

Comunicação serial entre Matlab e Arduino

Autor: Gabriel Daltro Duarte

e-mail: gdaltrod@gmail.com

1ª Edição - maio/2017

1 Introdução

Esse trabalho tem como objetivo mostrar como estabelecer uma comunicação serial entre a placa Arduino e o software Matlab. Para exemplificar como estabelecer essa comunicação, escolheu-se a seguinte aplicação: Utilizando o Matlab, escolhe-se qual dos três LED conectados ao Arduino deverá ser aceso. Ao Arduino estão conectados um LED amarelo, um LED verde e um LED vermelho. Após acender o LED escolhido, o Arduino envia ao Matlab um string confirmando que o LED foi aceso.

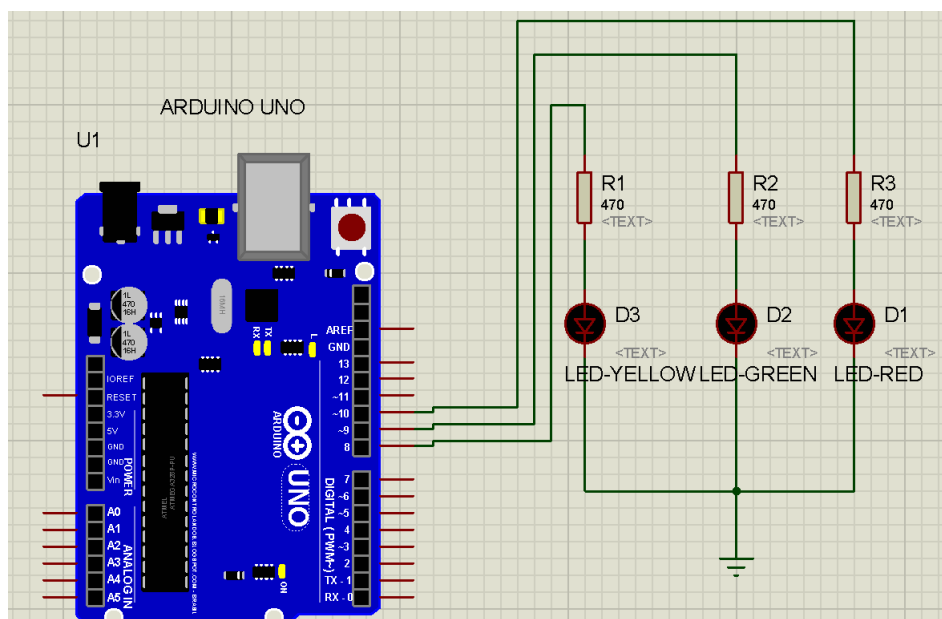


Figura 1 – Esquemático do circuito do projeto

Os códigos desenvolvidos nesse projeto foram testados utilizando o Matlab R2015a e o Matlab R2011a, ambos rodando no Windows 7. A versão do Arduino utilizada foi o Arduino UNO. O Arduino conecta-se ao computador utilizando a porta USB, porém os programas que rodam no Windows enxergam a conexão Arduino-Computador como uma conexão serial simples. Isso se deve ao fato de ocorrer uma emulação de uma porta serial

COM. Assim, é possível utilizar as funções do Matlab destinadas a comunicação serial com periféricos para enviar e receber dados para o Arduino pela porta USB.

Os códigos apresentados nesse trabalho estão disponíveis no seguinte link:

[<https://github.com/GabrielDaltro/SerialCommunicationMatlabArduino>](https://github.com/GabrielDaltro/SerialCommunicationMatlabArduino)

2 Script do Matlab

O script executado pelo Matlab para comunicar-se com o Arduino é mostrado abaixo:

```
1
2 %Autor: Gabriel Daltro Duarte
3 %Data: 12 de maio de 2017
4 %Local: Paraiba - Brasil
5 clc;
6
7 porta_serial = serial ('COM5', 'BaudRate', 9600);
8
9 fopen(porta_serial);
10
11 n = input ('Escolha qual LED deseja acender (1(amarelo),2(verde) ou ...
           3(vermelho)): ');
12
13 while ( (strcmp (fscanf(porta_serial,'%s'),'SIM')) == 0 )
14     %disp ('Esperando SIM');
15 end
16
17 fwrite(porta_serial,[n],'uint8');
18
19 while ( porta_serial.BytesAvailable == 0 )
20     %disp ('ESPERANDO DADOS...');
21 end
22
23 stringRecebido = fscanf(porta_serial,'%s');
24
25 disp (stringRecebido);
26
27 fclose(porta_serial);
```

