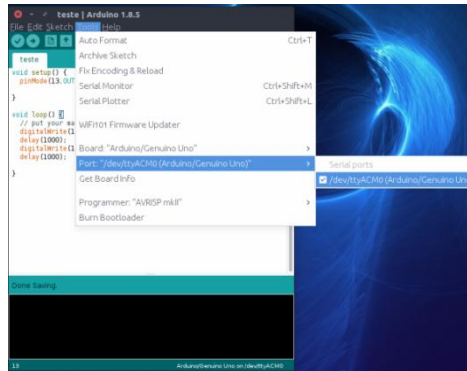
Conexão da placa Arduino - PC (Arduino IDE)

Agora que você já se familiarizou com o ambiente de desenvolvimento e conheceu as principais características da placa Arduino, chegou a hora de por a mão na massa. A seguir, conectaremos a placa Arduino no PC.

1. Com o cabo USB AB em mãos e a placa Arduino, conecte o terminal B (conector

maior) na placa Arduino e o terminal A (USB padrão) no PC.

2. Abra o Arduino IDE, clique no menu “Tools --> Port --> <porta>”. A porta selecionada deve ser a mesma que o sistema operacional escolheu para conversar com o microcontrolador, normalmente, em sistemas Windows, é COM1 ou COM3. Já em Linux, a ttyACM0. Veja na imagem abaixo o fluxo para selecionar a porta correta.



Em sistemas Linux é comum não termos permissão para escrita na porta designada ao conectar o Arduino, para isso, é necessário abrir o terminal e executar o comando abaixo e, caso seu sistema tenha senha, digitar a senha administrador.

“sudo chmod 777 -R /dev/ttyACM0”

No caso, a porta designada no exemplo foi a ttyACM0. Para Windows esse processo é transparente, bastando conectar a placa no PC.

Primeiro programa

Agora que já vimos os passos básicos de conexão e reconhecimento da placa Arduino, podemos testar alguns exemplos e desenvolver nosso primeiro programa. Os programas de Arduino são conhecidos como esboço - sketch. Com isso, serão demonstradas as primeiras funções utilizadas pelo Arduino para executar as tarefas que desejarmos.

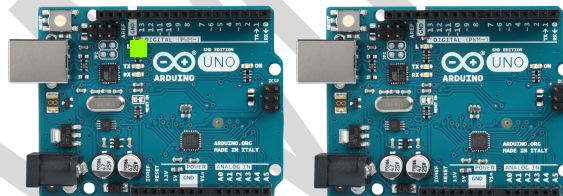
A fim de testar a conexão com a placa e executar um programa simples, utilizaremos um programa exemplo do próprio Arduino IDE, que é o Blink.

1. Acesse o menu “File --> Examples --> Basics --> Blink”. Com isso, será exibido em uma nova janela o sketch Blink.

```
void setup() {  
  pinMode(13, OUTPUT);  
}  
  
void loop() {  
  digitalWrite(13, HIGH);  
  delay(1000);  
  digitalWrite(13, LOW);  
  delay(1000);  
}
```

2. Clique no botão **Verify** e veja o output na área de erros (tela preta, parte inferior do IDE). Perceba que não é relatado nenhum erro.

3. Clique no botão **Upload** e veja o resultado. É esperado que o programa seja carregado sem erros e o LED conectado ao pino 13 comece a piscar, conforme ilustração a seguir.



Não se preocupe com o código exibido anteriormente, vamos detalhar todas as funções e comandos utilizados.

Principais funções e comandos

O Arduino possui vários comandos específicos, uns para os pinos digitais, outros para analógicos e funções básicas para o funcionamento do programa. Vamos detalhar tudo passo-a-passo para seu melhor entendimento. Para mais detalhes, acesse: <https://www.arduino.cc/reference/>.

Nesta apostila vamos definir modalidades para as funções, sendo assim, ficará mais organizado e fácil a introdução do conceito por detrás da função.

Essenciais

Abordaremos as funções essenciais para o Arduino funcionar, como a função **setup** e a **loop**.

Função setup()

A função `setup()` é essencial para a configuração inicial da placa Arduino. Ela é especial, pois é executada uma única vez, ou seja, quando seu código for embarcado no microcontrolador, a função `setup()` é chamada em primeiro lugar e será responsável por configurar os modos de pinos, interrupções, taxas de transferência serial, entre muitas outras. A função `setup()` é chamada toda vez que a placa Arduino é energizada ou resetada.

Função loop()

A função `loop()` é responsável por fazer a roda do Arduino girar, literalmente. Após a função `setup()` ser executada, a função `loop()` se inicia e executa, como nome sugere, um laço consecutivo. Isso permite que seu programa obtenha respostas, envie mensagens e mude conforme necessário, ou seja, controle o Arduino.

Com esse informação, você já deve ter decifrado o porque do sketch BLINK, falado anteriormente, conseguir ascender e apagar o LED inúmeras vezes mesmo tendo programado apenas uma única lógica. Ou seja, após a última linha codificada na função `loop()`, ela volta para o início e executa novamente.

Funções de temporização

Em projetos utilizando Arduino é muito recorrente a utilização de funções para retardarmos o laço recursivo existente devido à função **loop()**. Seja na leitura de um pino para evitar o *bouncing*, técnica conhecida como *debouncing*, que é a estabilização do sinal no pino para não capturar ruídos ou para piscar LEDs de tempos em tempos, sincronismo, entre outras aplicações.

Sendo assim, serão abordadas algumas funções interessantes relacionadas ao tempo e delay.

Função delay()

A função **delay()** exibida na seção “Primeiro Programa” é responsável por pausar o

laço da função **loop()** dado um parâmetro que representa uma quantidade de tempo (em milissegundos). Atente-se que a função **delay()** recebe um parâmetro, milissegundos.

Sua sintaxe é bem simples, veja:

delay(milissegundos);

Onde:

- **milissegundos** representa a quantidade em milissegundos da pausa;

Exemplo de código:

```
void setup() {           // função que é executada apenas 1x
  pinMode(13, OUTPUT);   // configura o pino 13 como SAÍDA (OUTPUT)
}
void loop() {
  digitalWrite(13,HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(500);
}
```

Não se preocupe com o entendimento total das funções exibidas no código acima, pois no decorrer desta apostila iremos revisitá-las para um melhor detalhamento.

Função millis()

Função responsável por retornar o número de milissegundos desde que o Arduino começou a executar programa atual. Tem sintaxe simples, pois não possui parâmetro. Além disso, o número de milissegundos irá estourar (*overflow*) após 50 dias, voltando ao zero.

Esta função tem papel interessante se utilizada em conjunto com a função `delay()`, vista acima, pois permite criar ações assíncronas durante a execução do `delay`. Claramente este é só um simples exemplo, pois o seu uso depende da aplicação em questão.

Veja, a seguir, a sintaxe da função `millis()`:

var_retorno = millis();

Note que, diferentemente da função `delay()`, a função `millis()` necessita de uma variável para guardar o valor de retorno. O tipo da variável deve ser **unsigned long**, conforme exemplo abaixo.

Exemplo de código:

```
unsigned long ret;           // variável declarada para guardar o retorno da func millis()
void setup() {               // função que é executada apenas 1x
    Serial.begin(9600);      // taxa de atualização de comunicação serial
}
void loop() {
    ret = millis();
    Serial.println("Tempo executado ate o momento: ");
    Serial.println(ret);
    delay(1000);
}
```

Para visualizar a saída reproduzida pelo código, abra o monitor serial. O número apresentado será o tempo transcorrido desde de que o arduino foi ligado.

Pinos digitais

A placa Arduino possui pinos digitais definidos de 0 - 13, conforme já destacado anteriormente, e esses pinos podem ser configurados como entradas - inputs - ou saídas - output. Pinos digitais trabalham com lógica booleana, ou seja, 0 ou 1, alto ou baixo, ligado ou desligado, não há meio termo no comportamento dos pinos digitais. Um simples exemplo é quando você clica no interruptor e acende uma luz, isso é nível alto, e quando clica novamente para apagar a luz, tem-se o nível baixo.

O microcontrolador do Arduino definido por padrão os pinos digitais como INPUTS, ou seja, não é necessário configurar explicitamente tal comportamento através da função pinMode(). Um detalhe importante sobre quando os pinos estão configurados como INPUT é que eles estão em um estado de alta-impedância, ou seja, pequenas variações de corrente mudam o estado do pino, o que proporciona benefícios para tarefas de leitura de sensores, entre outras. Porém, se o pino não estiver conectado a outros circuitos ou fios essa característica pode reportar respostas aleatórias devido aos ruídos do próprio ambiente.

Pinos configurados como OUTPUT possuem estado de baixa-impedância, ou seja, em sua saída, podem fornecer uma corrente dita como substancial para outros circuitos. Lembrando que a máxima corrente é de 40mA, suficiente para ascender LED, rodar pequenos motores ou outros sensores.

Função pinMode()

Uma vez definido os dois estados de um pino digital, precisamos falar ao

microcontrolador qual das duas características queremos utilizar. A função `pinMode()` possui esse poder e é utilizada dentro da função `setup()`, uma vez que é uma função de configuração.

A sintaxe da função é simples, tendo como parâmetros o pino escolhido e o tipo, e não possui retorno. Veja:

`pinMode(pino, modo);`

Onde:

- **PINO** representa o número do pino escolhido (0 a 13);
- **MODO** representa se o pino será de entrada ou saída, para isso deve utilizar INPUT ou OUTPUT.

