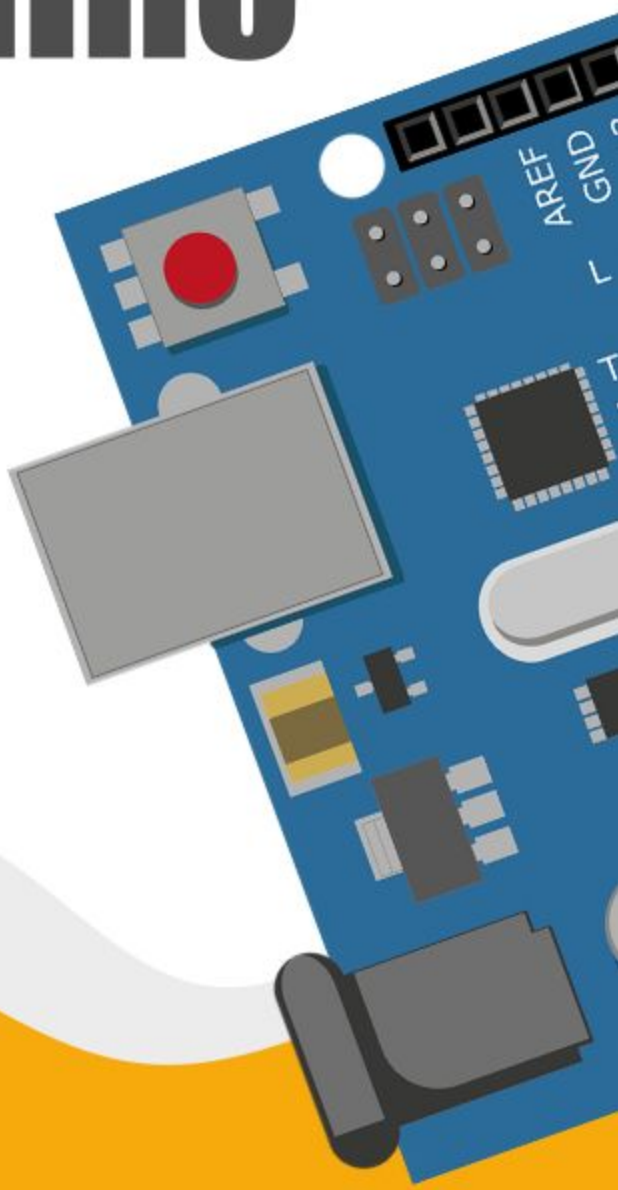
EMBARCADOS

Introdução ao Arduino



Agosto 2019

Ebook Introdução ao Arduino

Olá,

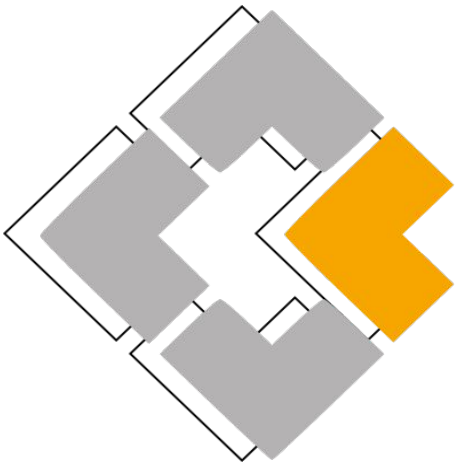
Obrigado por baixar o nosso ebook: **Introdução ao Arduino**. Esse ebook traz uma coleção de textos já publicados no Embarcados sobre a plataforma Arduino.

Fizemos um compilado de textos que consideramos importantes para os primeiros passos com a plataforma. Espero que você aproveite esse material e lhe ajude em sua jornada.

Continuamos com a missão de publicar textos novos diariamente no site.

Um grande abraço.

Equipe Embarcados.



EMBARCADOS

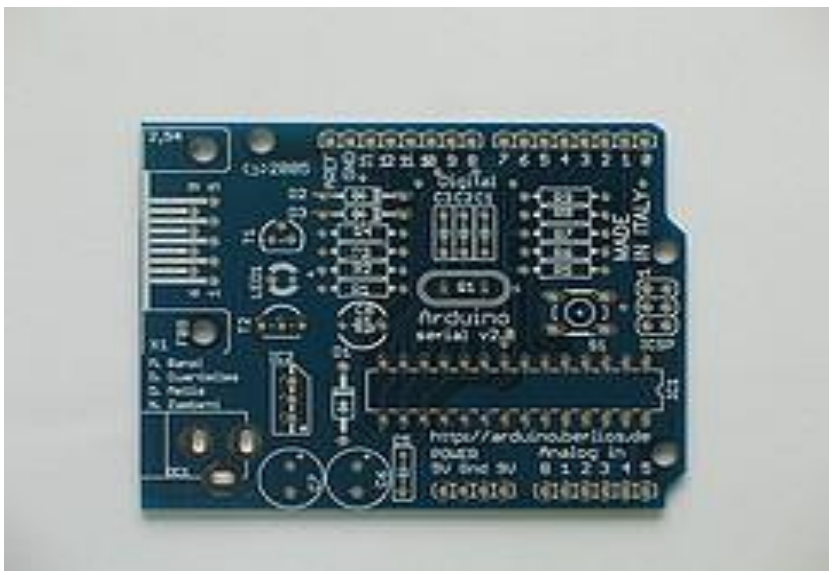
PLACAS ARDUINO HISTÓRIA ATÉ O ARDUINO UNO

por Fábio Souza

Você sabe como as placas Arduino evoluíram? Sabe qual é a última revisão das placas USB básicas? Este artigo exibirá a evolução das placas básicas da plataforma Arduino abordando as principais modificações de hardware até chegar na simpática placa conhecida como Arduino UNO, ou seja, toda história até o Arduino Uno.

As primeiras placas para plataforma Arduino iniciaram com comunicação serial e componentes discretos e eram vendidas desmontadas em kits ou apenas a PCB. Abaixo são exibidas suas versões:

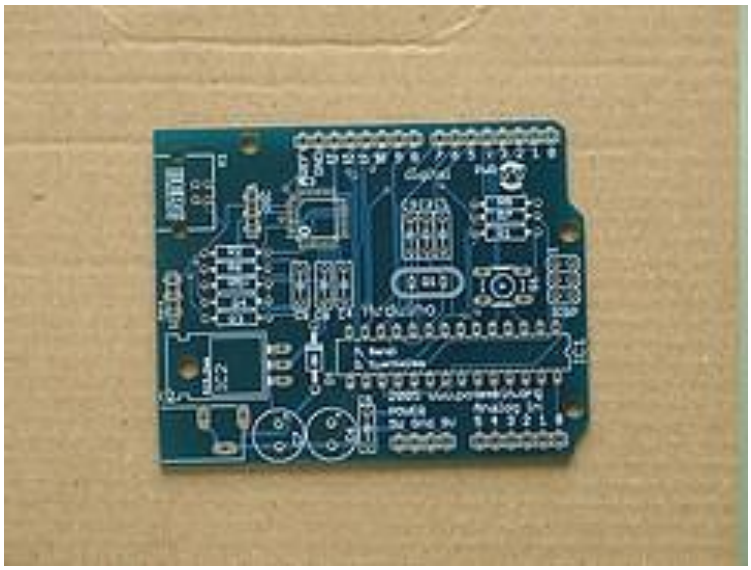
Placas Arduino: Arduino Serial



Placas Arduino: Arduino Serial v2.0

Essas placas usavam o padrão RS232 para interface com um computador e necessitavam de alimentação externa através de plug Jack para seu funcionamento.

Devido ao fato do grande uso da USB e da possibilidade de alimentar a placa diretamente através do cabo de comunicação, foi criada a placa Arduino USB, a primeira placa a sair com o nome Arduino. Porém, como todo projeto.... Houve um erro nessa versão, a pinagem do conector USB estava errada!!!



Arduino USB

Arduino USB v2.0

Após a correção da pinagem do conector USB, foi lançada a placa **Arduino USB v2.0**. Essa versão ainda mantinha a maioria dos componentes como discretos e trazia como conversor USB serial o chip FT232BM, da FTDI.

Outro ponto interessante a ser notado nessa placa é que ela possuía um jumper para seleção da alimentação ou pela USB, ou pelo adaptador externo. Dessa forma, poderia testar seus sketch diretamente em um notebook sem a necessidade de uma alimentação externa.

Após o lançamento da placa Arduino USB v2.0 foi lançada a placa Arduino Extreme,que trouxe a maioria dos componentes em montagem SMD e lançou os conectores headers fêmea, queaficou conhecido como "padrão arduino". Além disso foram colocados 2 leds, RX e TX, que indicavam o tráfego de dados entre a placa e o computador.

Arduino Extreme

Posteriormente uma nova versão da Arduino Extreme foi lançada, Arduino Extreme V2,que trouxe como melhoria um melhor layout com plano de terra mais elaborado. Além disso trouxe impressa a URL: www.arduino.cc.

Arduino Extreme V2

Após toda essa evolução da plataforma, o pessoal da equipe ainda não satisfeito, lançaram a Arduino NG (Nuova Generazione). A Arduino NG trouxe como novidade o conversor USB-SERIAL: FT232RL. Esse conversor necessita de menor quantidade de componentes externos em comparação ao FT232BM. Outra novidade foi o LED colocado no pino 13, porém este LED causava interferência na comunicação SPI.

Arduino NG

Até esse ponto as placas utilizavam o ATmega8 como microcontrolador, porém durante o projeto da NG os ATmega168 tomou o seu lugar dobrando a capacidade de memória para 16KB.

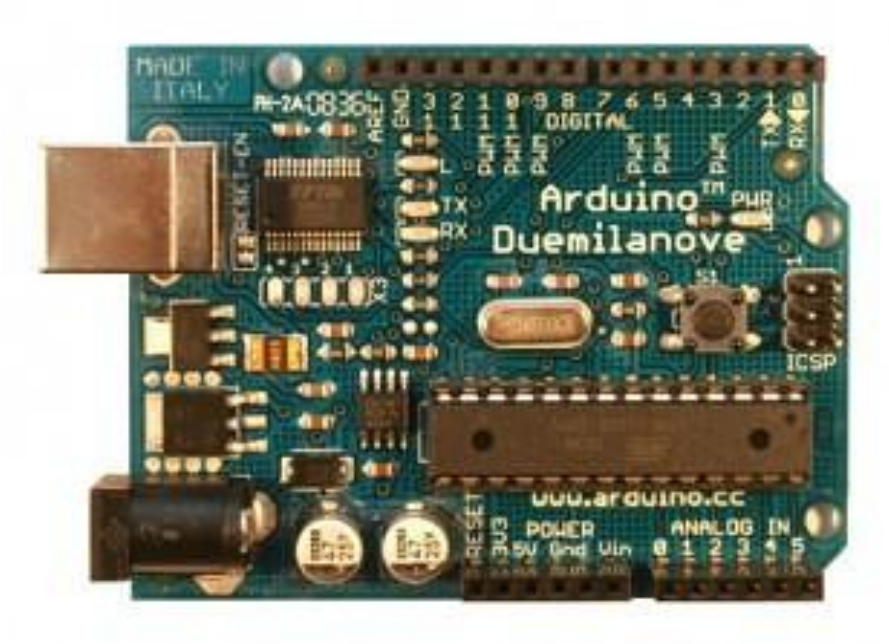
Para resolver o problema da comunicação SPI foi lançada a Arduino NG REV. C. Essa placa não trazia montado o LED conectado no pino 13, trazia apenas os pads para soldagem. Possuía um resistor de 1k em série com o conector do pino 13 onde era possível ligar um LED externo sem a necessidade de resistor.

Arduino NG REV. C

Todas as placas apresentadas até agora necessitavam de reset externo para ativar o bootloader, o que dificultava na hora de fazer upload do sketch para a placa. Para resolver esse problema foi lançada a placa Arduino Diecimila, que possuía circuito de reset através da comunicação serial. Dessa forma, sempre que era feito o upload, a placa entrava em modo bootloader automaticamente. Outro ponto interessante foi os fusíveis para proteção da USB em caso de curto-circuito, protegendo a porta USB do computador. A Diecimila trazia ainda novidades em seus conectores headers, foi inserido um conector de Reset e um de 3,3 V. O led conectado ao pino 13 foi inserido novamente à plataforma.

Placas Arduino: Arduino Diecimila

Apesar de todos os recursos apresentados pela **Arduino Diecimila,** a seleção da fonte de alimentação da placa era manual. Para resolver este problema foi lançada a **Arduino Duemilanove** que selecionava automaticamente a fonte sem a necessidade de um jumper para isso. Em março de 2009, a **Arduino Duemilanove** começou a ser fabricada com o ATmega328 duplicando novamente a memória que no caso iria para 32 KB.



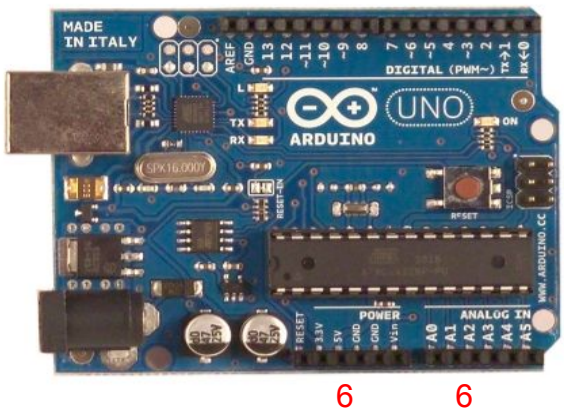
Placas Arduino: Arduino Duemilanove

Após o desenvolvimento do Duemilanove, foi lançada a placa homônima Arduino UNO, que trouxe algumas novidades no hardware em relação a sua antecessora. A principal modificação foi a substituição do conversor USB-serial por um microcontrolador ATmega8U2, que posteriormente na revisão 3 foi substituído pelo ATmega16U2. Além disso a descrição das entradas e saída foi melhorada para uma melhor identificação dos pinos. A placa Arduino UNO passou por 3 revisões até hoje, conforme exibido abaixo:

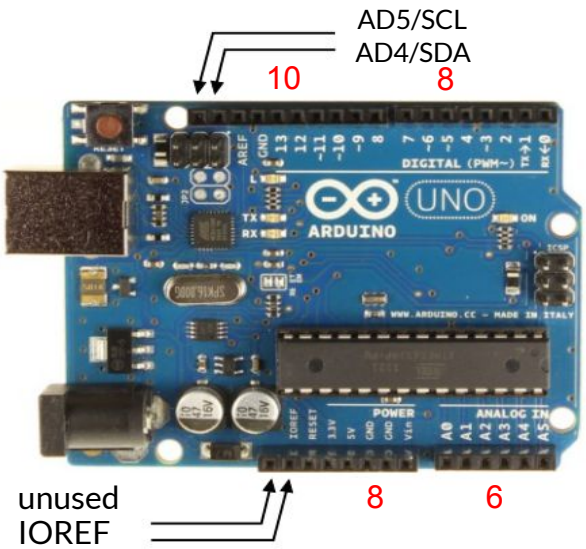
Arduino UNO Rev. 1	Arduino UNO Rev. 2	Arduino UNO Rev. 3

Consegue notar alguma diferença?