O programa inicia-se com a criação de um objeto do tipo Serial Port (porta serial) usando a função **serial()** na linha 7. A sintaxe da função **serial()** é a seguinte:

```
1 obj = serial('port');
2 obj = serial ('port', 'Nome da propriedade', 'valor da propriedade', ...);
```

A função retorna um objeto do tipo **serial port** (porta serial) associado a porta serial especificada pelo parâmetro **'port'**. Se a porta designada por **'port'** não existir no computador ou se estiver em uso por outra aplicação, o objeto **obj** não será capaz de se conectar a porta especificada por **'port'**. A função deve sempre receber como primeiro parâmetro **'port'**. Na chamada da função **serial()**, na linha 7, o parâmetro **'port'** recebe o string **'COM5'**, pois foi esse o nome dado pelo Windows à conexão entre o computador e o Arduino nesse projeto. Para verificar qual nome o Windows deu a conexão Arduino-Computador, verifique o Painel de Controle do Windows. A conexão Arduino-Computador

sempre receberá um nome do tipo 'COMX' onde X é um número escolhido pelo Windows. O objeto retornado pela função **serial()** é salvo na variável **porta_serial**.

Outro importante parâmetro da função **serial** é o **BaudRate** que especifica a taxa de transmissão de dados da porta serial em bits/segundo. Na chamada da função **serial()**, o **BaudRate** é definido como a taxa de 9600 bits/segundo. Para que a comunicação serial seja possível, o **BaudRate** do Arduino deve ser configurado com a mesma taxa.

Alguns outros parâmetros de configuração das propriedades da porta serial estão disponíveis no Anexo. Eles não serão tratados nessa seção, porque por padrão já assumem valores de configuração compatíveis com o Arduino, não sendo necessário alterá-los.

Na linha 9, ocorre a chamada da função **fopen()**. Essa função conecta o objeto **porta_serial** ao disposto conectado à porta serial 'COM5', que em nosso projeto é o Arduino. Só após a chamada dessa função, é possível enviar e receber dados do Arduino. Na linha 11, ocorre a leitura de qual LED será aceso usando a função **input()**. O usuário do script deve digitar um valor entre 1 e 3 que será salvo em uma variável chamada **n**.

Após a chamada da função **input()** e antes de enviar qualquer dado para o Arduino, o Matlab tem que garantir que a comunicação serial foi completamente inicializada no Arduino. Dessa forma, após a chamada da função **input()**, o Matlab aguarda que o Arduino envie um string **SIM**, sinalizando que a comunicação foi completamente inicializada. O processo de espera desse string está implementado no código através do laço **while** presente na linha 13. Na condição de teste do **while**, a função **fscanf()** recebe dois parâmetros, **porta_serial** que indica que ela deverá lê o buffer da porta serial 'COM5' e o parâmetro '%s' que indica que os dados a serem lidos são do tipo string. O string lido é comparado com o string 'SIM' usando a função **strcmp()** que retornará 0 se os strings forem diferentes e retornará 1 se os strings forem iguais. O laço se repetirá até que o string lido seja igual a 'SIM'.

Após o recebimento de **SIM**, o Matlab envia o conteúdo da variável **n** para o Arduino utilizando a função **fwrite()** chamada na linha 17. **fwrite()** recebe 3 parâmetros: o primeiro é o objeto **porta_serial** que indica que a escrita deve ser feita em 'COM5'; O segundo parâmetro é um vetor constituído pelos dados que devem ser enviados à porta serial, em nosso projeto, esse vetor é constituído unicamente pela variável **n**; O terceiro parâmetro é 'uint8' que indica que os dados a serem enviados são números inteiros sem sinal de 8 bits. Para cada tipo de dado, há um string correspondente que deve ser passado como terceiro parâmetro. Para vê a lista completa consulte "<https://www.mathworks.com/help/matlab/ref/serial.fwrite.html>".

A função **fwrite()** permite o envio de caracteres, inteiros e ponto flutuantes através da porta serial. Para enviar strings deve-se utilizar a função **fprintf()**. Abaixo segue um exemplo de envio de string usando **fprintf()**.

```
1 fprintf(porta_serial, 'RS232');
```

O exemplo acima também poderia ter sido implementado da seguinte forma:

```
1 fprintf(porta_serial, '%s', 'RS232');
```

Voltando ao código do projeto, após o envio da variável **n** pelo Matlab, o Arduino irá acender o LED correspondente ao conteúdo da variável e enviará um string de confirmação.