Exemplo de código:

```
void setup() {  
  pinMode(13, OUTPUT); // configura o pino 13 como SAÍDA (OUTPUT)  
  pinMode(8, INPUT);   // configura o pino 8 como ENTRADA (INPUT)  
}
```

Com a escolha do comportamento do pino definida pela função `pinMode()`, conseguimos realizar escrita ou leitura desse pino.

Função `digitalWrite()`

A função `digitalWrite()` é utilizada quando um pino é definido como OUTPUT, ou seja, podemos “escrever” no pino um valor alto (high) ou baixo (low) de saída - ESTADO - para acionar/desligar algum componente, LED, motores, etc. Note que estamos definindo o comportamento do microcontrolador, ou seja, a função `digitalWrite()` é utilizada dentro da função `loop()`.

Sua sintaxe é mostrada abaixo, veja:

`digitalWrite(pino, ESTADO);`

Onde:

- **PINO** é o número do pino escolhido (0 a 13);
- **ESTADO** representa se o pino terá saída alta ou baixo (HIGH ou LOW).

Exemplo de código:

```
void setup() {  
    pinMode(13, OUTPUT);    // configura o pino 13 como SAÍDA  
    pinMode(9, OUTPUT);    // configura o pino 9 como SAÍDA  
}  
void loop() {  
    digitalWrite(13, HIGH); // pino 13 recebe valor ALTO  
    digitalWrite(9, LOW)   // pino 9 recebe valor BAIXO  
}
```

Função digitalWrite()

A função digitalWrite() é utilizada em conjunto com os pinos definidos como INPUT, ou seja, poderemos “LER” qual o estado do pino em questão. Lembre-se que, em pinos digitais, os estados podem assumir valores HIGH (1) ou LOW (0), apenas.

Como na função digitalWrite(), a função digitalWrite() também deve estar dentro do escopo da loop(). A seguir, é exibida a sintaxe.

var_retorno = digitalWrite(PINO);

Onde:

- **PINO** é o número do pino escolhido (0 a 13);
- **var_retorno** recebe o valor do pino, 0 ou 1, podendo ser do tipo **int**.

Um detalhe importantíssimo desta função é que precisamos guardar o valor de retorno dela em uma variável, esse retorno pode assumir 0 (LOW) ou 1 (HIGH).

Exemplo de código:

```
void setup() {
    pinMode(8, INPUT);      // configura o pino 8 como ENTRADA (INPUT)
    pinMode(13, OUTPUT);    // configura o pino 13 como SAÍDA
}

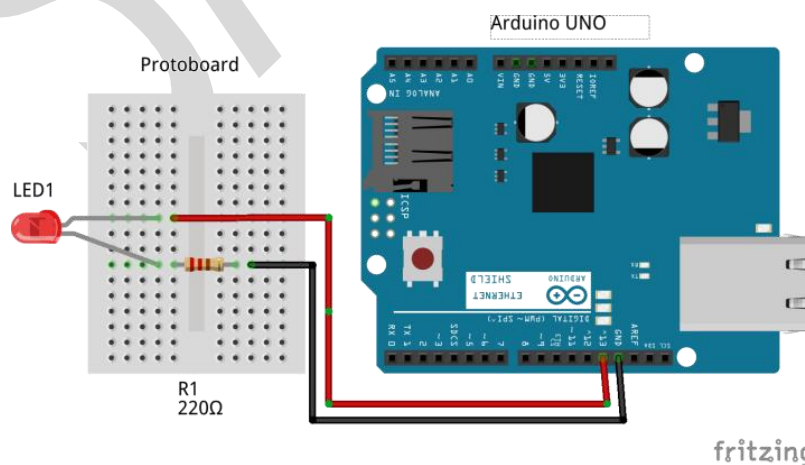
void loop() {
    int retorno;            // variável responsável por guardar o retorno da função
    retorno = digitalRead(8); // lembre-se que digitalRead() retorna 0 ou 1
    if (retorno) {          // testa se a variável "retorno" é TRUE
        digitalWrite(13, HIGH); // acende o LED do pino 13
        delay(1000);        // para o loop durante 1 segundo
    }
    else {
        digitalWrite(13, LOW); // caso a variável "retorno" seja FALSE, apaga o LED
    }
}
}
```

Programe você mesmo

Esta seção é destinada ao exercício das funções abordadas nas seções anteriores. Além disso, serão exibidos os esquemáticos dos circuitos, com isso, é esperado que você possa entender o básico de circuitos elétricos para fins didáticos.

Piscando um LED

O circuito abaixo representa o mínimo necessário para que você consiga acender um LED de forma segura através do seu Arduino.



São utilizados os seguintes componentes: Resistor - 220Ohms, LED, protoboard e

Arduino UNO.

Siga os passos abaixo para confeccionar o circuito:

1. Conecte as “pernas” do LED em duas linhas distintas da protoboard, conforme representado no circuito;
2. Utilize um fio para conectar a “perna” MAIOR do LED ao pino 13 do Arduino;
3. Conecte na “perna” MENOR do LED um resistor de 220Ohms;
4. Conecte o resistor de 220Ohms no GND da placa Arduino;
5. Conecte o Arduino no PC, via USB;
6. Abra o Arduino IDE e configure para reconhecer sua placa e a porta.

Utiliza-se o resistor de 220Ohms no circuito para garantir que a corrente drenada pelo circuito não ultrapasse o permitido pela placa Arduino e também não danifique o LED. Lembre-se de que os LEDs trabalham com correntes da ordem de 12 a 30mA, em caso de LEDs menores, esse valor é mais baixo.

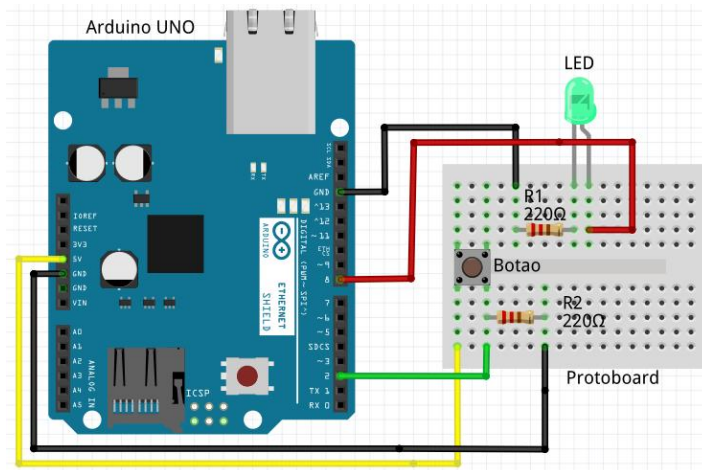
Uma vez confeccionado o circuito, vamos ao código. Abra o ARDUINO IDE e codifique as seguintes linhas:

```
void setup() {  
    pinMode(13, OUTPUT);    // configura o pino 13 como SAÍDA  
}  
  
void loop() {  
    digitalWrite(13, HIGH);  // manda nível lógico ALTO ao pino 13  
    delay(1000);             // define um tempo de espera de 1s  
    digitalWrite(13, LOW);   // manda nível lógico BAIXO ao pino 13  
    delay(1000);             // define um tempo de espera de 1s  
}
```

Após isso, clique no botão Verify, e se tudo estiver correto, no botão UPLOAD. É esperado que o LED conectado ao pino 13 pisque, de segundo em segundo.

Acendendo LED com botão

O circuito abaixo fornece o necessário para que você consiga acender/apagar um LED se pressionado o *pushbutton*.



Os componentes utilizados são: 2 resistores de 220Ohms, 1 LED, 1 *pushbutton* e o Arduino Uno.

Siga o circuito e conecte os fios necessários nos pinos correspondentes.

1. Pino GND do Arduino deve ser conectado no resistor de 220Ohms (R1);
2. Resistor R1 deve ser conectado na “perna” MENOR do LED;
3. Pino 8 deve ser conectado na “perna” MAIOR do LED;
4. Pino GND do Arduino deve ser conectado no resistor de 220Ohms (R2);
5. Resistor R2 deve ser conectado em uma das “pernas” do *pushbutton*;
6. Pino 5V deve ser conectado na “perna” oposta do *pushbutton*;
7. Pino 2 do Arduino deve ser conectado na “perna” do GND do *pushbutton*.

Após realizar as ligações, é hora de codificarmos nosso microcontrolador. O programa é simples, porém, já aborda conceitos básicos para confeccionar aplicações mais elegantes.

Abra o seu Arduino IDE e codifique as seguintes linhas:

```
int respBotao;           // variável que guardará o estado do botão
void setup() {
    pinMode(8, OUTPUT);   // configura o pino 8 como SAÍDA
    pinMode(2, INPUT);    // configura o pino 2 como ENTRADA
}
void loop() {
    respBotao = digitalRead(2); // lê o estado do botão
    delay(50);                // pequeno delay
    if ( respBotao == HIGH ) { // testa se o botão está pressionado
        digitalWrite(8, HIGH); // manda nível lógico ALTO ao pino 8
    }
    else { digitalWrite(8, LOW); } // caso o botão não estiver pressionado, manda nível
                                   // lógico BAIXO ao pino 8
}
```

Os comentários no código são apenas para explicar o que está ocorrendo em cada linha de código.

Após codificar, aperte **Verify** e, caso não houver erros, **Upload**. Agora que seu microcontrolador está pronto, é esperado que ao pressionar o botão, o LED acenda. Ao soltar o botão, ele apague.

Pinos analógicos

A placa Arduino possui pinos analógicos já destacados anteriormente, que vão de A0 - A5. Os pinos analógicos tem como principal função realizar leitura de dados de sensores analógicos, além disso, é possível utiliza-los como pinos de entrada e saída de uso geral, como os pinos digitais de 0 - 13.

