Arduino – Comunicação Serial

Por Fábio Souza – 21/01/2014

ÍNDICE DE CONTEÚDO

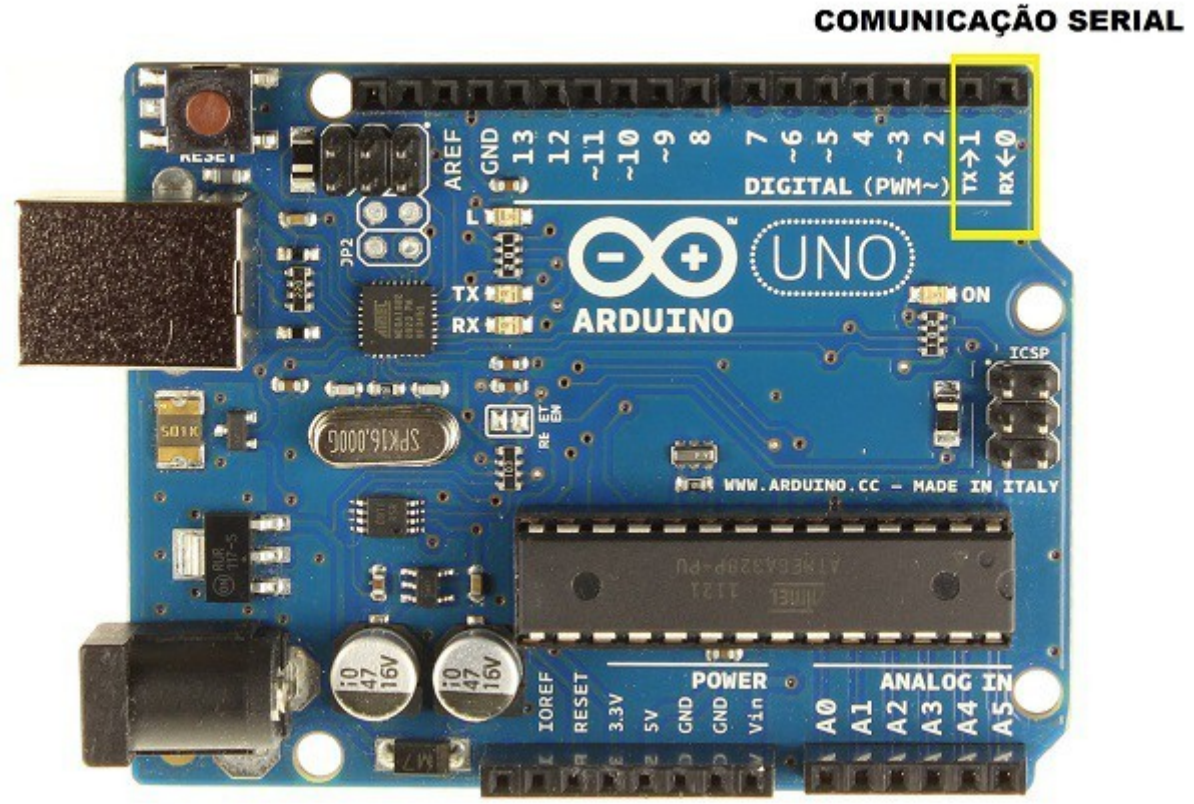
Continuando a serie de artigos introdutórios sobre a plataforma Arduino, neste vamos abordar a comunicação serial. Serão apresentadas as principais funções e alguns exemplos básicos que serão usados como base em nossos projetos.

1. Arduino Comunicação Serial

A comunicação serial (UART) na plataforma Arduino é, sem duvida, 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.

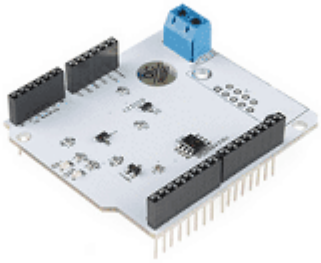
A placa Arduino UNO possui um canal de comunicação por hardware, conforme foi exibido no artigo publicado sobre a Arduino UNO. Esse canal está ligado aos pinos digitais 0 (RX) e 1 (TX). Esses mesmos pinos estão ligados ao microcontrolador ATMEGA16U2, responsável pela tradução do sinal para comunicação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

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



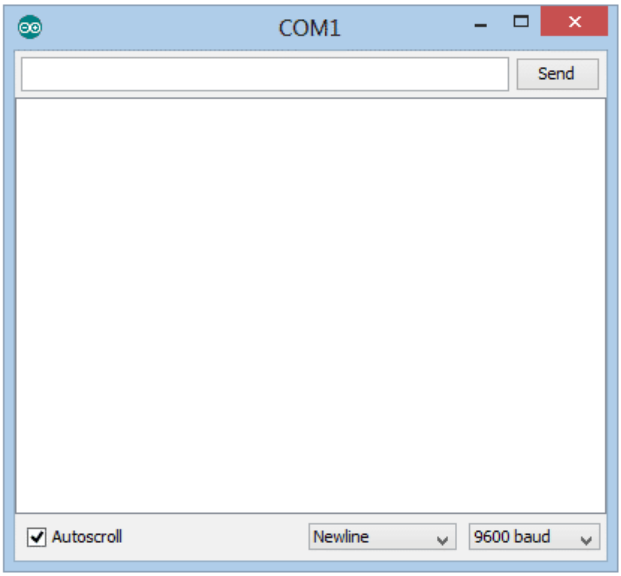
Deve-se ficar atento a dispositivos conectados a esses pinos pois podem interferir no upload do seu programa. Em alguns caso é recomendável desconectar os dispositivos ou shields ligados a esses pinos antes de fazer o upload.

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:


2. Terminal Serial

Além do recurso de upload através da comunicações 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:



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.

3. 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.

4. 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 escrito 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 desse numero é opcional.

5. 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.

```
1   /*
2     echo
3     reenvia para o computador o dado recebido pela serial
4   */
5
6   byte byteRead;
7
8   void setup() {
9   //configura a comunicação seria com baud rate de 9600
10    Serial.begin(9600);
11  }
12
13  void loop() {
14
15    if (Serial.available())  //verifica se tem dados diponível para leitura
16    {
17      byteRead = Serial.read(); //le bytwe mais recente no buffer da serial
18      Serial.write(byteRead);   //reenvia para o computador o dado recebido
19    }
20  }
```

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 variavel 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.*

```
1   /*
2   Dimmer
3   Demonstra como enviar dados do computador para controlar a intesindade
4   do brilho de um led conctado ao pino 9 do arduino.
5   */
6
7   const int ledPin = 9;      // the pin that the LED is attached to
8
9   void setup()
10  {
11    Serial.begin(9600); // configura a comunicação serial com 9600 bps
12    pinMode(ledPin, OUTPUT);  // configura pino do led como saída
13  }
14
15  void loop() {
16    byte brightness;  //cria uma variável para armazernar byte recebido
17
18    if (Serial.available()) //verifica se chegou algum dado na serial
19    {
20      brightness = Serial.read();//Lê o byte mais recente disponível na serial
21      analogWrite(ledPin, brightness);//atualiza a saída PWM do LED com valor recebido
22    }
23  }
```

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.

6. 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.

Fonte: https://www.embarcados.com.br/arduino-comunicacao-serial/