O Matlab aguarda o string de confirmação enviado pelo Arduino através do laço **while** da linha 19. O atributo **BytesAvailable** do objeto **porta_serial** informa a quantidade de bytes recebidos e não lidos pelo Matlab. Enquanto o valor de **BytesAvailable** for igual a zero, o Matlab não recebeu nenhum string de confirmação do Arduino e o programa permanece no laço **while** até que o string de confirmação seja recebido. Após o recebimento do string, seu conteúdo é lido na linha 23 através da função **fscanf()** e salvo na variável **stringRecebido**.

Na linha 25, o string de confirmação armazenado na variável **stringRecebido** é exibido e na linha 27 a conexão entre o Matlab e o Arduino é encerrada utilizando a função **fclose()**. A chamada dessa função é necessária para liberar a porta **'COM5'** para outras aplicações, como por exemplo o SerialMonitor da IDE do Arduino.

A função **fscanf()** permite a leitura de string e caracteres (ASCII) recebidos através da porta serial, porém não permite a leitura de números inteiros ou ponto flutuantes. Para realizar a leitura desse tipo de dados deve-se utilizar a função **fread()**. Abaixo segue um exemplo de utilização da função **fread()**. O primeiro parâmetro passado para a função é o objeto do tipo **serial port** que indica de qual porta serial será feita a leitura, o segundo parâmetro é a quantidade de dados lidos, e o terceiro parâmetro é um string que indica o tipo dos dados lidos. Como dito anteriormente, o parâmetro **uint8** indica que os dados lidos são números inteiros sem sinal de 8 bits. Para ter acesso a todos os possíveis string que podem ser passados para **fread()** e representam tipos de dados acesse "<https://www.mathworks.com/help/matlab/ref/serial.fread.html>".

```
1 dadosRecebidos = fread (porta_serial,1, 'uint8');
```

Resumindo, as funções **fscanf()** e **fprintf()** permitem a leitura e a escrita, respectivamente, de strings ou caracteres (ASCII) e as funções **fread()** e **fwrite()** permitem a leitura ou a escrita, respectivamente, de números binários.

Uma importante observação sobre a função **fscanf()** é que ela sempre espera que os dados (caracteres ou strings) recebidos possuam o caractere de finalização **'\n'** (line feed). Caso os dados recebidos não possuam esse caractere, a função irá esperar por um tempo pre-determinado, definido através do atributo **timeout** do objeto serial port, a chegada do caractere, atrasando o processo de leitura e emitindo um "warning" após o estouro desse tempo. Para evitar esse problema, ao realizar leituras com a função **fscanf()**, sempre envie caracteres ou strings que possuam no final o caractere **'\n'**. Ao enviar dados para o Matlab usando o Arduino, não é preciso colocar **'\n'** explicitamente no final dos strings, basta enviar o string com a função **Serial.println()** que ela fará isso automaticamente.

3 Programa do Arduino

A função **void setup()** do programa executado pelo Arduino é mostrado abaixo.

```
1
2 int ledA = 8; // Pino do LED amarelo conectado ao pino 8 do arduino
3 int ledVERD = 9; // Pino do LED verde conectado ao pino 9 do arduino
4 int ledVERM = 10; // Pino do LED vermelho conectado ao pino 10 do arduino
5
6 byte dadosByte = 0; // Variável que armazena os dados recebidos do matlab
7
8 void setup()
9 {
10     // define os pinos do arduino conectados aos leds como saída
11     pinMode(ledA, OUTPUT);
12     pinMode(ledVERD, OUTPUT);
13     pinMode(ledVERM, OUTPUT);
14
15
16     // Escreve 0 nos pinos conectados aos LED para garantir que ele iniciem apagados
17     digitalWrite(ledA, LOW);
18     digitalWrite(ledVERD, LOW);
19     digitalWrite(ledVERM, LOW);
20
21
22     //Configura a porta serial para a taxa de transmissão de 9600 bits/s
23     Serial.begin (9600);
24
25     // Envia o string "SIM" para o matlab para indicar que a porta serial já foi inicializada
26     //Ao receber "SIM", o matlab sabe que já pode enviar e receber dados do arduino
27     Serial.println ("SIM");
28 }
```

Nas linhas 1, 2 e 3 há a declaração das variáveis que representam os pinos do Arduino aos quais estão conectados os LEDs. Na linha 4 é declarada a variável que armazenará os dados recebidos pelo Arduino através da comunicação serial. Entre as linhas 11 a 19, os pinos do Arduino conectados aos LEDs são definidos como saída e colocados em nível lógico zero.