Assim como você viu na seção de pinos digitais, os pinos analógicos (A0 - A5) podem ser utilizados para a mesma finalidade - eles podem trabalhar como pinos digitais -, então, veja um trecho de código que faz o uso dessa funcionalidade.

```
int pinLed = A1;           // define a variável pinLed para o pino A1
int pinBut = A0;           // define a variável pinBut para o pino A0
int leitura;               // define a variável para guardar a leitura do pino

void setup() {
    pinMode(pinLed, OUTPUT); // configura o pino A1 como SAÍDA
    pinMode(pinBut, INPUT);  // configura o pino A0 como ENTRADA
}

void loop() {
    leitura = digitalRead(pinBut); // realiza a leitura digital do pino A0
    // códigos
    digitalWrite(pinLed);          // realiza a escrita digital no pino A1
}
```

Aqui cabe um detalhe importante, os pinos analógicos somente são utilizados como INPUT - essa é uma configuração padrão -, para realizar a leitura dos sensores analógicos, não é necessário definir através da função *pinMode* que os pinos A0 - A5 são INPUT. Outro detalhe importante é que quando se tem a necessidade de simular saídas analógicas, o Arduino possui pinos especiais - a saber: 3, 5, 6, 9, 10 e 11 - que utilizam PWM. A definição e os detalhes mais específicos a respeito de PWM serão vistos mais adiante nesta apostila.

Agora vamos voltar ao principal uso dos pinos analógicos, ou seja, leituras analógicas. Você já viu na seção anterior que entradas digitais assumem apenas dois estados, alto ou baixo, mas as entradas analógicas não. Como falado anteriormente, acender ou

apagar a luz representa apenas dois estados, mas você já deve ter notado alguma vez que existem meios de controlar a intensidade do brilho da luz, para isso é necessário leituras analógicas. Leituras analógicas - como: temperaturas, intensidade de brilho luminoso, pressão, etc - variam com o tempo e podem assumir diversos valores dentro de uma faixa.

Você já deve ter imaginado que o Arduino trabalha internamente apenas com dados digitais, pois você está na era digital, se você imaginou isso, acertou. Para que o Arduino consiga trabalhar com leituras analógicas, é necessário um conversor capaz de transformar algo analógico em digital, conhecido como Analógico-Digital (AD).

Conversor Analógico-Digital

O conversor AD é capaz de transformar o sinal analógico recebido pelos pinos A0 - A5 e transformá-los em digitais, ou seja, definir um valor único para esse sinal dada uma tensão de entrada no pino analógico. Essa conversão leva em consideração a resolução da porta que o microcontrolador possui, no caso do microcontrolador do Arduino Uno a resolução é de 10 bits. A fórmula completa que faz a conversão é a mostrada abaixo.

$$\text{Resolução} = V_{\text{Referência}} / 2^n$$

Onde:

- $V_{\text{Referência}}$ - Tensão de referência do conversor AD
- n - Resolução do microcontrolador

Suponha que a tensão referência do conversor AD seja de 5V, para o Arduino Uno, que tem uma resolução de 10 bits, tem-se:

$$\begin{aligned}\text{Resolução} &= 5 / 2^{10} \\ &= 5 / 1024 \\ &= 4.88\text{mV}\end{aligned}$$

Ou seja, a cada 4.88mV é possível incrementar um nível de resolução ao conversor AD do Arduino, simplificando, os valores entre 0 - 5V são mapeados para 0 - 1023. Se você quiser saber mais sobre conversor AD, é sugerido que você busque pelo *datasheet* do microcontrolador ATmega328.

Função analogRead()

A função ***analogRead*** é capaz de ler o valor do pino analógico que é passado por parâmetro. Veja a sintaxe abaixo

var_retorno = analogRead(PINO);

Onde:

- **PINO** é o número do pino escolhido (A0 a A5);
- **var_retorno** recebe o valor do pino analógico **int** (entre 0 e 1023).

O valor retornado pela função **analogRead** é mapeado, através do conversor AD, entre as tensões de 0 e tensão de referência (Ex.: 5V) para valores inteiros entre 0 e 1023 - resolução de 10 bits do AD. Veja um simples exemplo de código.

```
int pinPot = A3;           // variável que referencia o pino analógico de leitura
int respPot;              // variável que guardará o valor da entrada analógica

void setup() {
}

void loop() {
    respPot = analogRead(pinPot); // lê o valor do sensor conectado ao pino A3
    delay(1000);                 // delay de 1s
}
```

É importante ter em mente que se o pino analógico não possuir conexão, a leitura deste com a função **analogRead** retornará valores flutuantes devido a diversos fatores. Além disso, não é necessário especificar o comportamento do pino A3 como INPUT na função **setup()**, isso é *default*.

Função analogWrite()

Como você viu, grandezas que variam com o tempo são ditas analógicas e você é capaz de realizar a leitura dessas grandezas (sensores) pelo Arduino através da função **analogRead** - isso ficará mais claro na próxima seção. Mas, se é possível “ler” essas grandezas, como realizar uma escrita? Como definir a intensidade do brilho de um LED, por exemplo? Através da função **analogWrite**. A sintaxe desta função é:

analogWrite(PINO, VALOR);

Onde:

- **PINO**: Pinos do Arduino (Uno: 3, 5, 6, 9, 10 e 11);
- **VALOR**: Pulso PWM (*duty cycle*), que varia entre 0 (desligado) - 255 (ligado);

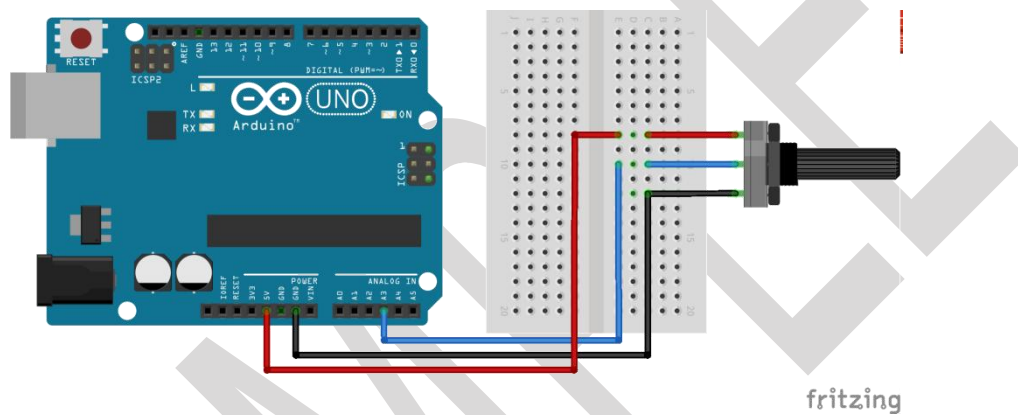
Essa função simula a escrita de uma grandeza analógica, utilizando uma onda **PWM (Pulse Width Modulation)**, no pino do Arduino. Veja no **Apêndice A**, desta apostila, o conceito de PWM. É extremamente importante saber que, os pinos que comportam sinais PWM são: 3, 5, 6, 9, 10, e 11.

Programe você mesmo

Esta seção é destinada ao exercício das funções abordadas nas seções anteriores. Além disso, serão exibidos os esquemáticos dos circuitos, com isso, é esperado que você possa entender o básico de circuitos elétricos para fins didáticos.

Leitura de potenciômetro

Para utilizar a função ***analogRead***, você deve montar o circuito esquematizado abaixo. Você precisará de apenas um potenciômetro, fios e o Arduino UNO.



O potenciômetro possui três conectores (“perninhas”), as quais devem ser conectadas seguindo o esquemático acima. Os conectores das extremidades são conectados ao **GND** e ao **5V** do Arduino, respectivamente. O conector central é responsável por enviar o sinal analógico ao pino **A3** do Arduino.

Abra a IDE Arduino e codifique as seguintes linhas de código.

```
int pinPot = A3;           // variável que referencia o pino analógico de leitura
int respPot;               // variável que guardará o valor da entrada analógica

void setup() {
    Serial.begin(9600);     // inicia uma comunicação serial
}

void loop() {
    respPot = analogRead(pinPot); // lê o valor do potenciômetro (0 - 1023)
    delay(1000);            // delay de 1s
    Serial.println(respPot); // mostra o valor da variável no Monitor Serial
}
```

Os comandos envolvendo comunicação serial serão detalhes mais adiante. Por hora, varie a resistência do potenciômetro e veja a saída no monitor serial, note que os valores de saída serão entre 0 e 1023 - atualizados a cada 1s -, isso demonstra o conversor AD

em pleno funcionamento. O que isso representa? O valor da tensão liberada pelo potenciômetro discretizado, ou seja, os valores entre 0 e 5V mapeados para 0 e 1023.

Como saber o valor real da tensão? Simples, utilizando a equação matemática mostrada na seção Analógico-Digital. Indo direto ao ponto, codifique as linhas de código a seguir.