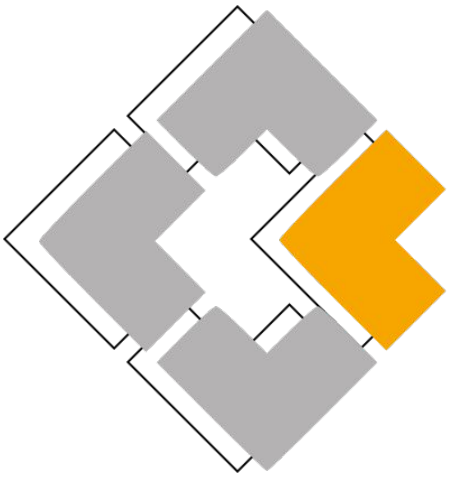
As principais modificações ocorreram da revisão 2 para 3 onde foram acrescentados mais dois pinos após o conector AREF. Estes pinos podem ser usados como entradas analógicas ou para comunicação I2C. No conector de POWER, que na revisão 2 possuía 6 pinos, também foram acrescentados mais 2 pinos após a entrada RESET, um deles é o pino IOREF que permite que os shields se adaptem conforme a tensão da placa. No futuro, os shields serão compatíveis com placas que com ATMEL ATMEGA que são alimentadas com 5 V e com a Arduino Due que opera em 3,3V. O segundo pino não é conectado, e é reservado para uma aplicação futura.



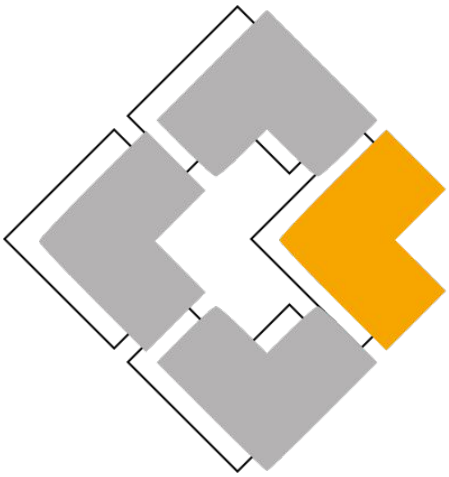
Uno & R2 Pins (R2 board shown)



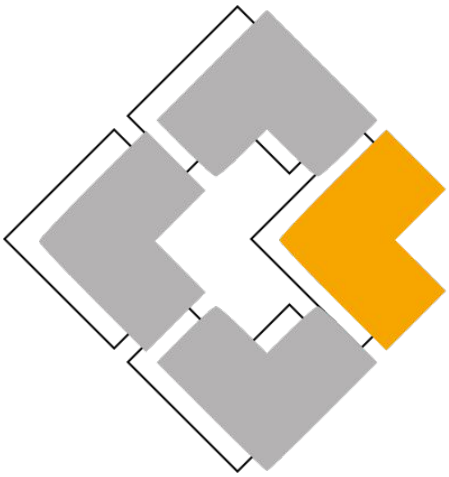
R3



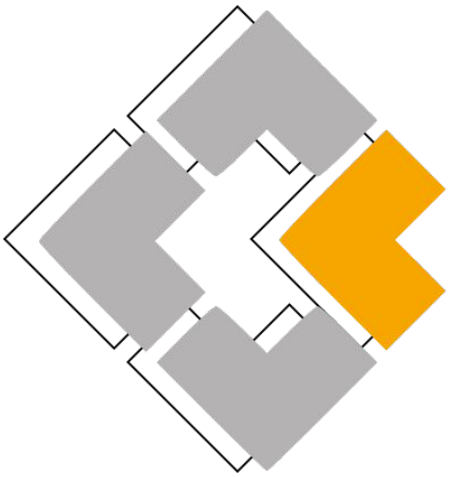
O que você achou da evolução da plataforma Arduino?



O que você faria diferente no desenvolvimento dessas placas?



Será que terá uma nova revisão da Arduino UNO em um futuro próximo?



O que poderia ser mudado?

Este artigo foi escrito por Fábio Souza.

Confira os webinars gratuitos

EMBARCADOS

DOCUMENTÁRIO SOBRE ARDUINO

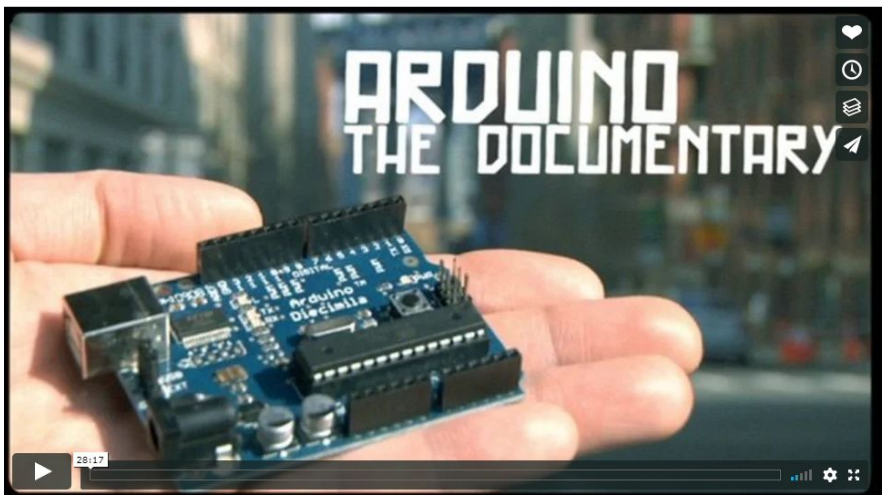
por Fábio Souza

Você que é fã de Arduino ou que usa essa plataforma em seus projetos, já deve ter feito essas perguntas: como essa plataforma surgiu? Como tiveram essa grande ideia? E qual foi o segredo para essa plataforma ganhar tanta popularidade no mundo?

O vídeo, com legendas em português, exibe em seus quase 30 minutos o desenvolvimento dessa plataforma em 2005 na Itália, por um grupo ligado ao Instituto de design de interação Ivrea. Exibe entrevistas com os criadores do projeto e imagens de instituições e pessoas que fizeram projetos com a plataforma Arduino.

A idéia principal do projeto foi o de criar uma placa de baixo custo e pronta pra uso para reduzir o valor do investimento na elaboração de projetos de hardware dos alunos. Além disso a plataforma deveria ter uma linguagem de programação de fácil compreensão. Basearam-se em duas linguagens: a Processing e a Wring, o uso dessa linguagens ajudou a tornar o projeto em código aberto permitindo a contribuição da comunidade para a evolução do projeto.

O que vocês acharam do vídeo? Possuem alguma ideia que pode virar sucesso? Pretende deixar em código aberto?



Este artigo foi escrito por Fábio Souza.

Participe da comunidade no Facebook

EMBARCADOS

PRIMEIROS PASSOS NA PLATAFORMA

por Fábio Souza

Arduino é uma plataforma de código aberto (hardware e software) criada em 2005 pelo italiano Massimo Banzi (e outros colaboradores) para auxiliar no ensino de eletrônica para estudantes de design e artistas. O objetivo principal foi o de criar uma plataforma de baixo custo, para que os estudantes pudessem desenvolver seus protótipos com o menor custo possível. Outro ponto interessante do projeto, foi a proposta de criar uma plataforma de código aberto, disponível para a comunidade o que ajudou em muito no seu desenvolvimento. Veja a sua história aqui.

Nesse artigo você irá conhecer um pouco mais sobre a plataforma e vai dar os primeiros passos para deixar o seu ambiente preparado para o desenvolvimento dos seus projetos.

Introdução ao Arduino

O site da plataforma o define como:

“O Arduino é uma plataforma de prototipagem eletrônica open-source que se baseia em hardware e software flexíveis e fáceis de usar. É destinado a artistas, designers, hobbistas e qualquer pessoa interessada em criar objetos ou ambientes interativos.

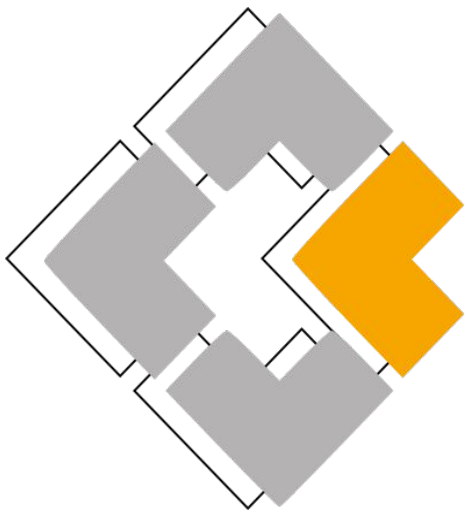
Plataforma de desenvolvimento Arduino

O Arduino pode *sentir* o estado do ambiente que o cerca por meio da recepção de sinais de sensores e pode interagir com os seus arredores, controlando luzes, motores e outros atuadores. O microcontrolador na placa é programado com a linguagem de programação Arduino, baseada na linguagem Wiring, e o ambiente de desenvolvimento Arduino, baseado no ambiente Processing. Os projetos desenvolvidos com o Arduino podem ser autônomos ou podem comunicar-se com um computador para a realização da tarefa, com uso de software específico (ex: Flash, Processing, MaxMSP)."

A plataforma é formada por dois componentes principais: Hardware e Software.

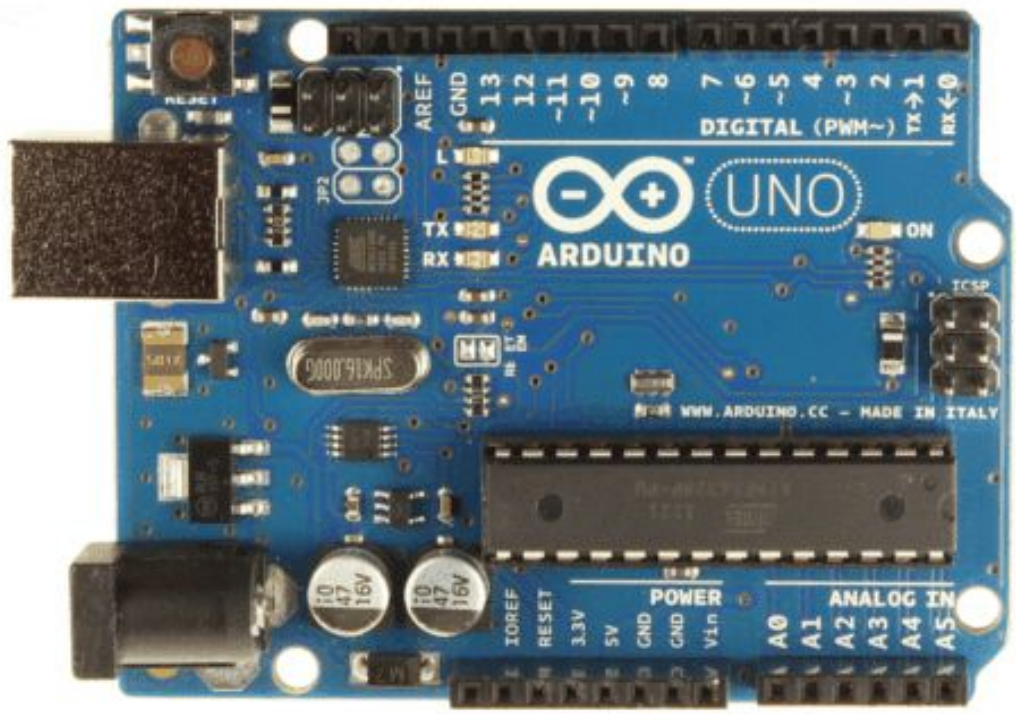
O hardware é composto por uma placa de prototipagem na qual são construídos os projetos.

O software é uma IDE, que é executado em um computador onde é feita a programação, conhecida como sketch, na qual será feita upload para a placa de prototipagem Arduino, através de uma comunicação serial. O sketch feito pelo projetista dirá à placa o que deve ser executado durante o seu funcionamento.