Na linha 23 é chamada a **Serial.begin()** e é passada para ela o parâmetro 9600, dessa forma, a comunicação serial do Arduino é inicializada com **BaudRate** de 9600 bits/segundo. Após a completa inicialização da comunicação serial, o Arduino irá chamar a função **Serial.println()** que enviará para o Matlab o string 'SIM'. Como mostrado anteriormente, o Matlab só enviará dados ao Arduino após o recebimento desse string. Aqui é usado a função **Serial.println()** em vez da função **Serial.print()** porque a função de leitura de string **fscanf()** executada pelo Matlab, sempre aguarda que o string recebido seja finalizado pelo caractere '\n'. A função **Serial.println()**, sempre envia o caractere '\n' após o envio de um string, o que a função **Serial.print()** não faz.

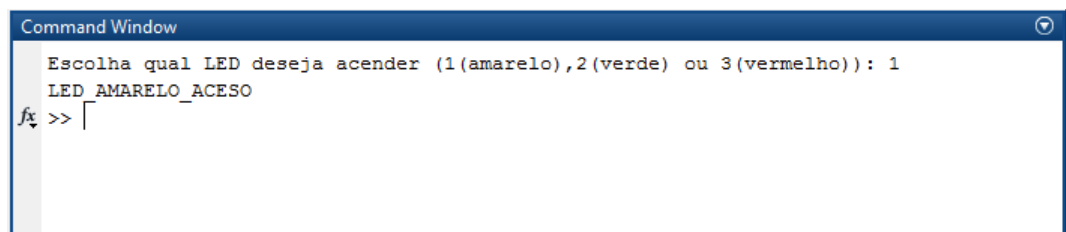
A função **void loop()** do programa executado pelo Arduino é mostrado abaixo. Todas as instruções de leitura de dados e posteriormente, envio de dados e acionamento dos LEDs, só serão executadas se na linha de código 33, a função **Serial.available()** retornar um número maior que zero indicando que algum dado foi recebido. Após a confirmação de recebimento do dado, na linha 35 é chamada a função **Serial.read()** que lê o dado recebido e o salva na variável **dadosByte**. O dado recebido é um valor entre 1 e 3 que indica qual

LED deverá ser aceso. Após a leitura do dado, o conteúdo da variável **dadosByte** é testado através de uma estrutura de seleção **switch-case**. Caso o conteúdo de **dadosByte** seja 1, o pino do Arduino conectado ao LED amarelo será colocado em nível alto acendendo o LED amarelo, os demais LEDs serão apagados e o string "**LED_AMARELO_ACESO**" será enviado para o Matlab que o exibirá. Essa mesma sequência de instruções ocorre se o conteúdo de **dadosByte** for 2 ou 3, mudando apenas qual LED será aceso e o string enviado para o Matlab. Se o conteúdo de **dadosByte** for diferente de 1, 2 ou 3, o nenhum LED será aceso e será enviado ao Matlab o string "**ERRO!**";

```
30 void loop()
31 {
32
33     if (Serial.available() > 0) // Condição válida quando o arduino recebe algum byte
34     {
35         dadosByte = Serial.read(); // Lê os dados recebidos
36         switch (dadosByte) // Switch que define qual LED irá acender
37         {
38             // Ao acender algum LED, o arduino envia ao matlab um
39             // string confirmando que o LED selecionado foi aceso
40
41             case 1:
42                 digitalWrite(ledA, HIGH);
43                 digitalWrite(ledVERD, LOW);
44                 digitalWrite(ledVERM, LOW);
45                 Serial.println ("LED_AMARELO_ACESO");
46                 break;
47
48             case 2:
49                 digitalWrite(ledA, LOW);
50                 digitalWrite(ledVERD, HIGH);
51                 digitalWrite(ledVERM, LOW);
52                 Serial.println ("LED_VERDE_ACESO");
53                 break;
54
55             case 3:
56                 digitalWrite(ledA, LOW);
57                 digitalWrite(ledVERD, LOW);
58                 digitalWrite(ledVERM, HIGH);
59                 Serial.println ("LED_VERMELHO_ACESO");
60                 break;
61
62             default:
63                 Serial.println ("ERRO!");
64                 break;
65         }
66     }
67 }
```