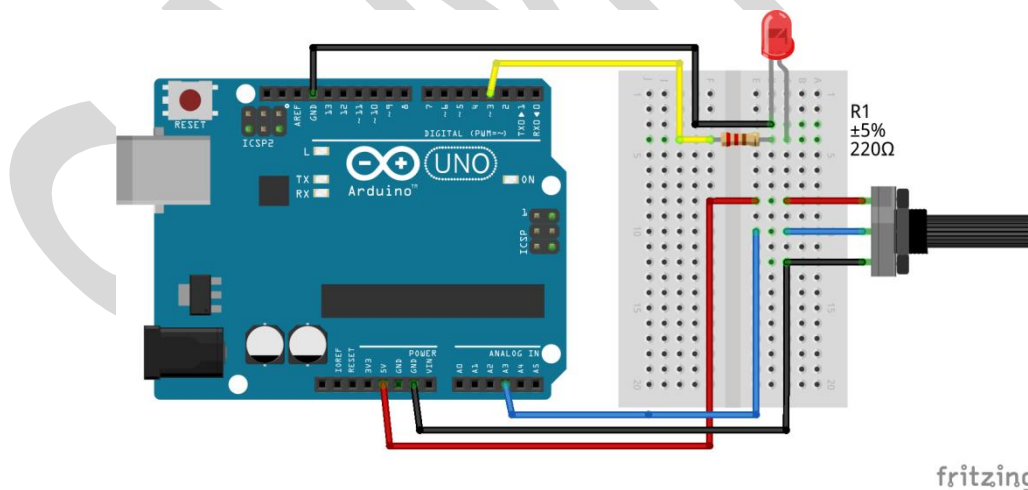
```
int pinPot = A3;           // variável que referencia o pino analógico de leitura
int respPot;               // variável que guardará o valor da entrada analógica

void setup() {
    Serial.begin(9600);     // inicia uma comunicação serial
}

void loop() {
    respPot = analogRead(pinPot); // lê o valor do potenciômetro
    float t = respPot * (5.0 / 1024) // converte a leitura analógica em tensão (0 - 5V)
    delay(1000);           // delay de 1s
    Serial.println(t);     // mostra o valor da variável no Monitor Serial
}
```

Controlando o brilho do LED

Para utilizar a função ***analogWrite***, você deverá codificar o exemplo mostrado abaixo. Primeiro, veja o esquemático do circuito.



Os componentes utilizados serão: 1 resistores de 220Ohms, 1 LED, 1 potenciômetro e o Arduino Uno. Siga o esquemático para conectar os fios corretamente. Após isso, codifique o código abaixo.

```
int pinPot = A3;           // variável que referencia o pino analógico de leitura
int led = 3;               // variável que referencia o output para o LED
int respPot = 0;           // variável que guardará o valor da entrada analógica

void setup() {
    pinMode(led, OUTPUT);   // configura o pino como saída
}

void loop() {
    respPot = analogRead(pinPot); // lê o valor do potenciômetro
    analogWrite(led, respPot / 4); // mapeia o valor de respPot (0 - 1023) para
                                   // o PWM correspondente (0 - 255)
}
```

Comunicação serial

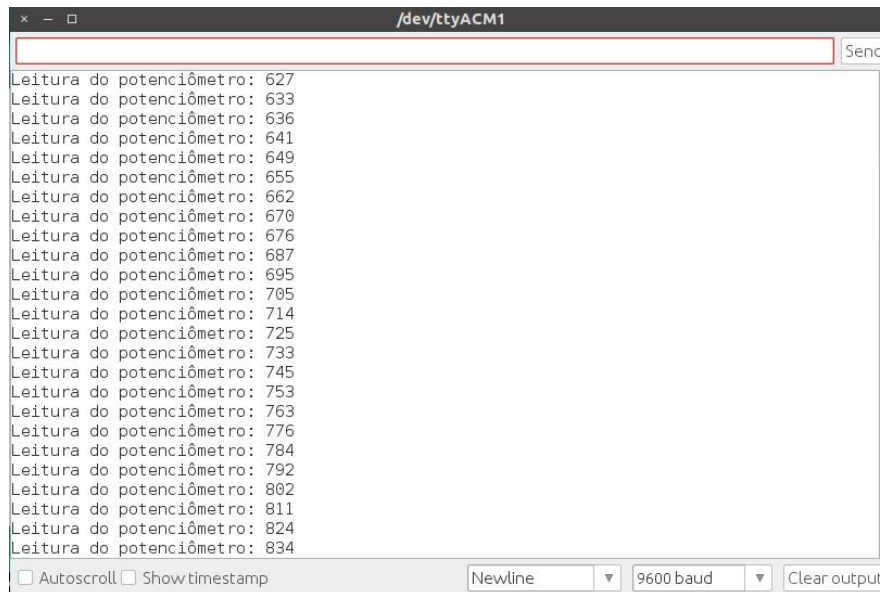
Na seção anterior que você aprendeu a capturar os sinais analógicos do potenciômetro, e deve ter notado que foi utilizado funções relacionadas à **serial** e ao **monitor serial**. A comunicação serial, portanto, é utilizada para fazer com que a placa Arduino converse com o computador (Ex.: Arduino IDE) ou outros dispositivos.

O canal de comunicação serial do Arduino está ligado aos pinos digitais 0 (RX) e 1 (TX). De maneira geral, são esses pinos que entendem e manipulam o sinal TTL (lógica transístor-transístor) para que a comunicação ocorra entre o USB e seu computador, esse sinal tem tensão de 5V.

Existem variadas funções que cobrem o assunto de comunicação serial via Arduino, mas nesta apostila será tratadas as principais. Se você quiser saber mais, consulte as referências deste assunto no site Arduino (arduino.cc).

Monitor serial

A IDE Arduino disponibiliza um monitor para auxiliar na visualização dos dados enviados e recebidos feitos via serial. Essa ferramenta é simples de se utilizar e possui uma função de configuração que define a taxa de envio dos dados, chamada de **baud rate**. Veja abaixo uma imagem da ferramenta, que pode ser acessada pelo menu superior direito da IDE Arduino.



Por padrão, o **baud** do monitor é definido em 9600 *bits* por segundo (bps). Assim, para que a comunicação ocorra de forma correta entre o dispositivo e a IDE, é necessário configurar a taxa da placa Arduino, que pode receber diversos valores (Ex.: 4800, 9600, 19200, 38400).

Função Serial.begin()

Para realizar a configuração de **baud** do Arduino, é necessário utilizar a função **Serial.begin**. Você já notou que para visualizar os dados do potenciômetro, na seção anterior, foi necessário utilizar essa função, essa função é definida dentro da função **setup**, e tem a seguinte sintaxe:

Serial.begin(velocidade);

Onde:

- **velocidade**: Define o *baud* (bps) da comunicação;

Veja um exemplo de código:

```
void setup() {
    Serial.begin(9600);    // inicia uma comunicação serial com 9600bps
}

void loop() {
    // mostra o valor da variável no Monitor Serial
    Serial.println("Comunicação inicializada!");
}
```

Você poderá visualizar a saída da frase pelo monitor serial.

Aguardando conexão com a porta serial

Em algumas aplicações é necessário garantir que a conexão com a porta serial tenha sido estabelecida, porém, essa conexão pode, por muitas vezes, demorar. Uma maneira simples de verificar ou aguardar que a conexão seja estabelecida é utilizando estruturas de testes na função **setup**.

Veja um possível código que aguarda até que a porta serial seja conectada. Lembre-se, a função **setup** é executada apenas uma única vez, ou seja, se você garantir que a função aguarde até que a comunicação seja estabelecida, a função **loop não será executada**.

```
void setup() {  
    Serial.begin(9600);    // inicia uma comunicação serial com 9600bps  
    while(!Serial);       // Enquanto a porta serial não é conectada, aguarde  
}  
void loop() {  
    // mostra o valor da variável no Monitor Serial  
    Serial.println("Comunicação inicializada!");  
}
```

Função Serial.print()

Função responsável por imprimir dados na porta serial - os dados possuem formatos legíveis (texto em ASCII). Essa função pode imprimir muitos formatos, como números inteiros, decimais (float) com precisão de duas casas por padrão, textos, etc.

Sua sintaxe é definida a seguir:

Serial.print(valor);

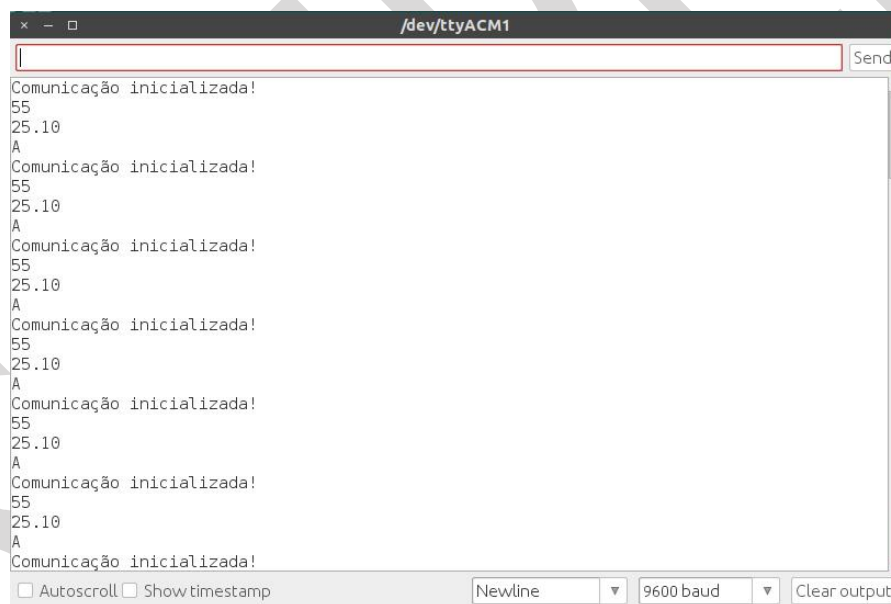
Onde:

- **valor**: Valor impresso na serial (qualquer tipo: int, float, string)

Veja um exemplo de código abaixo, seguido de sua saída no monitor.

```
void setup() {  
    Serial.begin(9600);    // inicia uma comunicação serial com 9600bps  
}  
void loop() {  
    Serial.print("Comunicação inicializada!\n"); // imprime a frase na serial  
    Serial.print(55);                          // imprime o número 55 na serial  
    Serial.print("\n");                        // imprime uma quebra de linha na serial  
    Serial.print(25.0986);                     // imprime o valor 25.10 na serial  
    Serial.print("\n");                        // imprime uma quebra de linha na serial  
    Serial.print('A');                         // imprime a letra A na serial  
    Serial.print("\n");                        // imprime uma quebra de linha na serial  
}
```

A saída do código acima pode ser visualizada no monitor serial e deve ser algo como a imagem abaixo.