Hardware

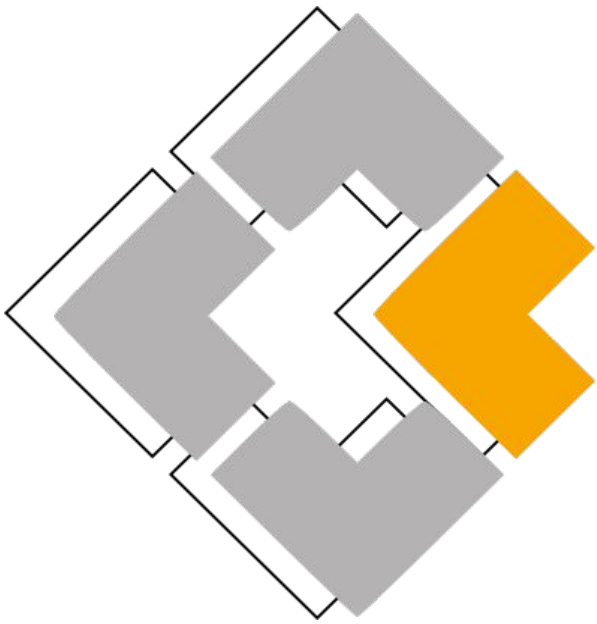
Existem diversas placas oficiais de Arduino e muitas outras não oficiais. Vamos abordar a placa Arduino Uno neste artigo. A seguir é exibida a placa Arduino Uno REV3:



Conforme visto na imagem acima a placa Arduino UNO possui diversos conectores que servem para interface com o mundo externo. Vejamos como estão organizados os pinos na placa:

- 14 pinos de entrada e saída digital (pinos 0-13):
    - Esses pinos podem ser utilizados como entradas ou saídas digitais de acordo com a necessidade do projeto e conforme foi definido no sketch criado na IDE.
- 6 pinos de entradas analógicas (pinos A0 - A5):
    - Esses pinos são dedicados a receber valores analógicos, por exemplo, a tensão de um sensor. O valor a ser lido deve estar na faixa de 0 a 5 V onde serão convertidos para valores entre 0 e 1023.
- 6 pinos de saídas analógicas (pinos 3, 5, 6, 9, 10 e 11):
    - São pinos digitais que podem ser programados para ser utilizados como saídas analógicas, utilizando modulação PWM.

A alimentação da placa pode ser feita a partir da porta USB do computador ou através de um adaptador AC. Para o adaptador AC recomenda-se uma tensão de 9 a 12 volts.



Software

O software para programação do Arduino é uma IDE que permite a criação de sketches para as placas. A linguagem de programação é modelada a partir da linguagem Wiring . Quando pressionado o botão upload da IDE, o código escrito é traduzido para a linguagem C e é transmitido para o compilador avr-gcc, que realiza a tradução dos comandos para uma linguagem que pode ser compreendida pelo microcontrolador.

A IDE apresenta um alto grau de abstração, possibilitando o uso de um microcontrolador sem que o usuário conheça o mesmo, nem como deve ser usado os registradores internos de trabalho.

A IDE possui uma linguagem própria baseada na linguagem C e C++.

O Ciclo de programação do Arduino pode ser dividido da seguinte maneira:

1. Conexão da placa a uma porta USB do computador;
2. Desenvolvimento de um sketch com comandos para a placa;
3. Upload do sketch para a placa, utilizando a comunicação USB.
4. Aguardar a reinicialização, após ocorrerá à execução do sketch criado.

A partir do momento que foi feito o upload o Arduino não precisa mais do computador: o Arduino executará o sketch criado, desde que seja ligado a uma fonte de energia.

IDE do Arduino

A IDE pode ser baixada gratuitamente no site do Arduino, onde pode ser escolhida a melhor opção de download conforme plataforma utilizada.

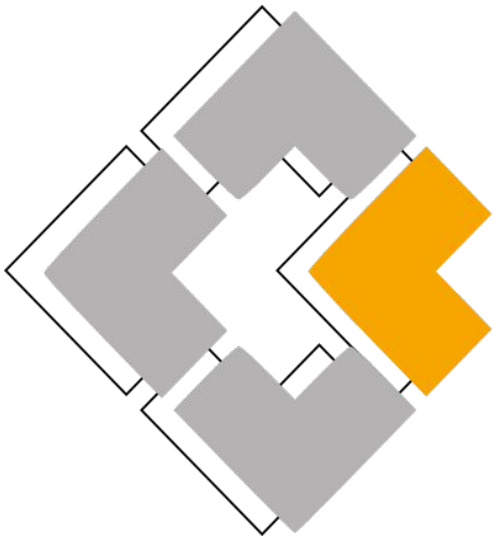
Quando se abre o IDE do Arduino, será exibido algo semelhante à figura abaixo:

O IDE é dividido em três partes: A Toolbar no topo, o código ou a Sketch Window no centro, e a janela de mensagens na base, conforme é exibido na figura anterior.

Na Toolbar há uma guia, ou um conjunto de guias, com o nome do sketch. Ao lado direito há um botão que habilita o serial monitor. No topo há uma barra de menus, com os itens File, Edit, Sketch, Tools e Help. Os botões na Toolbar fornecem acesso rápido às funções mais utilizadas dentro desses menus.

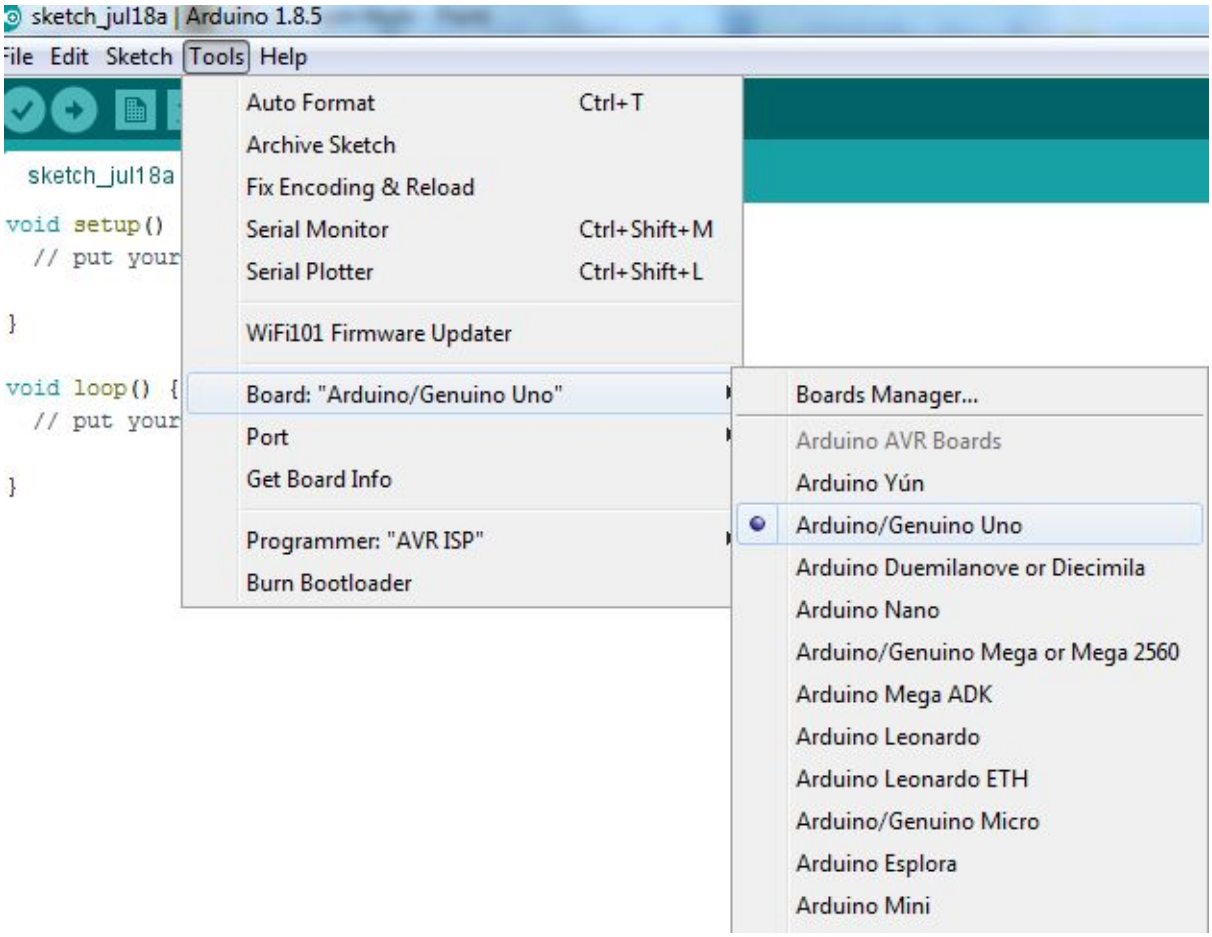
Abaixo são identificados os ícones de atalho da IDE:

- **Verify**
    - Verifica se existe erro no código digitado.
- **Upload**
    - Compila o código e grava na placa Arduino se corretamente conectada;
- **New**
    - Cria um novo sketch em branco.
- **Open**
    - Abre um sketch, presente no sketchbook.
- **Save**
    - Salva o sketch ativo
- **Serial monitor**
    - Abre o monitor serial.

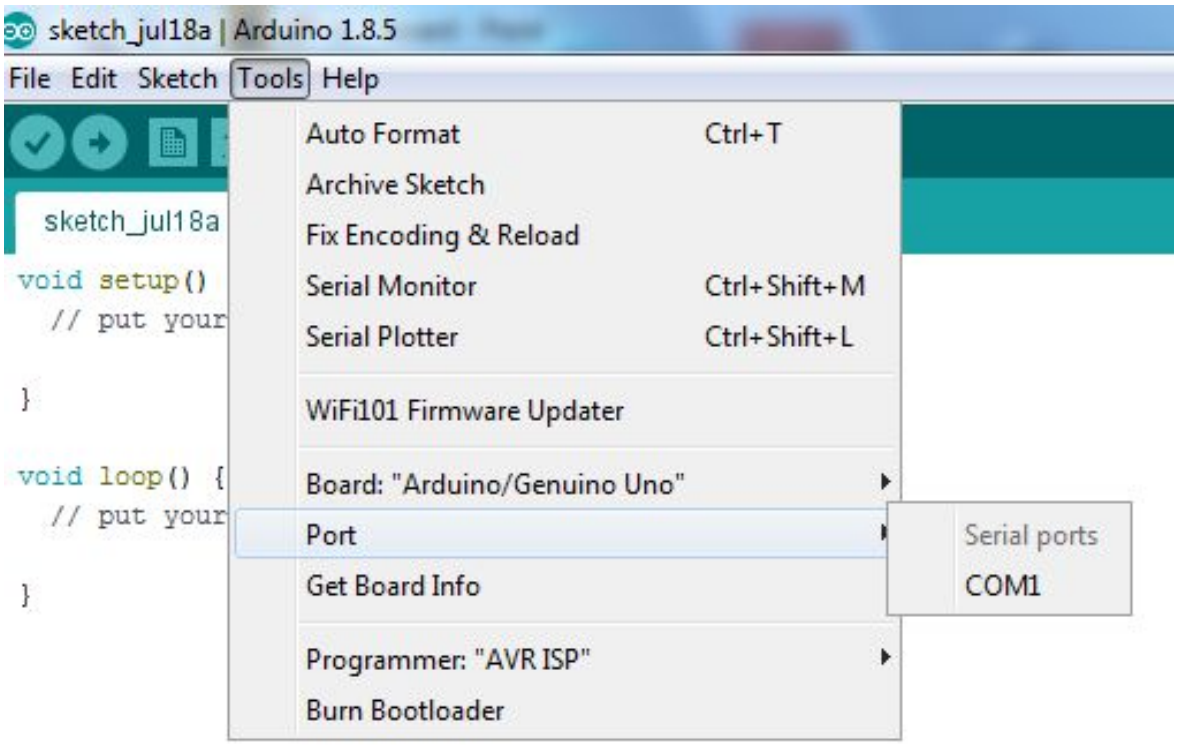


Os demais comandos presentes na barra de menus podem ser consultados através do menu <help><Environment>.

Após a conexão do Arduino ao computador, é atribuído a placa uma COM. A primeira vez que o programa Arduino for executado deve-se selecionar o modelo de placa utilizado, no nosso caso escolhemos Arduino Uno, conforme figura abaixo:



Após a definição do modelo, deve-se selecionar em qual COM a placa foi atribuída:

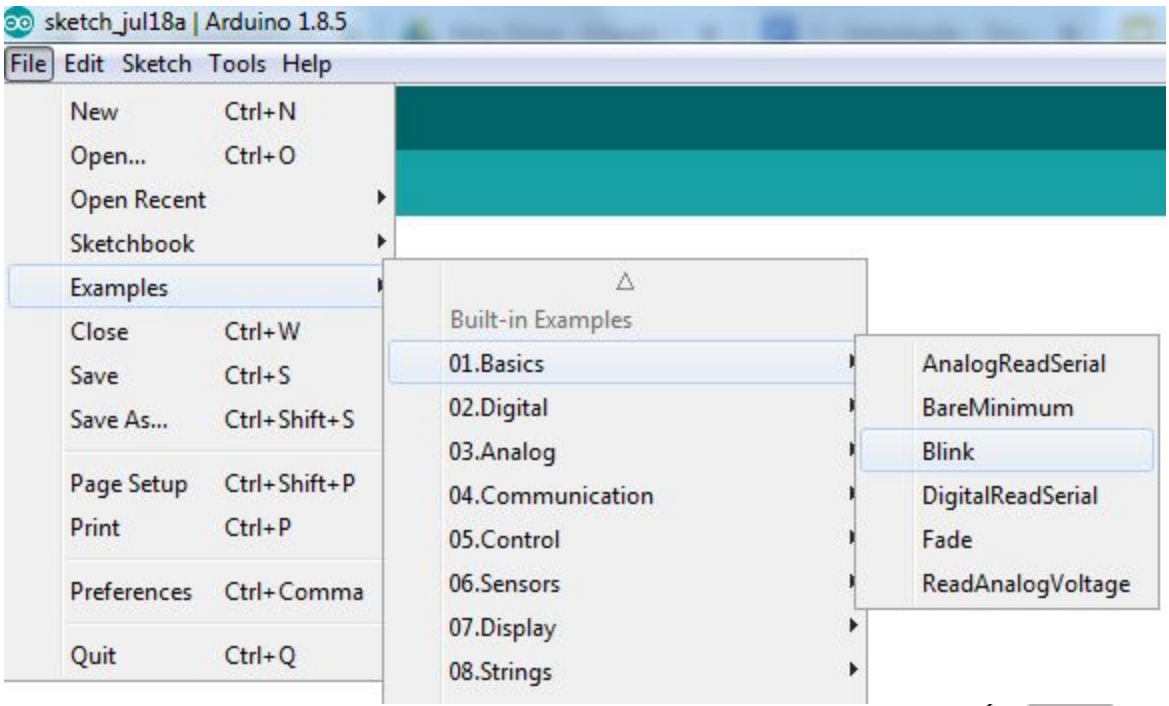


Após estas configurações o ambiente está preparado para uso e pode-se testar qualquer um dos exemplos que acompanham a IDE ou até mesmo com um novo sketch.