4 Resultados

A Figura 2 mostra a "Command Window", janela de comandos do Matlab, após a execução do script descrito na seção anterior. Observe que o LED escolhido para ser aceso foi o LED amarelo. A exibição da mensagem "LED_AMARELO_ACESO" significa que o Arduino conseguiu acender com sucesso o LED amarelo como mostrado na Figura 3.



```
Command Window
Escolha qual LED deseja acender (1(amarelo),2(verde) ou 3(vermelho)): 1
LED_AMARELO_ACESO
fx >> |
```

Figura 2 – Janela de comandos do Matlab após a execução do script e seleção do LED amarelo

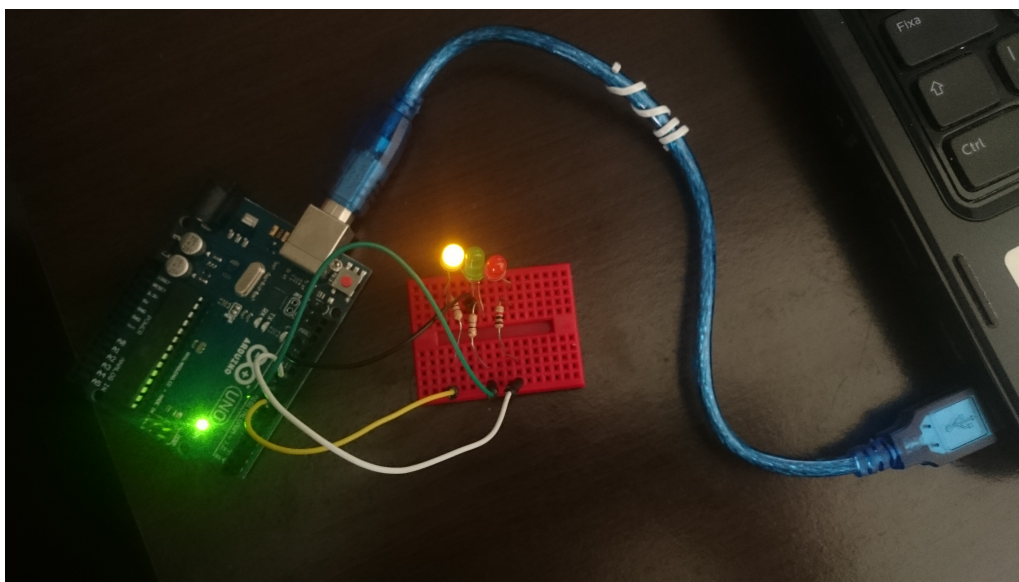


Figura 3 – LED amarelo aceso após a execução do script do Matlab e seleção do LED amarelo

Referências

Documentação do Arduino sobre as funções relacionadas a comunicação serial:

<<https://www.arduino.cc/en/Reference/Serial>>

Documentação do Matlab sobre a função serial():

<[http://www.mathworks.com/help/matlab/ref/serial.html?searchHighlight=serial\(\)&s_tid=doc_srchttitle](http://www.mathworks.com/help/matlab/ref/serial.html?searchHighlight=serial()&s_tid=doc_srchttitle)>

Documentação do Matlab sobre a função fopen(serial):

<<http://www.mathworks.com/help/matlab/ref/serial.fopen.html>>

Documentação do Matlab sobre a função fclose(serial):

<<http://www.mathworks.com/help/matlab/ref/serial.fclose.html>>

Documentação do Matlab sobre a função fread(serial):

<<http://www.mathworks.com/help/matlab/ref/serial.fread.html>>

Documentação do Matlab sobre a função fwrite(serial):

<<http://www.mathworks.com/help/matlab/ref/serial.fwrite.html>>

Documentação do Matlab sobre a função fscanf(serial):

<<http://www.mathworks.com/help/matlab/ref/serial.fscanf.html>>

Documentação do Matlab sobre a função fprintf(serial):

<<http://www.mathworks.com/help/matlab/ref/serial.fprintf.html>>

Documentação do Matlab sobre o atributo BytesAvailable:

<http://www.mathworks.com/help/matlab/matlab_external/bytesavailable.html?searchHighlight=bytesavailable&s_tid=doc_srchttitle>