Perceba que para números com casas decimais, houve arredondamento, pois o padrão é de apenas dois dígitos.

Função **Serial.println()**

A função **Serial.println** possui as mesmas características da função mostrada acima, **Serial.print**, com o adicional de já inserir a quebra de linha após o valor impresso na serial.

Sua sintaxe é definida a seguir:

Serial.println(valor);

Onde:

- **valor**: Valor impresso na serial (qualquer tipo: int, float, string)

Veja um exemplo de código.

```
void setup() {  
    Serial.begin(9600);    // inicia uma comunicação serial com 9600bps  
}  
void loop() {  
    Serial.println("Comunicação inicializada!"); // imprime a frase na serial  
    Serial.println(55);                          // imprime o número 55 na serial  
    Serial.println(25.0986);                      // imprime o valor 25.10 na serial  
    Serial.println('A');                          // imprime a letra A na serial  
}
```

A saída deve ser exatamente a mesma já mostrada na função **Serial.print**.

Formatando as saídas: Serial.print() e Serial.println()

Em alguns casos é interessante formatar a saída das funções vistas anteriormente, ou seja, um número exibido na forma decimal pode ser mostrado em hexadecimal, ou até mesmo binário. Ambas as funções **Serial.print** e **Serial.println** suportam essa formatação. Veja como é simples a sintaxe:

Serial.print(valor, formato);
Serial.println(valor, formato);

Onde:

- **valor**: Valor impresso na serial (qualquer tipo: int, float, string)
- **formato**: Especifica a base numérica para tipos (int) e o número de casas decimais para (float). As bases podem ser: DEC, HEX, OCT, BIN.

Veja exemplos das saídas formatadas no código abaixo.

```
void setup() {
    Serial.begin(9600);    // inicia uma comunicação serial com 9600bps
}

void loop() {
    Serial.println(55, BIN);           // imprime o número 55 em binário
    Serial.println(25.0986, 4);        // imprime o valor 25.0986 na serial
    Serial.println(10, HEX);          // imprime o número 10 em hexadecimal
}
```

Função Serial.available()

A função **Serial.available** tem por objetivo retornar a quantidade inteira de *bytes* (caracteres) contidos no *buffer* de leitura existente na comunicação serial. Esse *buffer* possui um tamanho padrão de 64 bytes.

Você viu nas seções anteriores como enviar dados do Arduino para o monitor serial, mas e se você quiser receber dados do monitor para tratar no Arduino (no seu código)? É nesse momento que o *buffer* deve ser verificado.

Sua sintaxe é bem simples, veja só:

```
int retorno = Serial.available();
```

Onde:

- **int retorno**: Variável que guardará a quantidade de bytes do buffer;

Em muitos casos essa variável pode ser omitida, sendo comum utilizar essa função da seguinte forma, conforme mostrado no código abaixo:

```
void setup() {
    Serial.begin(9600);    // inicia uma comunicação serial com 9600bps
}

void loop() {
    if (Serial.available() > 0) {
        Serial.println("Há dados no buffer");
    }
}
```

Isso garante que só haverá leitura dos dados se existir algum dado no *buffer* para ser lido, caso contrário, não há a necessidade.

Função `Serial.read()`

Essa função tem por característica ler o **primeiro** byte contido no buffer (byte mais antigo). Ao contrário das funções **`Serial.print`** e **`Serial.println`**, essa função trata os dados em bytes, isso pode ser útil caso você estiver trabalhando com protocolos de comunicações (padrões de mensagens) mais específicos ou criados por você. Sua sintaxe é simples, veja.

retorno = `Serial.read()`;

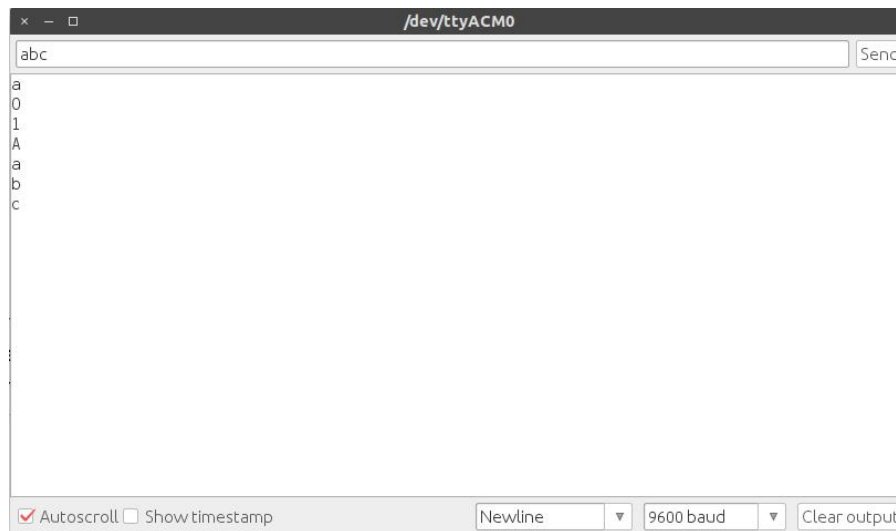
Onde:

- **retorno**: variável que guardará o retorno da função (byte).

Veja um exemplo de código que é capaz de ler uma letra digitada no monitor serial.

```
void setup() {  
    Serial.begin(9600);    // inicia uma comunicação serial com 9600bps  
}  
void loop() {  
    if (Serial.available()) {  
        char r = Serial.read(); // leitura do primeiro byte do buffer  
        if(r != '\n') {        // verifica se o byte lido não é um "pula linha" (enter)  
            Serial.println(r); // imprime no monitor serial o caracter  
        }  
    }  
    delay(1000);            // atraso de 1s para verificar novas letras  
}
```

Note que o exemplo acima é capaz de ler e exibir apenas uma única letra por vez no monitor serial - mesmo que você digite uma palavra, isso ocorre pois o comando **`Serial.read()`** retorna apenas o **primeiro byte** do buffer. Um detalhe importante é que há um comando de verificação (***if***) que testa se o byte lido é diferente de **`'\n'`** (nova linha ou enter), isso ocorre pois, ao digitar letras ou palavras no monitor serial o símbolo **`'\n'`** fica remanescente no buffer. Na seção "Programe você mesmo" você terá exemplos mais completos sobre essa função. Uma parte da saída do programa acima pode ser vista na figura a seguir.



Quando digitado a palavra “abc” no monitor serial, as letras aparecem de forma individual na saída do monitor, pois cada letra (tipo char) tem tamanho de 1 byte e o retorno da função **Serial.read()** recupera apenas essa mesma quantidade.

Função **Serial.write()**

Essa função é capaz de escrever dados binários diretamente na porta serial. Os dados são enviados através de bytes, uma vez que outros dispositivos trocam informações na forma binária, ou seja, se você quiser enviar ou visualizar dados representados simbolicamente em ASCII, você deverá utilizar as funções de **Serial.print** e **Serial.println**. Quando utilizada, a função **Serial.write** retornará o número de bytes escritos na porta serial, a leitura desse número é facultativa, seguindo o padrão da função **Serial.available()**. Veja a sintaxe desta função abaixo.

```
retorno = Serial.write(dado);  
Serial.write(dado);
```

Onde:

- **retorno**: Variável que guardará a quantidade de bytes enviados pela serial (facultativo);
- **dado**: Informação a ser enviada através da porta serial (bytes).

Veja um exemplo simples que mostra a diferença entre os comandos **Serial.write** e **Serial.print**. Baseado na tabela ASCII (Apêndice A - Tabela ASCII), o programa exibirá símbolos gráficos visíveis, cujo range é de 33 a 126.

```
// Função executada apenas a primeira vez
void setup() {
    // Inicializa a comunicação serial com uma taxa de 9600 bauds.
    Serial.begin(9600);

    Serial.println("Comando print \t=\t Comando write");

    // Range de símbolos visíveis da tabela ASCII
    for (int i = 33; i <= 126; ++i) {
        Serial.print(i);    // Representação em ASCII
        Serial.print(" \t=\t ");
        Serial.write(i);    // Representação em byte
        Serial.println();
    }
}

// Função loop não terá nada a ser executado
void loop() {
}
```

Veja parte da saída do código acima.

```
Comando print = Comando write
33 = !
34 = "
35 = #
36 = $
37 = %
38 = &
39 = '
40 = (
41 = )
42 = *
43 = +
44 = ,
45 = -
46 = .
47 = /
48 = 0
49 = 1
50 = 2
51 = 3
52 = 4
53 = 5
54 = 6
55 = 7
56 = 8
57 = 9
58 = :
59 = ;
60 = <
61 = =
62 = >
```

Programe você mesmo

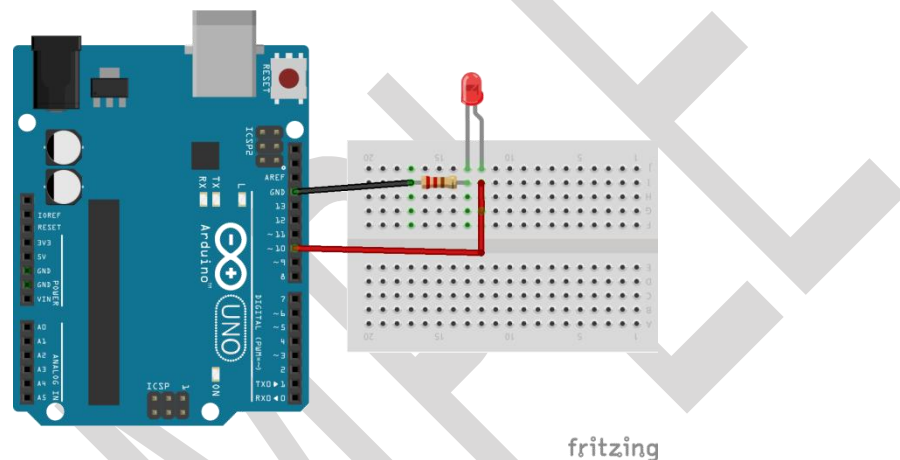
Esta seção é destinada ao exercício das funções abordadas nas seções anteriores. Além disso, serão exibidos os esquemáticos dos circuitos, com isso, é esperado que você

possa entender o básico de circuitos elétricos para fins didáticos.