"Hello World" – Blink

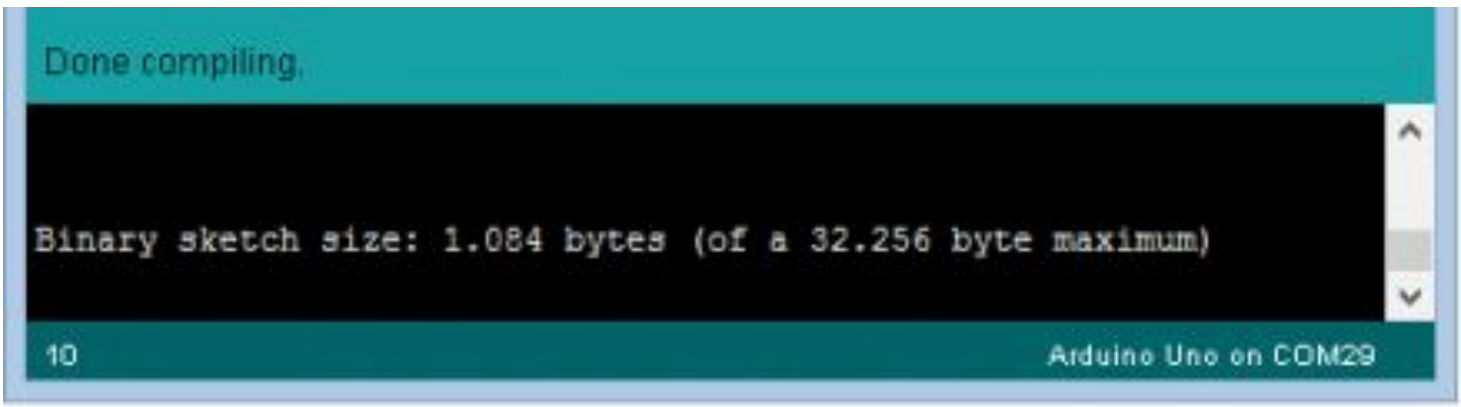
O exemplo mais simples para iniciar a programação do Arduino, que pode ser considerado como o conhecido “Hello World” das linguagens de programação, consiste em acionar um LED através de uma saída digital.

A placa Arduino Uno já possui um Led ligado ao pino digital 13 que pode ser utilizado para o teste, e na IDE podemos carregar o exemplo *Blink*:



```
/*
Blink
Turns on an LED on for one second, then off for one second, repeatedly.

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
```

Para verificar de o código está correto deve-se clicar no ícone verify, após a compilação é exibida uma mensagem de status da operação e caso esteja tudo certo será exibida a quantidade de bytes gerados pelo programa:



Para gravar o código na memória flash do microcontrolador é necessário clicar no ícone Upload, será transferido o código para a placa e após alguns segundos o LED ligado ao pino 13 começará a piscar em intervalos de 1 segundo.

Analisando o Código

O código do exemplo Blink é relativamente simples, porém apresenta a estrutura básica de um programa desenvolvido na IDE Arduino. Inicialmente nota-se que existem duas funções obrigatórias em um programa Arduino, setup() e loop().

A função setup () é executada na inicialização do programa e é responsável pelas configurações iniciais do microcontrolador, tal como definição dos pinos de I/O, inicialização da comunicação serial, entre outras.

A função loop () será onde ocorrerá o laço infinito da programação, ou seja, onde será inserido o código que será executado continuamente pelo microcontrolador.

Dentro do loop principal está o código que fará o led ligado pino 13 piscar em intervalos de 1 segundo.

A função digitalWrite(led, HIGH); coloca o pino em nível lógico 1, ligando o led.

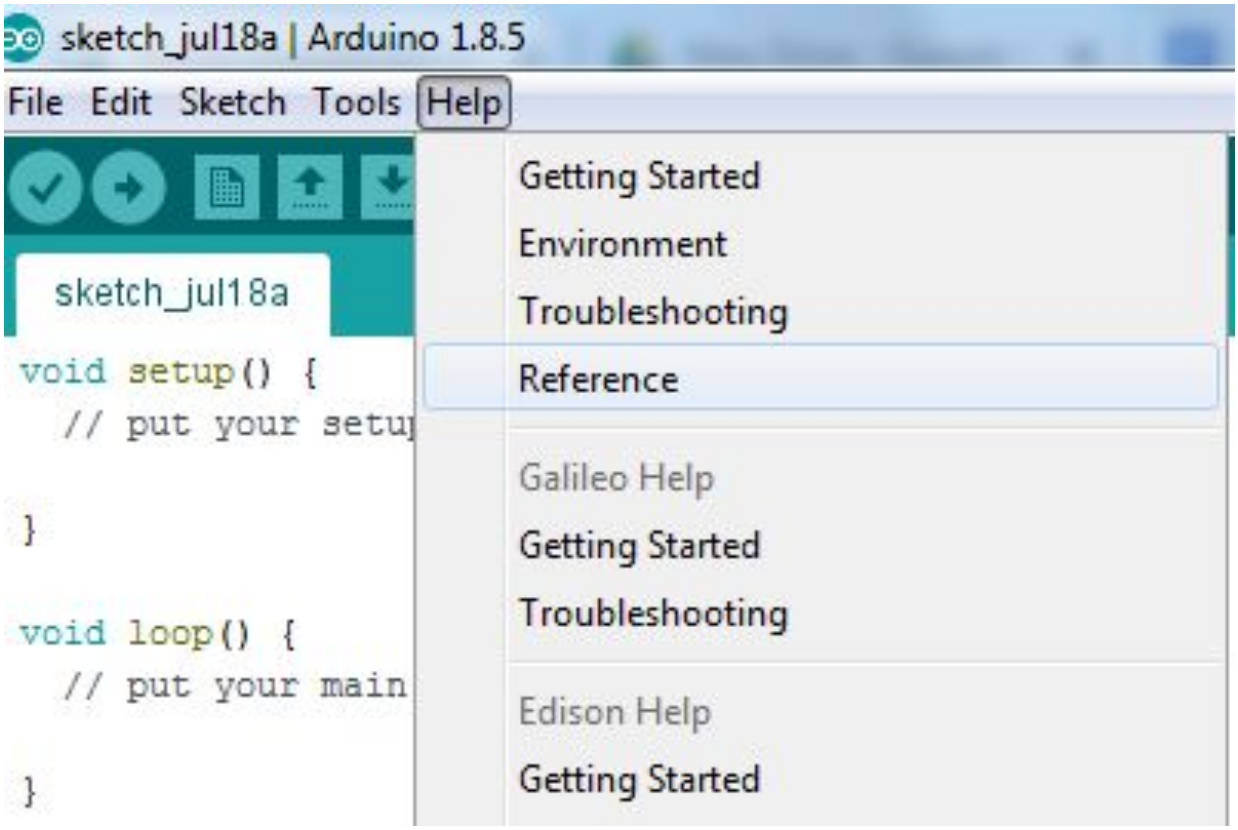
A função delay(1000); aguarda o tempo de 1000 ms, ou seja, 1 segundo para que possa ser executada a próxima instrução.

A função digitalWrite(led, LOW); coloca o pino em nível lógico 0, desligando o led.

E novamente é esperado 1 segundo com a função delay();

O loop é repetido infinitamente enquanto a placa estiver ligada.

A referência da linguagem pode ser acessada através do menu <help>:



Testar outros exemplos básicos, é um bom começo para aprender mais sobre a plataforma.

Este artigo foi escrito por Fábio Souza.

EMBARCADOS

USANDO PINOS DIGITAIS NO ARDUINO

por Fábio Souza

Quando você adquire uma placa Arduino Uno, a primeira coisa que vem em mente é como ligar e desligar um equipamento ou eletrodoméstico, ou como ler uma tecla do computador para enviar comandos para a placa, ou seja, usar os pinos digitais do Arduino para controlar o mundo a sua volta.

Este artigo visa apresentar as funções de manipulação dos pinos digitais do Arduino para usá-los como entrada ou saída digital, exibindo os passos para correta configuração e por fim o seu uso em algumas aplicações.

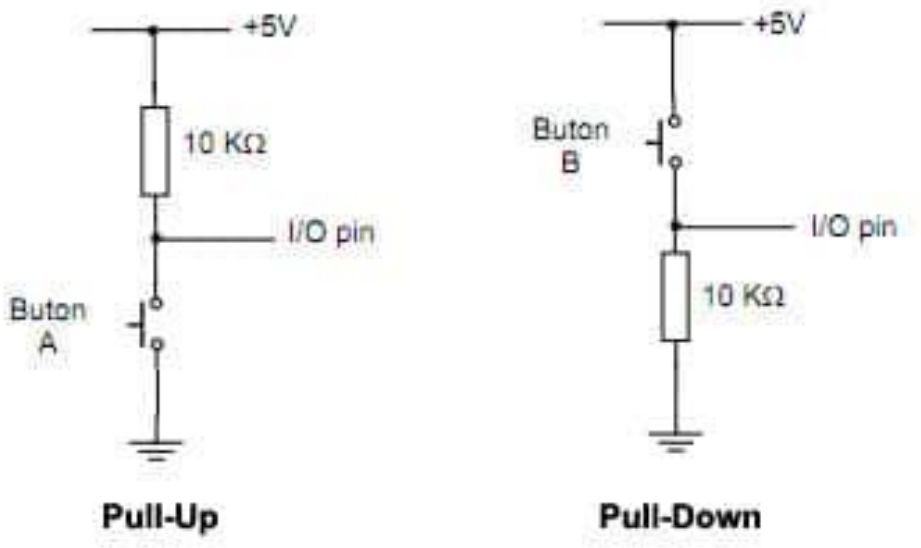
Pinos digitais da Arduino UNO

A placa Arduino UNO possui 14 pinos que podem ser configurados como entrada ou saídas digitais conforme a necessidade de seu projeto. Estes pinos são numerados de 0 a 13, conforme destacado na figura a seguir:

Pinos digitais da Arduino UNO - Entradas e Saídas Digitais da placa

Antes de utilizar cada um desses pinos em sua aplicação, você deve configurá-lo como entrada ou saída digital, conforme a necessidade. Por exemplo, para acionar um LED você deve configurar o pino como saída e para ler uma tecla você deve configurar o pino como entrada.

Por padrão os pinos digitais do Arduino estão configurados como entradas digitais, porém, para ficar mais explícito na programação, deve-se configurar o pino como entrada. Dessa forma o pino é colocado em um estado de alta impedância, equivalente a um resistor de 100 MegaOhms em serie com o circuito a ser monitorado. Dessa forma, o pino absorve uma corrente muito baixa do circuito que está monitorando. Devido a essa característica de alta impedância, quando um pino colocado com entrada digital encontrasse flutuando (sem ligação definida), o nível de tensão presente nesse pino fica variando não podendo ser determinado um valor estável devido a ruído elétrico e até mesmo capacitância de entrada do pino. Para resolver esse problema é necessário colocar um resistor de pull up (ligado a +5V) ou um resistor de pull down (ligado a GND) conforme a necessidade. Esses resistores garantem nível lógico estável quando por exemplo uma tecla não está pressionada. Geralmente utiliza-se um resistor de 10K para esse propósito. A seguir é exibida a ligação desses resistores no circuito para leitura de tecla:



Pull-Up **Pull-Down**

O microcontrolador ATmega328, da placa Arduino UNO, possui resistores de pull-up internos (20 kΩ) que facilitam a ligação de teclas, sensores sem a necessidade de conectar externamente um resistor de pull-up. A habilitação desses resistores é feita de maneira simples via software.

Quando um pino é configurado com saída, ele se encontra em estado de baixa impedância. Dessa forma, o pino pode fornecer ou drenar corrente para um circuito externo. A corrente máxima que um pino pode fornecer ou drenar é de 40 mA, porém a soma das correntes não pode ultrapassar 200 mA. Deve-se ficar atento a corrente maiores que este limite e a curto-circuitos que podem danificar o transistor de saída danificando o pino e até mesmo queimar o microcontrolador. Essa é uma característica perigosa para a placa Arduino e seria interessante se tivessem resistores ou algum tipo de proteção em todos os pinos utilizados como saída para limitar a corrente em uma situação anormal.

Funções para usos dos Pinos digitais do Arduino

A plataforma Arduino possui funções para trabalhar com entradas e saídas digitais que abstraem toda a configurações dos registradores que configuram e acessam os pino de I/O. Isso torna a programação do Arduino realmente fácil e esse é seu encanto. Essas funções são:

- void pinMode();

Essa função é utilizada para configurar um pino como entrada ou saída digital. Ela geralmente é utilizada dentro da função setup(). Apresenta as seguintes características:

Sintaxe:

pinMode(pino, modo);

Parâmetros:

pino: Número correspondente ao pino que se deseja configurar, conforme a placa que está trabalhando. No caso da Arduino UNO pode ser de 0 a 13;

modo: Modo que deseja configurar o pino. INPUT, INPUT_PULLUP, OUTPUT.

- INPUT: Entrada digital;
- INPUT_PULLUP: Entrada digital com resistor de pull-up (ligado ao VCC) interno habilitado;
- OUTPUT: Saída digital;

Retorno:

Essa função não tem retorno algum.

- int digitalRead();

Essa função lê o valor presente em um pino digital. Este valor pode ser HIGH ou LOW. Apresenta as seguintes características:

Sintaxe:

digitalRead(pino);

Parâmetros:

pino: valor correspondente ao pino que se deseja ler.

Retorno:

HIGH ou LOW.

- void digitalWrite();

A função digitalWrite() coloca um nível lógico Alto (HIGH, 5V) ou baixo (LOW, 0V) em um pino configurado como saída digital.