Controlando o brilho do LED através da serial

Como você viu nas seções passadas, é possível abrir um canal de comunicação entre o Arduino e outros dispositivos, que neste exemplo será o Arduino IDE. Neste exemplo você será capaz de enviar dados pelo monitor serial, controlando o brilho de um LED, que é feito através do método de PWM (Veja o Apêndice A - PWM).

O circuito utilizado pode ser visto a seguir.



Neste exemplo, o controle do brilho não será com potenciômetro, mas sim com dados enviados pela comunicação serial. Sendo os dados em PWM, a informação transmitida pode ser de 0 a 255, diretamente via monitor serial. Veja o código abaixo.

```
int ret;           // variável que guardará o retorno da serial

int pwm;          // variável que guardará o PWM do led

void setup() {
    // Inicializa a comunicação serial com uma taxa de 9600 bauds.
    Serial.begin(9600);
    pinMode(10, OUTPUT);
}

void loop() {
    if (Serial.available()) {
        ret = Serial.parseInt(); // realiza a leitura do valor do monitor serial
                                   // já convertido para inteiro
    }

    if (ret) { // se o valor for superior ao 0
        pwm = ret; // guarda o PWM
        Serial.print("LED aceso com PWM de: ");
    }
}
```

```

    Serial.println(pwm);
    analogWrite(10, pwm);
  }
  delay(1000);
}

```

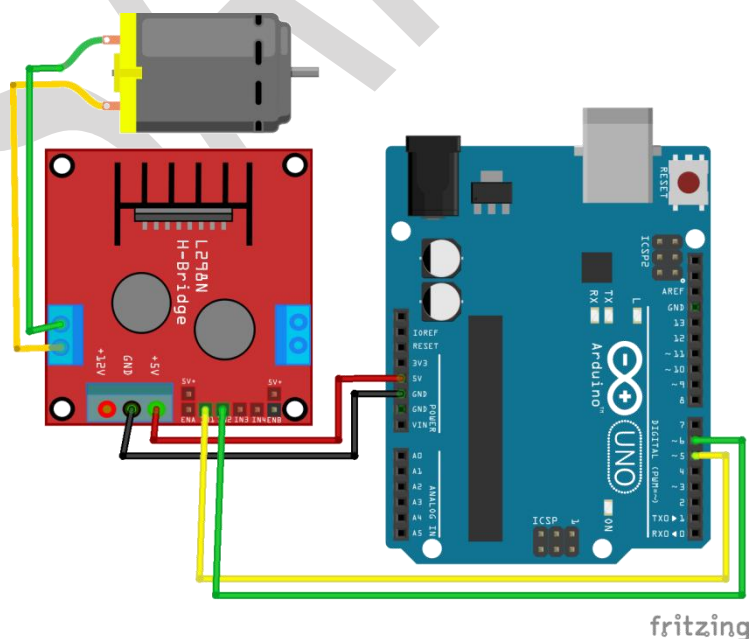
Aqui você viu algo novo, a função **Serial.parseInt()**, que tem como objetivo ler valores inteiros válidos do buffer, exatamente como o PWM é.

Como desafio, tente implementar a lógica que desliga o LED. Além disso, como um *plus*, tente realizar o controle do brilho do LED utilizando a função **Serial.read()**, será necessário unir byte a byte dos valores passados via monitor serial em uma palavra (tipo char ou string), após isso, converter essa palavra para inteira.

Controlando o sentido de giro do motor DC

Através do serial é possível passar as informações para o Arduino a fim de controlar o sentido de giro de um motor DC. A ideia é criar um programa capaz de identificar uma palavra reservada (um protocolo) que faça o motor girar em um sentido e outra palavra reservada que faça o motor girar no sentido oposto, fica o desafio para você mesmo criar uma palavra reservada que faça o motor se manter parado. Para os sentidos do foram definidas as palavras: se1 e se2. Os giros do motor são controlados através de um módulo de Ponte H (Veja o Apêndice A - Ponte H).

Veja o esquemático do circuito proposto. Neste circuito é utilizado uma ponte H L298N (Leia o Apêndice A - Ponte H) e um motor DC de 5V. Os pinos que receberão o PWM serão o IN1 (Pino 5) e IN2 (Pino 6).



fritzing

Um detalhe importantíssimo é que a ponte H L298N possui um jumper que ativa 5V (Leia o Apêndice A - Ponte H), garanta que esse jumper esteja **desconectado**, pois a alimentação da ponte H será fornecida pelo próprio Arduino.

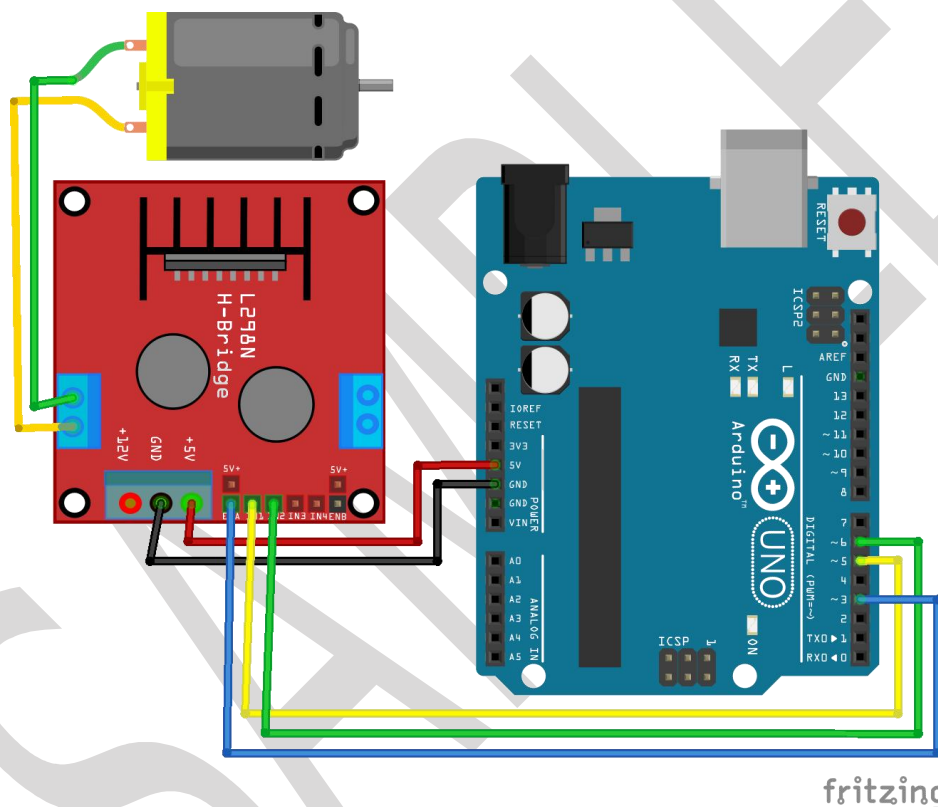
Veja o código capaz de definir o sentido de rotação do motor.

```
String sentido = "";
void setup() {
    // Inicializa a comunicação serial com uma taxa de 9600 bauds.
    Serial.begin(9600);
    pinMode(5, OUTPUT);
    pinMode(6, OUTPUT);
}
// Função loop não terá nada a ser executado
void loop() {
    if (Serial.available()) {
        // enquanto tiver bytes no buffer, realiza a leitura do buffer
        while (Serial.available()) {
            char r = Serial.read();
            // realiza a unir das letras para formar (se1 ou se2)
            if (r != '\n') sentido.concat(r);
        }
        delay(10);
    }
    // caso se1, manda um PWM ao pino 3;
    if (sentido.equals("se1")) {
        Serial.println("Lado 1");
        analogWrite(5, 255); // Manda PWM máximo ao pino 5
        analogWrite(6, 0);   // Manda PWM mínimo ao pino 6
    }
    // caso se2, manda um PWM ao pino 3;
    if (sentido.equals("se2")) {
        Serial.println("Lado 2");
        analogWrite(5, 0);   // Manda PWM mínimo ao pino 5
        analogWrite(6, 255); // Manda PWM m ao pino 6
    }
    // limpa a variável para, se existir, ler o sentido oposto
    sentido = "";
    delay(1000);
}
```


Controlando a velocidade e a rotação do motor DC

Este exemplo é uma continuação do exemplo anterior, que além de controlar o sentido do motor, será controlado a velocidade de rotação. As velocidades foram especificadas em três faixas de PWM, mínima: 20; média: 120; máxima: 255. Para cada faixa, foi definido uma palavra reservada (protocolo), veja: mínima: min; média: med; máxima: max. Além das já conhecidas se1 e se2 para controle de sentido.