Sintaxe:

digitalWrite(pino, valor)

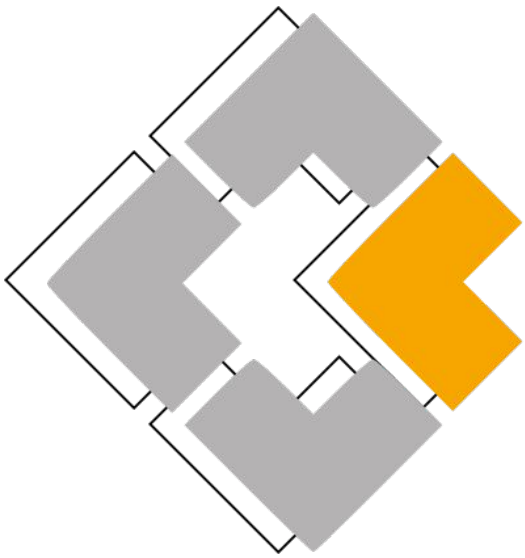
Parâmetros:

pino: Número correspondente ao pino;

valor: HIGH OU LOW

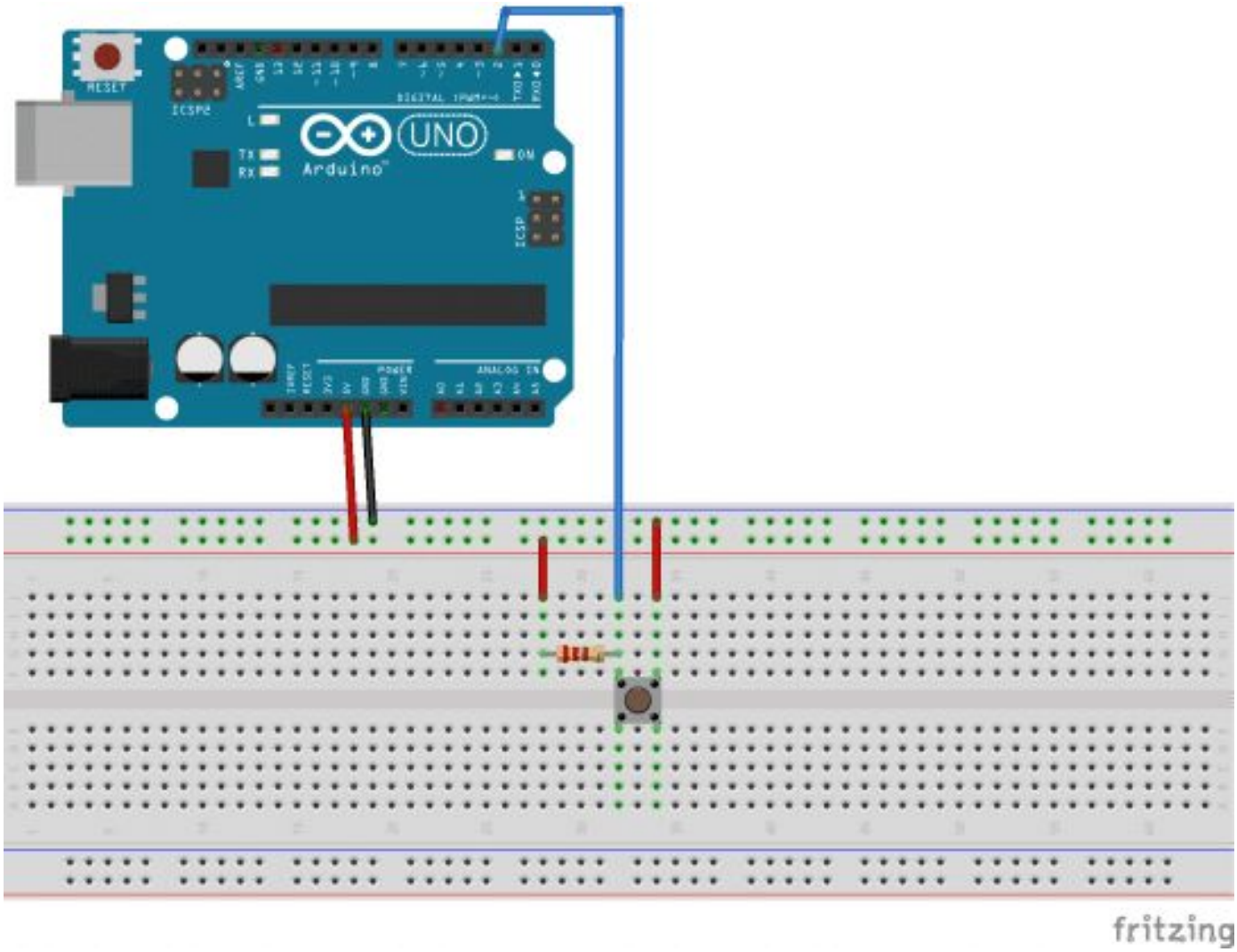
Retorno:

Essa função não tem retorno algum.

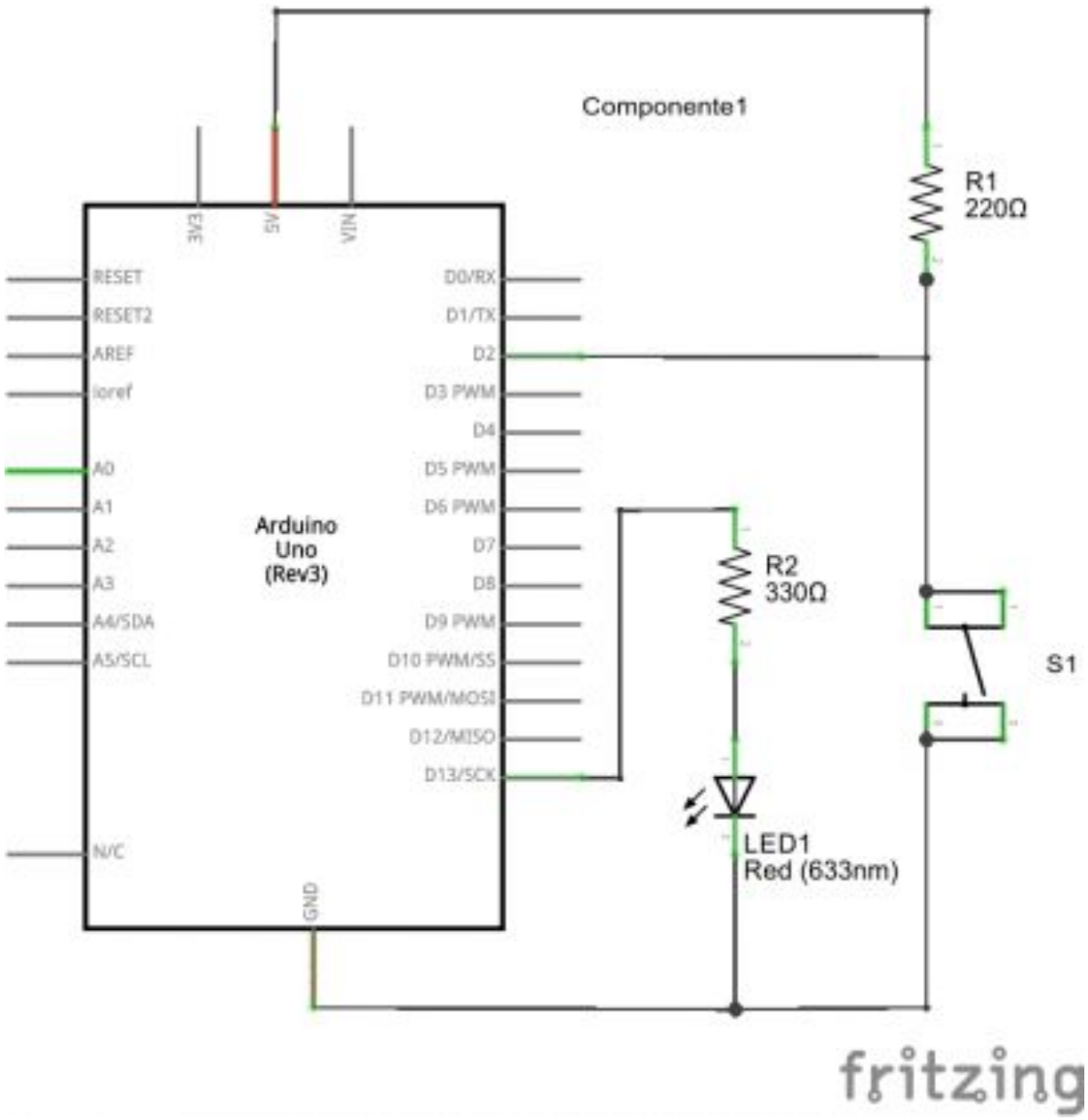


Exemplo

Para exemplificar a utilização de pinos de I/O digitais, vamos desenvolver uma aplicação de leitura de tecla e acionamento de LED, conforme o a montagem feita no software Fritzing exibida a seguir:



O esquema elétrico obtido a partir do Fritzing é exibido abaixo, note que no circuito há um resistor de pull-up que garante nível lógico alto quando tecla não está pressionada:



O exemplo consiste em ler a tecla S1 e ligar o LED caso a mesma estiver pressionada. Caso não esteja sendo pressionada, o LED deve permanecer desligado. O Sketch a seguir exibe a programação:

```
/*
Leitura de tecla
O exemplo le uma tecla conectada ao pino 2 e aciona um led conectado ao pino 13
*/

const int ledPin = 13; // cria uma constante com o numero do pino ligado ao LED
const int inputPin = 2; // cria uma constante com o numero do pino conectado a tecla

void setup()
{
  pinMode(ledPin, OUTPUT); // declara o pino do led como saída
  pinMode(inputPin, INPUT); // declara o pino da tecla como entrada
}

void loop()
{
  int val = digitalRead(inputPin); // le o valor na entrada
  if (val == LOW) // se valor está em zero( tecla pressionada)
  {
    digitalWrite(ledPin, HIGH); // Liga LED indicando tecla pressionada
  }
  else
  {
    digitalWrite(ledPin, LOW); // Desliga LED indicando tecla solta
  }
}
```

A programação apresentada acima possui uma estrutura bem simples e serve de início para a manipulação de pinos de I/O digitais. Com as três funções apresentadas é possível aplicar em diversos projetos que necessitem de acionamento e leitura de sinais digitais. Agora para fixar os conceitos apresentados é necessário colocar a mão na massa!!! Fica como exercício o desenvolvimento de um sketch para leitura de tecla com resistor de pull up interno habilitado para o pino onde é conecta a tecla S1.

Este artigo foi escrito por Fábio Souza.

EMBARCADOS

ENTENDENDO AS ENTRADAS ANALÓGICAS NO ARDUINO

por Fábio Souza

No artigo anterior descrevemos os pinos de entrada e saída digitais abordando as suas características e as funções para trabalhar com escrita e leitura de sinais digitais na plataforma Arduino. Nesse artigo vamos aprender sobre as entradas analógicas e compreender seu funcionamento para no futuro aplicar essa técnica para seus projetos.

Entradas Analógicas

As entradas digitais só podem assumir dois estados, HIGH e LOW, ou seja, 0 V ou 5 V. Dessa forma só é possível ler apenas dois estados. Por exemplo, verificar se uma porta está aberta ou fechada, identificar se um botão está pressionado ou solto, etc. Com as entradas digitais você aplica em lógica discreta para controle de seus projetos, porém em muitas situações a variação das grandezas envolvidas acontece de forma analógica. Ou seja, variam continuamente em relação ao tempo e podem assumir infinitos valores dentro de uma faixa. Como exemplo a temperatura, pressão e umidade são grandezas que variam dessa forma.

O microcontrolador da Arduino trabalha internamente com dados digitais, portanto é necessário traduzir um sinal analógico para um valor digital. A técnica utilizada para leitura de um sinal analógico pelo Arduino é a conversão analógica digital. Essa técnica consiste em converter o sinal analógico para um valor digital, dessa forma se pode quantificar o sinal presente no pino. Esse processo é feito pelo conversor Analógico digital, ADC ou conversor A/D.

Um conversor A/D quantifica o valor analógico conforme a quantidade de bits da sua resolução. A resolução de um conversor A/D é dada pela seguinte equação:

$$resolução = V_{ref} / 2^n$$

onde:

Vref: tensão de referência do conversor A/D;
n: número de bits do conversor.

Conversor A/D do Arduino

O conversor A/D do microcontrolador ATmega328 possui 10 bits de resolução, a sua tensão de entrada pode variar de 0 V até o valor de VCC e possui referência interna selecionável de 1,1 V.

Dessa forma quando está trabalhando com a referência em VCC o menor valor que pode ser lido será:

Resolução = 5 V/1024 = 4,88 mV

esse é o valor de degrau para uma conversão em 10 bits com referência em 5 V.

Caso trabalhe com a referência interna de 1,1V a resolução será:

$$resolução = \frac{1,1}{1024} = 1,07 \ mV$$

Nota-se que o passo é bem menor para esse valor de referência.

Se a referência externa for selecionada, a resolução dependerá do valor de tensão aplicada ao pino AREF.

A placa Arduino UNO possui 6 canais de conversor analógico digital. Essas entradas são nomeadas de A0 a A5 e são exibidas na figura a seguir:

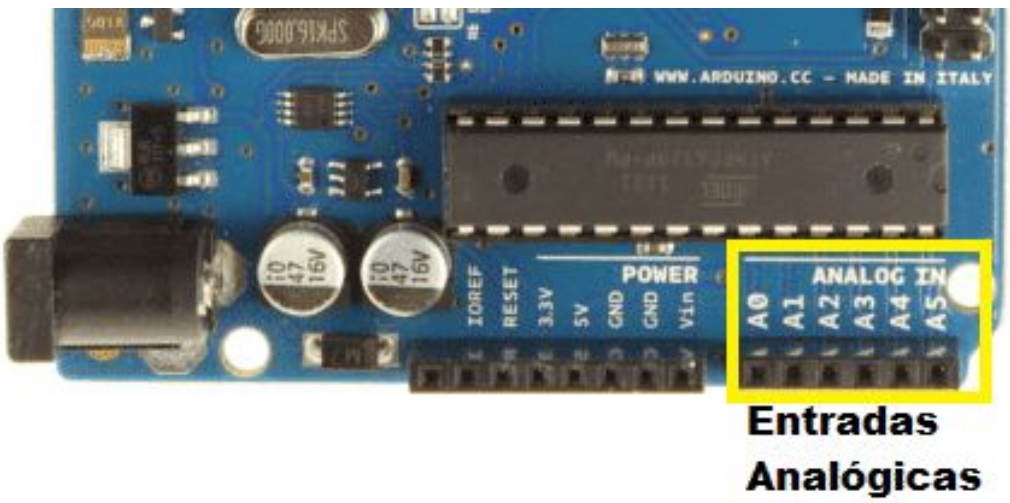


Figura 1 - Entradas analógicas da placa Arduino

Funções da plataforma Arduino para Entradas Analógicas

A plataforma Arduino possui funções para trabalhar com entradas analógicas, as quais abstraem a configuração dos registradores de configuração do conversor A/D, assim como selecionam o canal conforme o pino passado como parâmetro. São apenas duas funções e são exibidas a seguir:

- **analogReference(tipo)**

Descrição

Configura a referência de tensão para a conversão analógica/digital, usando esse valor como o máximo para a entrada analógica.

Os tipos possíveis de configurações são:

- DEFAULT: a tensão padrão para conversão é a tensão de alimentação da placa. 5 V para placas alimentadas com 5 V e 3,3 V para placas alimentadas com 3,3 V;
- INTERNAL: referência interna de 1,1V no Atmega168 e Atmega328, e 2,56 V no ATmega8;
- INTERNAL1V1: referência de 1,1V, apenas no Arduino Mega;
- INTERNAL2V56: referência interna de 5,6 V, apenas no Arduino Mega;
- EXTERNAL: referência de tensão aplicada no pino AREF (valor entre 0 e 5V).

Sintaxe:

analogReference(tipo);

Parâmetros:

tipo: DEFAULT, INTENAL, INTERNAL1V1, INTERNAL2V56, EXTERNAL.

Retorno:

Essa função não tem retorno algum.

- int analogRead(pino)

Descrição

Lê o valor presente em um pino configurado como entrada analógica. Internamente o Arduino possui um conversor A/D de 10 bits. Dessa forma o valor retornado por esta função estará na faixa de 0 a 1023 conforme o valor presente no pino.

O tempo para leitura pela função analogRead() é por volta de 100 micro segundos, dessa forma a máxima frequência de leitura que se pode ter é de 10000 vezes por segundo.

Sintaxe:

analogRead(pino);

Parâmetros:

pino: valor do pino configurado como entrada analógica (0 a 5 na maioria da placas, 0 a 7 na MINI e NANO, 0 a 15 na MEGA).

Retorno:

int (0 a 1023)

Exemplo - Entradas Analógicas com Arduino0

Para iniciar o estudo da entrada analógica, a maneira mais simples e rápida é ligando um potenciômetro a uma entrada analógica, conforme esquema apresentado em seguida:

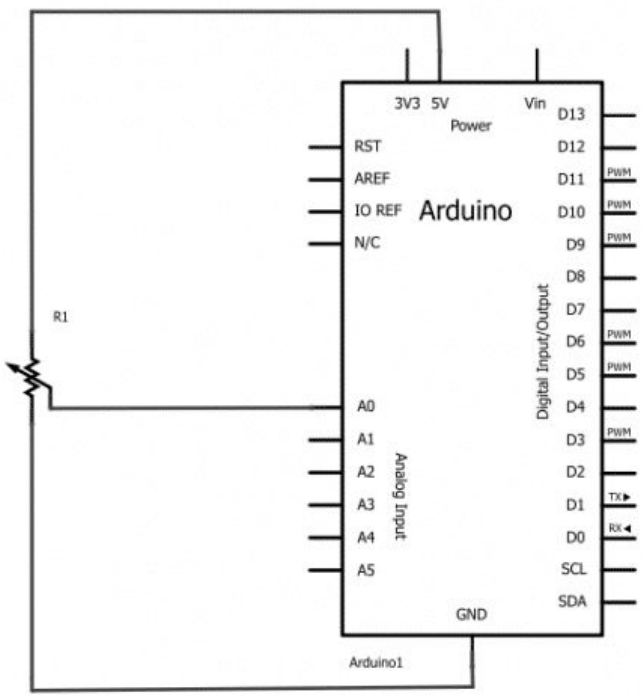
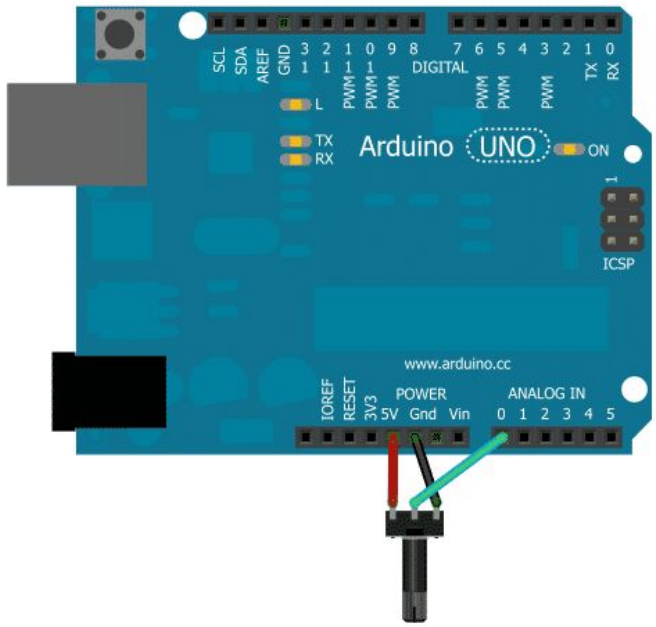


Figura 2 - Ligação de um potenciômetro a uma entrada analógica

Se girarmos o cursor do potenciômetro, alteramos a resistência em cada lado do contato elétrico que está conectado ao terminal central do botão. Isso provoca a mudança na proximidade do terminal central aos 5 volts ou GND, o que implica numa mudança no valor analógico de entrada. Quando o cursor for levado até o final da escala, teremos, por exemplo, 0 V a ser fornecido ao pino de entrada do Arduino e, assim, ao lê-lo obtém-se 0. Quando giramos o cursor até o outro extremo da escala, haverá 5 V a ser fornecido ao pino do Arduino e, ao lê-lo, teremos 1023. Em qualquer posição intermediária do cursor, teremos um valor entre 0 e 1023, que será proporcional à tensão elétrica sendo aplicada ao pino do Arduino. A ligação no Arduino UNO pode ser feita conforme a figura abaixo:



O exemplo a seguir lê o valor no potenciômetro. O tempo que o LED permanece ligado ou desligado depende do valor obtido pelo analogRead().

```
/*
* Entrada analógica
* Liga e desliga um LED conectado ao pino digital 13. O tempo
* que o LED permanece ligado ou desligado depende do valor
* obtido pelo analogRead().
*/


int potPin = 0; // selecione o pino de entrada ao potenciômetro
int ledPin = 13;   // selecione o pino ao LED
int val = 0;    // variável a guardar o valor proveniente do sensor

void setup() {

pinMode(ledPin, OUTPUT);  // declarar o pino ledPin como saída

}


void loop() {

val = analogRead(potPin); // ler o valor do potenciômetro

digitalWrite(ledPin, HIGH);  // ligar o led
delay(val);                  // espera tempo ajustado no potenciometro
digitalWrite(ledPin, LOW);   // desligar o led
delay(val);                  // espera tempo ajustado no potenciometro

}
```

O exemplo apresentado tem estrutura bem simples e serve para fixar os conceitos sobre o conversor A/D do Arduino. Aproveitamos essa estrutura para artigos futuros onde faremos a leituras de sensores.

Confira webinars gratuitos no Embarcados

Este artigo foi escrito por Fábio Souza.

EMBARCADOS

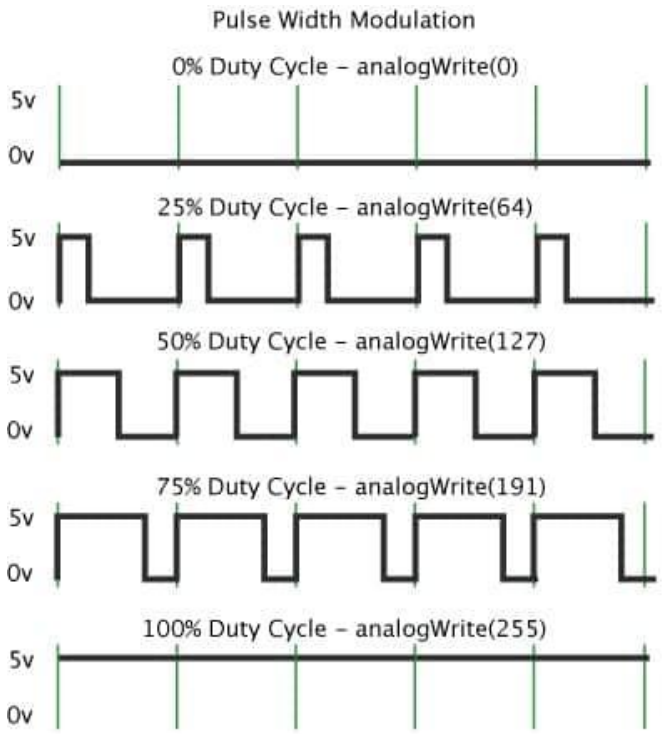
USANDO SAÍDAS PWM DO ARDUINO

por Fábio Souza

Continuando a sequência de artigos básicos sobre a plataforma Arduino, vamos aprender como utilizar sinais PWM na placa Arduino UNO. No final desse artigo você saberá o que é PWM e como usar o PWM do Arduino para controlar a intensidade de um LED. O conteúdo apresentado servirá de base para suas aplicações usando o PWM do Arduino.

O que é PWM?

PWM, do inglês Pulse Width Modulation, é uma técnica utilizada por sistemas digitais para variação do valor médio de uma forma de onda periódica. A técnica consiste em manter a frequência de uma onda quadrada fixa e variar o tempo que o sinal fica em nível lógico alto. Esse tempo é chamado de duty cycle, ou seja, o ciclo ativo da forma de onda. No gráfico abaixo são exibidas algumas modulações PWM:



Fonte: http://arduino.cc/en/Tutorial/PWM

Analisando as formas de onda nota-se que a frequência da forma de onda tem o mesmo valor e varia-se o duty cycle da forma de onda. Quando o duty cicle está em 0% o valor médio da saída encontra-se em 0 V e consequentemente para um duty cycle de 100% a saída assume seu valor máximo, que no caso é 5V. Para um duty cycle de 50% a saída assumirá 50% do valor da tensão, 2,5 V e assim sucessivamente para cada variação no duty cycle. Portanto, para calcular o valor médio da tensão de saída de um sinal PWM pode-se utilizar a seguinte equação:

$$Vout = (duty\ cycle/100)^* Vcc$$

Onde:

- Vout - tensão de saída em V;
- duty cycle - valor do ciclo ativo do PWM em %;
- Vcc - tensão de alimentação em V.

PWM pode ser usada para diversas aplicações, como por exemplo:

- controle de velocidade de motores;
- variação da luminosidade de leds;
- geração de sinais analógicos;
- geração de sinais de áudio.

PWM do Arduino

A placa Arduino Uno possui pinos específicos para saídas PWM e são indicados pelo carácter '~' na frente de seu número, conforme exibido a seguir:

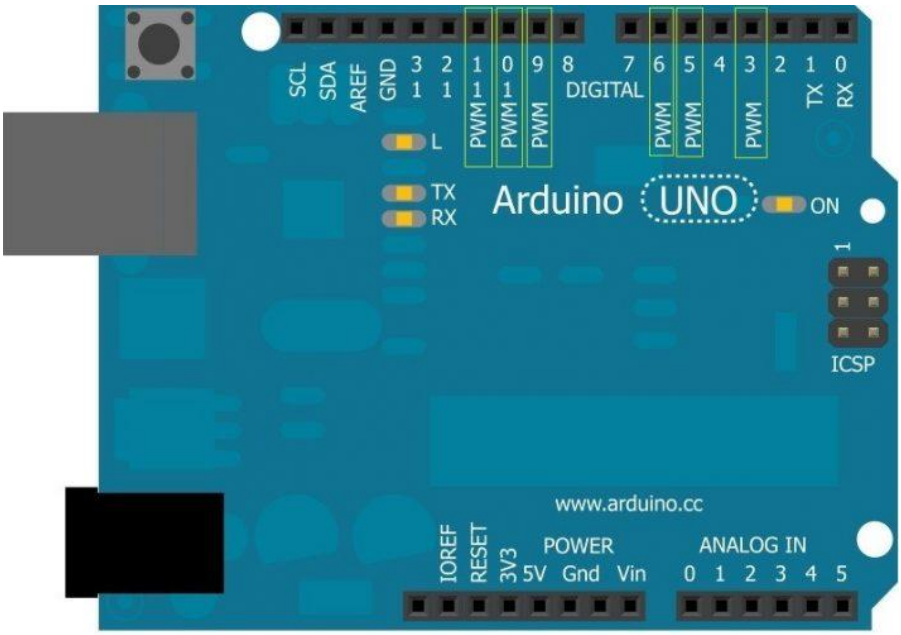


Figura 2 - Saídas PWM na placa Arduino UNO

Observa-se na figura acima, que a Arduino Uno possui 6 pinos para saída PWM (3,5,6,9,10,11). Para auxiliar na manipulação desses pinos a plataforma possui uma função que auxilia na escrita de valores de duty cycle para esses pinos, assim você pode facilmente usar o PWM do Arduino UNO e outras placas.