Veja o esquemático do circuito abaixo. Ainda é utilizado a ponte H L298N, Arduino UNO e um motor DC de 5V. A alimentação do módulo da ponte H é feita pelo próprio Arduino.



Os pinos que controlarão o sentido do motor são IN1 (pino 5) e IN2 (pino 6); O pino que controlará a velocidade de rotação é o ENA (pino 3). Todos os pinos trabalharão com PWM. O código abaixo realiza esses controles através do envio das palavras definidas via monitor serial.

```

String palavra = "";
void setup() {
    // Inicializa a comunicação serial com uma taxa de 9600 bauds.
    Serial.begin(9600);
    pinMode(3, OUTPUT);
    pinMode(5, OUTPUT);
    pinMode(6, OUTPUT);
}
// Função loop não terá nada a ser executado
void loop() {
    if (Serial.available()) {
        // enquanto tiver bytes no buffer, realiza a leitura do buffer
        while (Serial.available()) {
            char r = Serial.read();
            // realiza a unir das letras para formar (se1; se2 ou min; med; max)
            if (r != '\n') palavra.concat(r);
        }
        delay(10);
    }
    // caso se1, ativa PWM máximo no pino 5 e mínimo no pino 6;
    if (palavra.equals("se1")) {
        Serial.println("Lado 1");
        analogWrite(5, 255); // Manda PWM máximo ao pino 5
        analogWrite(6, 0);   // Manda PWM mínimo ao pino 6
    }
    // caso se2, ativa PWM máximo no pino 6 e mínimo no pino 5;
    if (palavra.equals("se2")) {
        Serial.println("Lado 2");
        analogWrite(5, 0);   // Manda PWM mínimo ao pino 5
        analogWrite(6, 255); // Manda PWM m ao pino 6
    }
    // caso min, manda PWM de 10 no pino 3
    if (palavra.equals("min")) {
        Serial.println("Velocidade de rotação mínima");
        analogWrite(3, 20);
    }
    // caso med, manda PWM de 120 no pino 3
    if (palavra.equals("med")) {
        Serial.println("Velocidade de rotação média");
        analogWrite(3, 120);
    }
}

```

```
// caso max, manda PWM de 255 no pino 3
if (palavra.equals("max")) {
    Serial.println("Velocidade de rotação máxima");
    analogWrite(3, 255);
}
// limpa a variável para, se existir, ler a nova palavra
palavra = "";
delay(1000);
}
```

Apêndice A

Neste apêndice você encontrará um pouco mais sobre o que foi utilizado nas seções de Programe você mesmo.

Tabela ASCII

A tabela ASCII (acrônimo de *American Standard Code for Information Interchange*; ou Código Padrão Americano para Intercâmbio de Informação) nada mais é do que códigos binários que representam uma série de símbolos, os símbolos são divididos em dois tipos: de controle e gráficos.

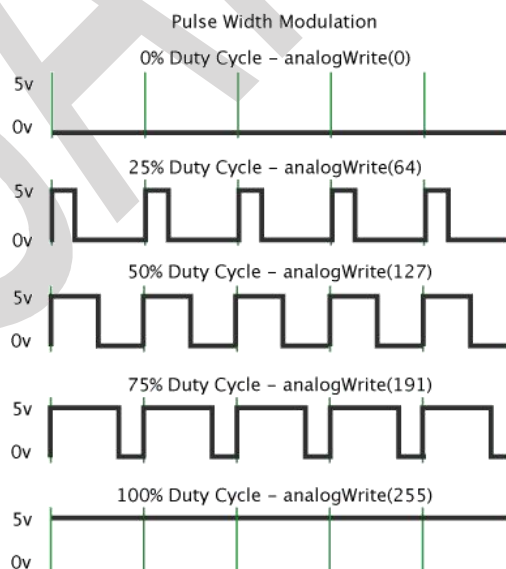
Símbolos de controle possuem um range de 0 a 31, e em especial o 127. Símbolos de controle são necessários para fazer com que a comunicação entre dispositivos, canais de comunicação serial, etc, consigam identificar as “palavras” transmitidas de maneira correta, por exemplo: início de texto e final da texto. Um símbolo de controle muito conhecido quando você inicia na programação é o nulo (0 ou \0), esse símbolo está presente em finais de palavras (tipo char), significando o seu final.

Símbolos gráficos ou visuais são os já conhecidos entre nós, que são: letras, caracteres especiais, entre outros; O range dos símbolos gráficos é de 32 a 126. O caractere espaço em branco, por exemplo, é representado pelo decimal 32, já a letra ‘A’ é definido pelo decimal 65. Veja a tabela ASCII abaixo.

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	\$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(72	48	110	H	104	68	150	h
9	9	11		41	29	51)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	
29	1D	35		61	3D	75	=	93	5D	135]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

PWM (Modulação por Largura de Pulso)

Uma onda PWM, em linhas gerais, é uma forma de se conseguir representar grandezas analógicas por meios digitais. O PWM é representado por uma onda quadrada com padrão de pulsos ligado-desligado, o tempo que essa onda mantém o padrão ligado a cada ciclo regular é representa a **largura do pulso**. Veja a representação abaixo, retirada do próprio site do Arduino (arduino.cc), note que a imagem mostra ondas com valores entre 0 - 255 na função **analogWrite**.



Uma vez definido a largura de pulso da onda, se você é capaz de repetir esse padrão

rápido o suficiente para manter um LED aceso, o resultado final será um valor entre 0 - 5V, que é capaz de controlar o brilho desse LED. A frequência de repetição desses padrões, no Arduino UNO, é de aproximadamente 490Hz para os pinos 3, 9, 10 e 11; e aproximadamente 980Hz para os pinos 5 e 6.

Para os pinos com frequências de 490Hz, o intervalo entre as linhas é de aproximadamente 2 milissegundos.



Para os pinos com frequências de 980Hz, o intervalo entre as linhas é de aproximadamente 1 milissegundo.

O **duty cycle** é a relação do tempo em que o pulso fica ativo, dividido pelo ciclo completo (ou período) da onda - o que representa uma certa porcentagem em relação ao período completo. O **ciclo completo (ou período)** é dado em segundos e identificado pelo intervalo entre as bordas de subida de um pulso (linhas verdes).



Como encontrar o *duty cycle*?

$$\text{Duty cycle} = (\text{Pulso Ativo} / \text{Ciclo}) * 100$$

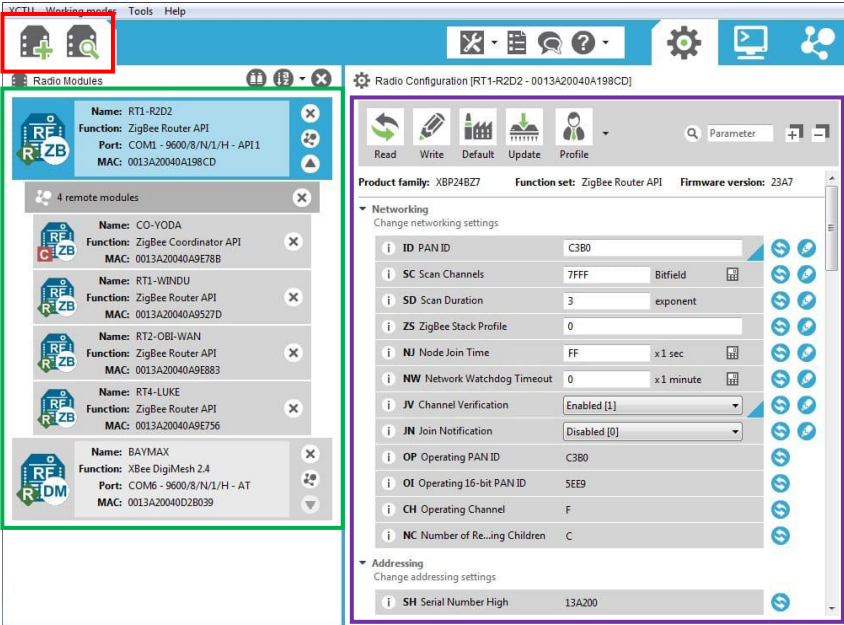
Agora você é capaz de entender que o valor passado para a função **analogWrite** é o **duty cycle** do PWM, o que resultará em um valor do **pulso ativo** no pino

XCTU

O software XCTU é uma plataforma gratuita que permite configuração de módulos e redes sem-fio. O XCTU entrega uma visualização gráfica da rede, o que permite configurar e gerenciar facilmente os dispositivos da rede, como o módulo Xbee.

Permite adicionar e pesquisar dispositivos conectados ao computador.

Dispositivos encontrados ao estarem conectados com o computador. Uma vez exibidos, os dispositivos podem ser configurados para serem, em linhas gerais, coordenadores e end-devices em rede, assim, é possível comunicar dois ou mais dispositivos por troca de mensagens.



The screenshot shows the XCTU software interface. On the left, the 'Radio Modules' list displays several connected devices, including RT1-R2D2, CO-VODA, RT1-WINDU, RT2-OB1-WAN, RT4-LUKE, and BAYMAX. On the right, the 'Radio Configuration' window for RT1-R2D2 is open, showing various settings for the ZigBee Router API, including network parameters like PAN ID, Scan Channels, and Stack Profile.

Janela de configuração dos dispositivos. Nesta janela é possível definir como os dispositivos vão trabalhar, ou seja, qual o nome da rede que se conectarão, o nome do dispositivo na rede (NI Node Identification), qual o seu ID na rede (ID PAN ID), quais canais devem acessar (SC Scan Channels), qual o tempo de varredura para encontrar redes disponíveis (SD Scan Duration), o modo de endereçamento que será configurado (DL e DH destination adress), entre outras diversas opções. O módulo Xbee permite diversas configurações a fim de garantir que a comunicação sem fio seja estabelecida e mantida sem interferências e com baixo consumo de energia, portanto, é um assunto longo, que não será coberto nesta apostila.

Esse software é compatível com Windows, Linux e MacOS, sendo de fácil instalação, para saber mais, acesse o site: (<https://www.digi.com/products/iot-platform/xctu>).

XBee

O módulo Xbee é um circuito integrado que permite comunicação via radiofrequência utilizando o padrão de rede sem fio Zigbee.



Existem várias séries de Xbee, entre as mais conhecidos estão as versões: S1, S2B, PRO, entre outros. Suas versões variam entre melhorias na administração dos dispositivos na rede, firmwares atualizadas, perfis diferentes em rede, mais rapidez na configuração, etc.

Todas as informações e configurações que o módulo Xbee deve portar podem ser feitas através do XCTU, apresentado na seção anterior. As principais configurações a serem feitas serão descritas nesta seção.

A configuração do módulo Xbee é relativamente simples para utilizá-los em redes sem muitas restrições, como exemplo: conexão entre um módulo conectado ao computador e um módulo conectado ao Arduino (em um robô), desta forma, teremos dois papéis na rede, um módulo será o **end-device** e o outro módulo o **coordenator**.

Através do XCTU, você deverá conectar os módulos e navegar até a janela de configuração, os principais campos a serem configuração são:

Na tela de configuração - **Networking**:

- **ID PAN ID**: Nome de identificação da Rede - Ex.: DF;
- **SC Scan Channels**: Alcance de canais da Rede - Ex.: FFFF;

Na tela de configuração - **Addressing**:

- **SL Serial Number Low**: Parte baixa do endereço contido na traseira do módulo Xbee (Primeira linha)
- **SH Serial Number High**: Parte alta do endereço contido na traseira do módulo Xbee (Segunda linha)



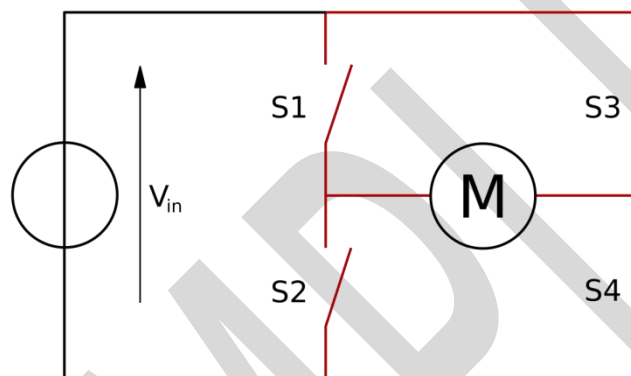
- **DH Destination Address High**: Parte alta do endereço de destino (Ex.: 0)
- **DL Destination Address Low**: Parte baixa do endereço de destino (Ex.: FFFF)
- **NI Node Identification**: Identificação do módulo Xbee (Ex.: modulo1)

Para salvar, clique no botão **Write**, parte superior do XCTU. As configurações dos demais módulos deverão ser realizadas da mesma forma, como a ideia é trocar informações entre os dispositivos da mesma rede, as informações do **ID**, **SC**, **DH** e **DL** devem ser as mesmas para todos.

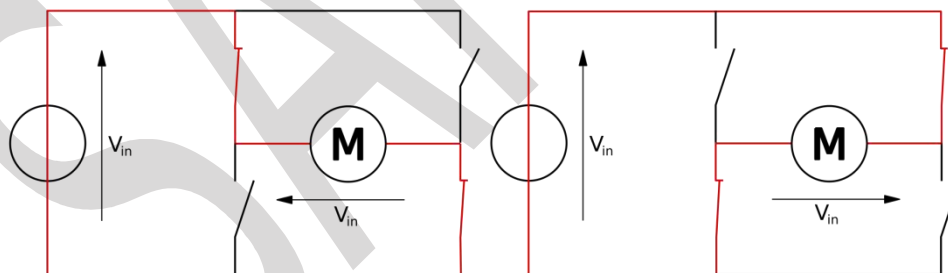
Ponte H

A ponte H é responsável por controlar motores. Antes de você saber o básico de ponte H e o motivo de utilizá-las, vale a pena falar sobre motores DC. Motores DC - Corrente Contínua - são compostos de bobinas que, ao receberem corrente, criam campos magnéticos (polos magnéticos) fazendo com que o eixo do motor gire.

Para controlar os movimentos desses motores (velocidade e sentido), um módulo muito utilizado é a Ponte H, que recebe esse nome pelo fato de o circuito parecer um H, veja abaixo.

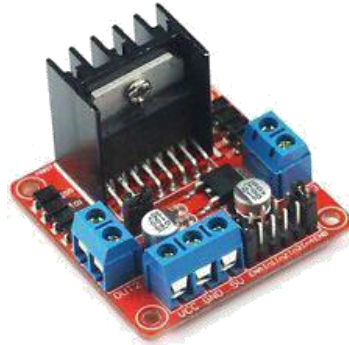


Uma Ponte H controla o sentido da corrente que flui sobre o circuito e a polaridade da tensão aplicada, isso é feito chaveando-se os componentes do circuito. Esse chaveamento é realizado comumente através de PWM, que é responsável por controlar a tensão média aplicada no Motor M, consequentemente, a velocidade de giro do motor.



Acionando as chaves S1 e S4 do circuito, o terminal direito fica com carga positiva, o que faz a corrente fluir da direita para a esquerda, fazendo com que o eixo do motor gire em um sentido, se o acionamento ocorrer nas chaves S2 e S3, o motor gira no sentido oposto.

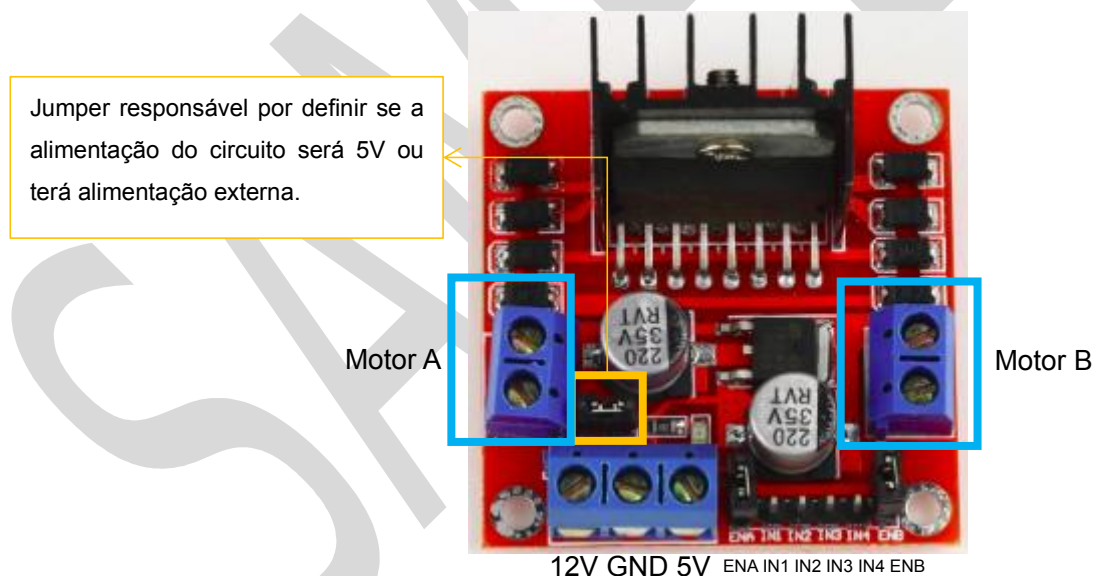
O circuito da Ponte H é muito encontrado em projetos que envolvem motores e Arduino, o módulo comumente utilizado é o L298.



Esse módulo possui as seguintes características:

- Tensão de operação: 4 a 35V
- Capacidade de controlar até 2 motores
- Corrente máxima de operação: 2A para cada motor
- Tensão lógica: 5V
- Corrente lógica: 0 a 36mA

Os terminais azuis, da imagem abaixo, correspondem às conexões dos motores DC A e B. Este módulo suporta um único motor de passo, caso não forem utilizados os motores DC. Na imagem abaixo é possível ver os pinos que são utilizados para controle dos motores.



O jumper destacado em amarelo é responsável por definir se o circuito receberá 5V de alimentação (conectado ao **pino 5V**) ou se receberá tensão maior (conectado ao **pino 12V**). Quando o jumper estiver conectado, o pino 5V é capaz de **fornecer uma tensão de saída de 5V**, que pode ser utilizada para alimentar outro componente. Quando o jumper não estiver conectado, é possível **alimentar o circuito da Ponte H fornecendo uma tensão de entrada de 5V no pino 5V**.

Os pinos **ENA** e **ENB** são os responsáveis por receberem o sinal PWM para controlar a velocidade dos motores. Quando os pinos **ENA** e **ENB** estão “jumpeados”, a tensão

lógica aplicada é de 5V de forma direta, não havendo controle de velocidade.

Os pinos **IN1** e **IN2** são responsáveis por controlar o sentido de rotação do motor A, e os pinos **IN3** e **IN4** são responsáveis por controlar o sentido de rotação do motor B. A tabela abaixo define os sentidos dos motores DC.

IN1	IN2	Estado
0V	0V	Desligado
0V	5V	Sentido 1
5V	0V	Sentido 2
5V	5V	Parado