Função analogWrite()

A função analogWrite() escreve um valor de PWM em um pino digital que possui a função PWM. Após a chamada dessa função, o pino passa a operar com uma onda quadrada de frequência fixa e com duty cycle conforme valor passado pela função. A frequência dessa onda, na maioria dos pinos é em torno de 490 Hz, porém, os pinos 5 e 6 da Arduino UNO operam em 980 Hz.

Para utilizar a função **analogWrite()** , deve-se configurar o pino correspondente como saída digital. É interessante notar que essas saídas não são conversores digital-analógico como o nome sugere, e estes pinos não estão relacionados às entradas analógicas.

A função analogWrite deve ser utilizada da seguinte forma:

Sintaxe:

analogWrite(pino, valor);

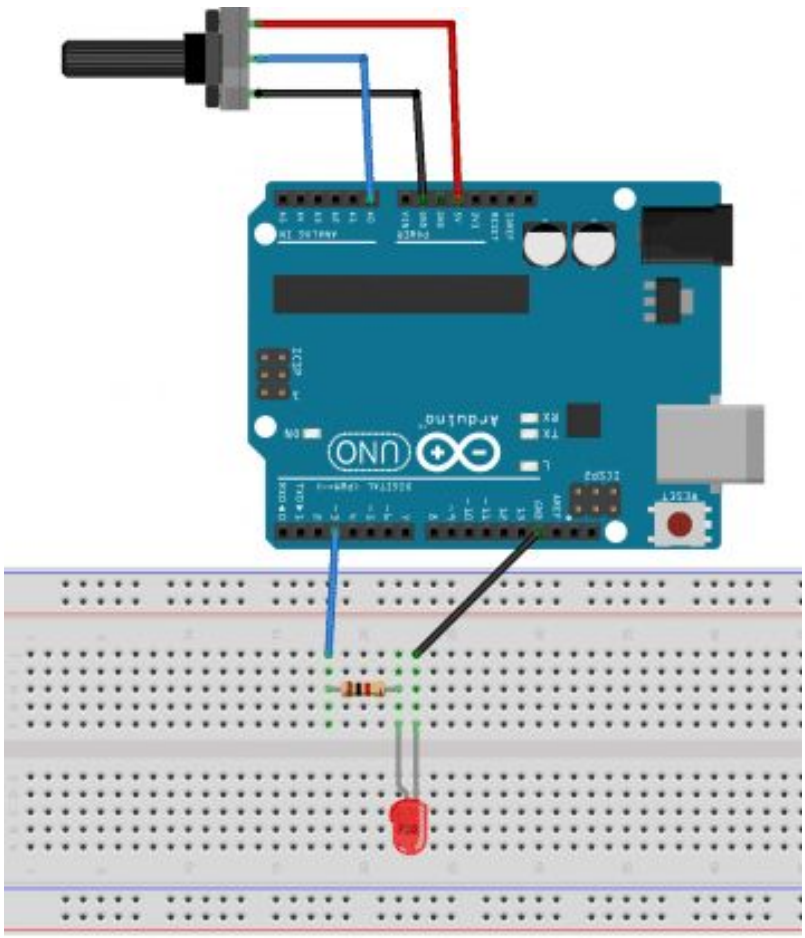
Onde:

- pino corresponde ao pino que será gerado o sinal PWM;
- valor corresponde ao duty cycle, ou seja, o valor que permanecerá em nível alto o sinal.

O valor deve ser de 0 a 255 onde com 0 a saída permanece sempre em nível baixo e 255 a saída permanece sempre em nível alto.

Exemplo - Variando o brilho de um LED usando PWM do Arduino

Vamos utilizar a montagem a seguir para exemplificar o uso de um sinal PWM para variação do brilho de um LED, com a placa Arduino UNO:



O circuito possui um LED ligado ao pino 3 (~:PWM) com seu devido resistor e um potenciômetro ligado à entrada analógica 0. A proposta é controlar a intensidade do brilho do LED através da variação do valor do potenciômetro. Vejamos o sketch a seguir:

```
/*
PWM
controla a luminosidade de um led conforme o valor do potenciometro
*/
int ledPin = 3; // pino do led
int analogPin = 0; // pino para leitura do potenciômetro
int val = 0; //variável para armazenar o valor lido
void setup()
{
pinMode(ledPin, OUTPUT); // configura pino como saída
}
void loop()
{
val = analogRead(analogPin); // le o valor analógico
analogWrite(ledPin, val / 4); // aciona led com o valor analógico
lido
//dividido por 4 para ajustar ao
valor
//máximo que pode ser atribuído a
função
}
```

Conclusões sobre o PWM do Arduino

A função **analogWrite()** fornece um modo simples para se trabalhar com sinais PWM, porém não fornece nenhum controle sobre a frequência do sinal aplicado ao pino. Em alguns casos a frequência do sinal é muito importante para o sistema, como por exemplo a frequência de acionamento de uma bobina de um motor. Em um artigo futuro vamos abordar como manipular os registradores do ATmega328 para alterar a frequência do sinal PWM.

O exemplo apresentado acima exibiu como usar o PWM do Arduino para variar a intensidade de um LED. Você pode usar o mesmo programa para variar a velocidade de motores DC, adaptar para criar cores usando LEDs RGBs, etc. Use a imaginação para novos projetos e os coloque em prática.

Este artigo foi escrito por Fábio Souza.

Publicado originalmente no Embarcados, no dia 10/01/2014: link para o artigo original, sob a Licença Creative Commons Atribuição-CompartilhaIgual 4.0 Internacional.

EMBARCADOS

COMUNICAÇÃO SERIAL

por Fábio Souza

Continuando a série de artigos introdutórios sobre a plataforma Arduino, neste vamos abordar a comunicação serial. Serão apresentadas as principais funções e alguns exemplos básicos que serão usados como base em nossos projetos.

Arduino Comunicação Serial

A comunicação serial (UART) na plataforma Arduino é, sem dúvida, um poderoso recurso que possibilita a comunicação entre a placa e um computador ou entre a placa e outro dispositivo, como por exemplo um módulo GPS ou um módulo GSM. É através desse canal que é realizado o upload do código para a placa.

A placa Arduino UNO possui um canal de comunicação por hardware, conforme foi exibido no artigo publicado sobre Arduino UNO. Esse canal está ligado aos pinos digitais 0 (RX) e 1 (TX). Esses mesmos pinos estão ligados ao microcontrolador ATMEGA16U2, responsável pela tradução do sinal para comunicação USB com o computador.

A figura a seguir exibe os pinos de comunicação serial na placa Arduino UNO:

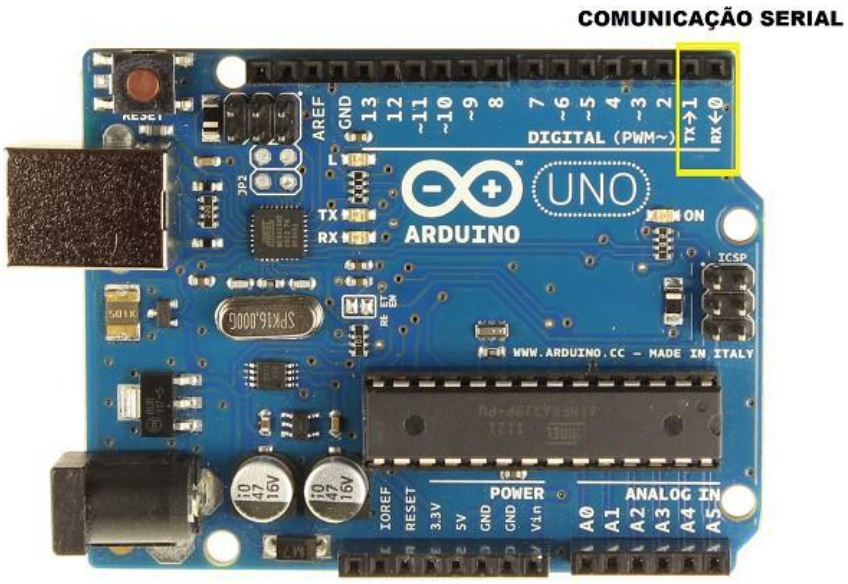


Figura 1 - Pinos para comunicação serial

Deve-se ficar atento a dispositivos conectados a esses pinos pois podem interferir no upload do seu programa. Em alguns caso é recomendável desconectar os dispositivos ou shields ligados a esses pinos antes de fazer o upload.

O sinal de comunicação na placa Arduino UNO, é um sinal TTL de 5V. Para comunicação com um computador ou outro dispositivo que não tenha o mesmo nível de tensão é necessário um conversor de nível. Existem várias opções de conversores, como por exemplo TTL/ RS232, TTL/RS485, TTL/USB, entre outros. A seguir são exibidos alguns modelos de conversores:

Figura 2 - Shields e módulos para comunicação serial

Terminal Serial

Além do recurso de upload através da comunicação serial, a IDE trás um terminal serial que auxilia no recebimento e envio de dados para a placa sem a necessidade de recorrer a uma ferramenta externa. Para acessar essa ferramenta basta clicar no ícone Serial Monitor ou acessar o menu Tools> Serial Monitor. É aberta a janela a seguir:

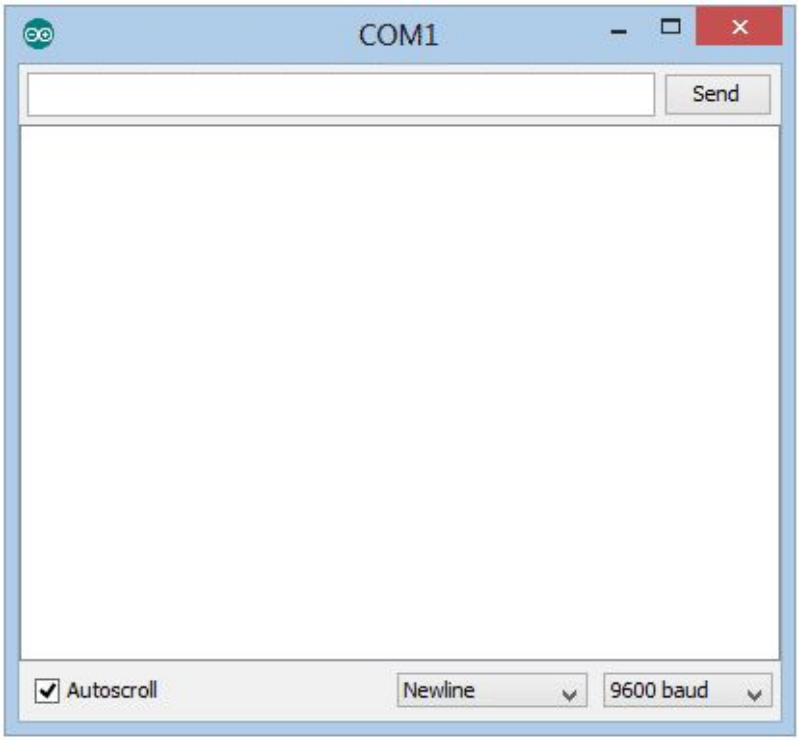


Figura 3 - Terminal para comunicação serial do Arduino

A ferramenta é bem simples, contendo apenas alguns parâmetros de configuração, onde se pode definir a taxa de envio (baud rate). Possui dois campos, um onde pode ser inserido a mensagem a ser enviada e outro maior onde é exibido os caracteres enviados pela placa para o computador.

Funções da plataforma Arduino

A plataforma Arduino possui em sua biblioteca uma variedade de funções para manipulação de dados através de comunicação serial. Essas funções auxiliam o desenvolvedor em tarefas mais complexas de envio e recebimento de dados.

Funções mais usadas com a placa Arduino UNO

Serial.begin()

É a primeira função a ser utilizada quando vai trabalhar com a comunicação serial. Ela configura a taxa de comunicação em bits por segundo (baud rate). Possui um segundo parâmetro opcional para a definição da quantidade de bits, paridade e stop bits. Se for omitido esse parâmetro o padrão será 8 bits, sem paridade e 1 stop bit.

Sintaxe:

Serial.begin(speed)
Serial.begin(speed, config)

Parâmetros:

speed: velocidade em bit por segundo (baud rate) - long
config: configura a quantidade de bits, paridade e stop bits. Os valores válidos são :

- SERIAL_5N1
- SERIAL_6N1
- SERIAL_7N1
- SERIAL_8N1 (padrão)
- SERIAL_5N2
- SERIAL_6N2
- SERIAL_7N2
- SERIAL_8N2
- SERIAL_5E1
- SERIAL_6E1
- SERIAL_7E1
- SERIAL_8E1
- SERIAL_5E2
- SERIAL_6E2
- SERIAL_7E2
- SERIAL_8E2
- SERIAL_5O1
- SERIAL_6O1
- SERIAL_7O1
- SERIAL_8O1
- SERIAL_5O2
- SERIAL_6O2
- SERIAL_7O2
- SERIAL_8O2

Retorno

Essa função não retorna nada.

Serial.available()

Retorna a quantidades de bytes disponíveis para leitura no buffer de leitura. Essa função auxilia em loops onde a leitura dos dados só e realizada quando há dados disponível. A quantidade máxima de bytes no buffer é 64.

Sintaxe:

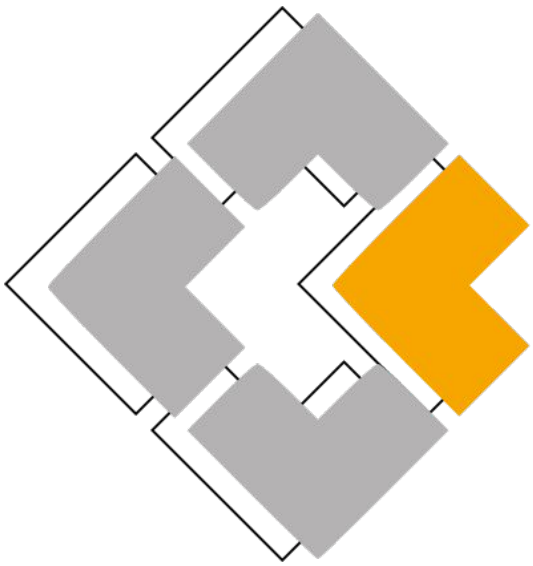
Serial.available();

Parâmetros:

Não passa nenhum parâmetro.

Retorno:

(int) - quantidade de bytes disponíveis para leitura



Serial.read()

Lê o byte mais recente apontado no buffer de entrada da serial.

Sintaxe:

Serial.read();

Parâmetros:

Não passa nenhum parâmetro.

Retorno:

(int) - O primeiro byte disponível no buffer da serial. Retorna -1 caso n tenha dado disponível.

Serial.print()

Escreve na serial texto em formato ASCII. Essa função tem muitas possibilidades. Números inteiros são escritos usando um caractere ASCII para cada dígito. O mesmo ocorre para números flutuante e, por padrão, são escritas duas casas decimais. Bytes são enviados como caracteres únicos e strings e caracteres são enviados como escritos.

Vejamos alguns exemplos:

```
Serial.print ( 123 ); // Envia "123"
Serial.print ( 1.234567 ); // Envia "1.23"
Serial.print ( 'N' ); // Envia "N".
Serial.print ( "Hello world" ); // Envia "Hello world".
```

Obs.: caracteres são enviados com aspas simples e strings com aspas duplas.

Um segundo parâmetro opcional define a base numérica para formatar o valor enviado. São aceitos os seguintes parâmetros:

- BIN - binário, base 2
- OCT - octal, base 8
- HEX - hexadecimal, base 16
- DEC - decimal, base 10

Para números em ponto flutuante esse parâmetro define a quantidade de casas decimais a serem enviadas após o ponto. Exemplos:

- Serial.print(78, BIN) envia em binário "1001110"
- Serial.print(78, OCT) envia emr octal "116"
- Serial.print(78, DEC) envia em decimal "78"
- Serial.print(78, HEX) envia em hexadecimal "4E"
- Serial.println(1.23456, 0) envia apenas "1", sem casas decimais
- Serial.println(1.23456, 2)envia "1.23", ou seja, duas casas decimais
- Serial.println(1.23456, 4) envia "1.2346", ou seja, 4 casas decimais

Sintaxe:

Serial.print(val)
Serial.print(val, format)

Parâmetros:

val:valor para ser escrito na serial - qualquer tipo de dado.
format: base numérica para tipos inteiros ou a quantidade de casas decimais para números flutuantes.

Retorno:

size_t (long): retorna a quantidade de bytes escritos

Serial.println()

Funciona praticamente igual a função Serial.print(), a única diferença é que esta função acrescenta ao fim da mensagem o caractere de retorno de carro (ASCII 13 ou '\r') e o caractere de nova linha(ASCII 10 ou '\n'). A sintaxe, os parâmetros e o retorno são os mesmos da função Serial.print().

Serial.write()

Escreve um byte na porta serial.

Sintaxe:

Serial.write(val)
Serial.write(str)
Serial.write(buf, len)

Parâmetros:

val: um valor para ser enviado como um único byte.
str: uma string para ser enviada como uma sequência de bytes.
buf: um array para ser enviado como uma serie de bytes.
len: o tamanho do buffer a ser enviado.

Retorno:

(byte) - Retorna a quantidade de bytes escritos na serial. A leitura deste número é opcional.

Manipulação de dados através da comunicação serial

Exemplos:

Echo

No Sketch a seguir é exibido como receber um caractere do computador e enviar este mesmo caractere para o computador, onde será exibido o que é digitado na terminal serial.

Exemplos:

```
/*
  echo
  reenvia para o computador o dado recebido pela serial
*/

byte byteRead;

void setup() {
//configura a comunicação seria com baud rate de 9600
  Serial.begin(9600);
}

void loop() {

  if (Serial.available())  //verifica se tem dados diponível para leitura
  {
    byteRead = Serial.read(); //le bytwe mais recente no buffer da serial
    Serial.write(byteRead);   //reenvia para o computador o dado recebido
  }
}
```

Neste exemplo foi utilizada a função Serial.available() para verificar se há dado disponível no buffer da serial. Quando há um byte para leitura, o mesmo é lido pela função Serial.read() e armazenado na variável byteRead. A próxima função, Serial.write(), imprime de volta o dado recebido para o computador.

Dimmer

Este exemplo demonstra como enviar dados do computador para controlar o brilho de um LED conectado a uma saída PWM. Este exemplo já vem com a plataforma e pode ser acessado em: File -> Examples -> 04.Communication -> Dimmer.

```
/*
Dimmer
Demonstra como enviar dados do computador para controlar a intesindade
do brilho de um led conctado ao pino 9 do arduino.
*/

const int ledPin = 9;      // the pin that the LED is attached to

void setup()
{
  Serial.begin(9600); // configura a comunicação serial com 9600 bps
  pinMode(ledPin, OUTPUT);  // configura pino do led como saída
}

void loop() {
  byte brightness;  //cria uma variável para armazernar byte recebido

  if (Serial.available()) //verifica se chegou algum dado na serial
  {
    brightness = Serial.read();//Lê o byte mais recente disponível na serial
    analogWrite(ledPin, brightness);//atualiza a saída PWM do LED com valor recebido
  }
}
```

Com este exemplo pode -se varia o brilho do LED conectado à saída PWM através de comandos enviados pelo PC. O byte recebido pela serial é atribuído a variável brightness, que na instrução a seguir é passado como parâmetro na função analogWrite(), definindo o brilho do LED. Junto com este exemplo é exibido um código em processing para variação do brilho através do clique do mouse no PC.

Liga/Desliga LED

Este exemplo exibe como ligar e desligar um LED conectado as saída digital da Arduino UNO através de comando enviados pelo computador.

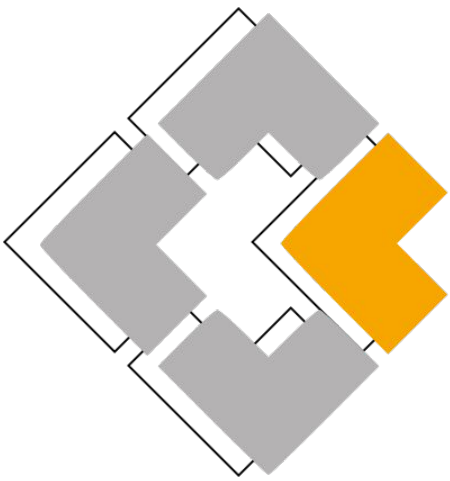
```
/*
* comandos via serial
* inverte o estado do led conctado a saída 13 do arduino quando recebe o caracter 'A' pela serial
*/

const int LED = 13;

void setup() {
  Serial.begin(9600);    //configura comunicação serial com 9600 bps
  pinMode(LED,OUTPUT);   //configura pino do led como saída
}

void loop() {
   if (Serial.available()) //se byte pronto para leitura
   {

    switch(Serial.read())      //verifica qual caracter recebido
    {
      case 'A':                  //caso 'A'

        digitalWrite(LED,!digitalRead(LED)); //inverte estado do LED

      break;
    }
  }
}
```

Neste exemplo o estado do LED ligado ao pino 13 é invertido sempre que o Arduino recebe o caractere 'A'. A estrutura desse sketch permite adicionar mais saídas para serem acionadas. Este exemplo pode ser aproveitado para uma aplicação gráfica no PC para controlar cargas automação residencial, por exemplo.

Conclusão sobre comunicação serial na plataforma Arduino

Dominar a comunicação serial na plataforma Arduino é essencial para o desenvolvedor de projetos. Muitos dispositivos e módulos possuem uma interface de comunicação, seja para configuração ou para comandos.

O uso de uma comunicação serial permite também o controle ou monitoramento de sistemas utilizando o computador ou mesmo outra placa eletrônica. É uma interface tradicional e bem conhecida e permite que ligue, de forma simples diferentes dispositivos.
A comunicação serial na plataforma Arduino aliada ao terminal da IDE, se torna uma ótima ferramenta para visualização de dados e Debug durante o processo de desenvolvimento, já que a plataforma não possui tais ferramentas de depuração.



Este artigo foi escrito por Fábio Souza.

EMBARCADOS

CRIANDO SUAS PRÓPRIAS BIBLIOTECAS PARA ARDUINO

por Fábio Souza

A plataforma Arduino trouxe a facilidade de criar projetos com microcontroladores, principalmente para iniciantes, devido à abstração de código de baixo nível. Quando você utiliza uma função para escrita ou leitura de um sinal digital, você não precisa se preocupar com a correta manipulação dos registradores internos do microcontrolador que está utilizando em seu projeto. Essa abstração apresenta vantagens, como a portabilidade de código, e desvantagens, como, por exemplo, o possível aumento do código de programa gerado e o consequente atraso para manipulação de sinais.

Outra vantagem de se utilizar bibliotecas é a abstração de hardware, uma vez que você pode manipular o hardware com métodos mais próximos à nossa linguagem.

Nesse tutorial vamos apresentar como criar uma biblioteca para manipular um pino de saída de forma mais alto nível.

As bibliotecas para Arduino são feitas em C++. Inicialmente deve-se criar uma pasta com o mesmo nome que vai ser chamada a biblioteca.
A pasta examples conterá os exemplos que apareceram na IDE do Arduino. O Arquivo keywords.txt servirá para as palavras da biblioteca mudarem de cor na IDE. Os arquivos Saída.h e Saída.cpp conterão os códigos da biblioteca. Vamos começar a estruturar nossa biblioteca Saída.

Inicialmente vamos pensar nas ações que uma saída digital poderá demonstrar:

- ligar;
- desligar;
- inverter o seu estado.

Agora temos que editar o arquivo Saida.h, que é o arquivo de cabeçalho da nossa biblioteca. Primeiro vamos inserir as seguintes diretivas de compilação:

```
#ifndef SAIDA_H
#define SAIDA_H



#endif
```

Essas diretivas não deixarão as declarações/definições da biblioteca serem inseridas mais de uma vez em um projeto.

Para ter acesso às funções do Arduino, é necessário fazer uso da biblioteca Arduino, inserindo o arquivo de cabeçalho Arduino.h:

```
#ifndef SAIDA_H
#define SAIDA_H

#include <Arduino.h>



#endif
```

Agora vamos criar a classe Saida:

```
#ifndef SAIDA_H
#define SAIDA_H

#include <Arduino.h>

class Saida
{
public:
   Saida(int pin);
   void  liga();
   void  desliga();
   void  inverte();

private:
   int pino;
};

#endif
```

Como pode ser observado, a classe Saida possui um construtor que recebe como parâmetro o pino correspondente a saída. Possui também 3 métodos públicos que poderão ser acessados por quem for utilizar a biblioteca e um atributo privado que só poderá ser acessado dentro da classe.

Vamos agora para a codificação dos métodos da classe no arquivo *Saida.cpp*:

```
#include"Saida.h"

Saida::Saida(int pin)
{
   pinMode(pin, OUTPUT);
   pino = pin;
}

void Saida::liga()
{
   digitalWrite(pino,HIGH);
}

void Saida::desliga()
{
   digitalWrite(pino,LOW);
}

void Saida::inverte()
{
   digitalWrite(pino, !digitalRead(pino));
}
```

O construtor Saida::Saida(int pin) configura o pino passado como parâmetro como saída e depois atribui o seu valor à variável privada pino, de modo que esse pino possa ser utilizado pelos métodos da classe futuramente.

Os métodos demonstram as ações que os seus nomes propõem, fazendo uso das funções da biblioteca Arduino.

O arquivo keywords.txt deve ficar da seguinte forma:

```
Saida KEYWORD1
liga KEYWORD2
desliga KEYWORD2
inverte KEYWORD2
```

O nome da classe deve estar na linha de KEYWORD1 e os métodos serão KEYWORD2. Agora que nossa biblioteca está pronta, basta adicionar a pasta libraries no diretório do Arduino e vamos criar dois exemplos para testes. Esses exemplos devem ser salvos na pasta examples da nossa biblioteca.

O primeiro exemplo é SaidaBlink, que consiste em piscar um led em intervalos de 1 segundo:

```
#include <Saida.h>

// Instancia um objeto chamado LED no pino 13
Saida LED(13);

void setup(){
}

void loop()
{
 LED.liga();        // liga o led
 delay(1000);       // aguarda 1 segundo
 LED.desliga();     // desliga o Led
 delay(1000);       // aguarda 1 segundo
}
```

No segundo exemplo é criada uma saída chamada rele, que inverte seu estado quando uma tecla for pressionada:

```
#include <Saida.h>


 // Cria um rele passando como parâmetro o pino no qual está ligado

Saida rele(8);
const byte tecla = 2;

// Configura arduino

void setup()
{
 pinMode(tecla, INPUT);
}


// Loop principal
void loop()
{
 if (digitalRead(tecla) == LOW)
 {
   while (digitalRead(tecla)  == LOW); // Aguada tecla ser liberada
   rele.inverte();
 }
}
```

Conclusão

O exemplo apresentado para criação de bibliotecas foi bem simples e servirá de base para que você possa criar as suas próprias bibliotecas, ou caso você queira entender como uma biblioteca é criada. Você pode adicionar outros métodos a essa biblioteca e testar com o seu hardware, por exemplo, um método que retorne o estado atual de uma saída, etc.

Este artigo foi escrito por Fábio Souza.

Considerações Finais

Chegamos ao final do nosso ebook.

Caso você tenha alguma dúvida, não deixe ela sem resposta. Você pode entrar em contato com o autor através da seção de comentários do artigo, ou então participar da nossa comunidade, interagindo com outros profissionais da área:
Comunidade Embarcados no Facebook
Comunidade Embarcados no Telegram

Caso você tenha encontrado algum problema no material ou tenha alguma sugestão, por favor, entre em contato conosco. Sua opinião é muito importante para nós:

contato@embarcados.com.br

Siga o Embarcados na Redes Sociais

facebook.com/osembarcados/

instagram.com/portalembarcados/

youtube.com/embarcadostv/

linkedin.com/company/embarcados/

twitter.com/embarcados