Placa Arduíno MEGA 2560 – IDE e comunicação serial

Nesta nota iremos tratar especificamente de tópicos específicos à ferramenta de Monitor Serial da IDE do Arduíno e da comunicação serial com protocolo I2C da placa Arduíno MEGA 2560. A nota dá prosseguimento ao que foi tratado nas notas 1, 2 e 3.

Ambiente da IDE

A Figura 1 apresenta a tela principal da IDE.

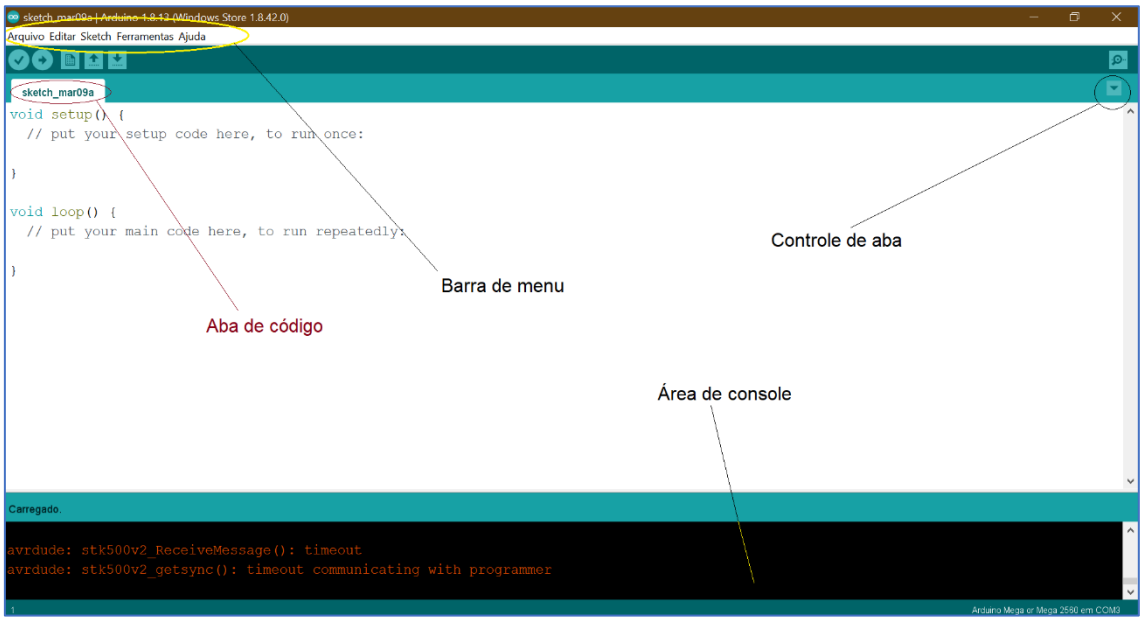

Figura 1 – Tela inicial da IDE do Arduíno

A barra de menu é constituída por: *Arquivo*, *Editar*, *Sketch*, *Ferramentas* e *Ajuda*. Vamos nos concentrar somente em alguns elementos da barra de menu. No menu Ferramentas, conforme apresenta a Figura 2, nesta nota focaremos no Monitor Serial.

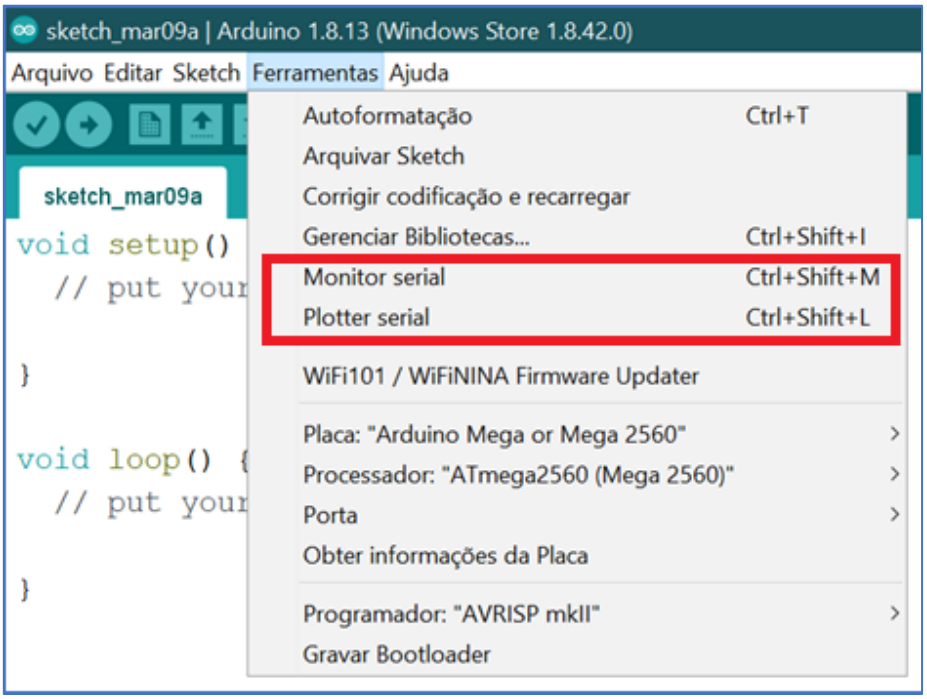

Figura 2 – Menu Ferramentas

- *Ferramentas>Monitor serial* – abre uma janela de monitor serial e inicia a troca de dados com qualquer placa conectada na porta selecionada; normalmente reinicializa a placa se ela suportar a reinicialização pela porta serial – é necessário que a placa esteja alimentada e conectada.

Merece ser comentada a ferramenta Plotter Serial, que foge ao escopo desta nota e será aprofundada oportunamente, também relacionada com comunicação serial na placa Arduíno MEGA 2560

- *Ferramentas>Plotter serial* – abre uma janela que mostra graficamente o comportamento da taxa da comunicação serial das portas analógicas e digitais com dispositivos de I/O.

Após realizada a conexão da placa na porta adequada automaticamente o programador é executado (código de inicialização da placa). É possível ver o LED de TX piscando. A IDE apresentará uma mensagem que o *sketch* foi carregado ou que houve um erro.

Monitor serial

Ao selecionar no menu *Ferramentas>Monitor serial* a janela apresentada na Figura 3 será aberta.

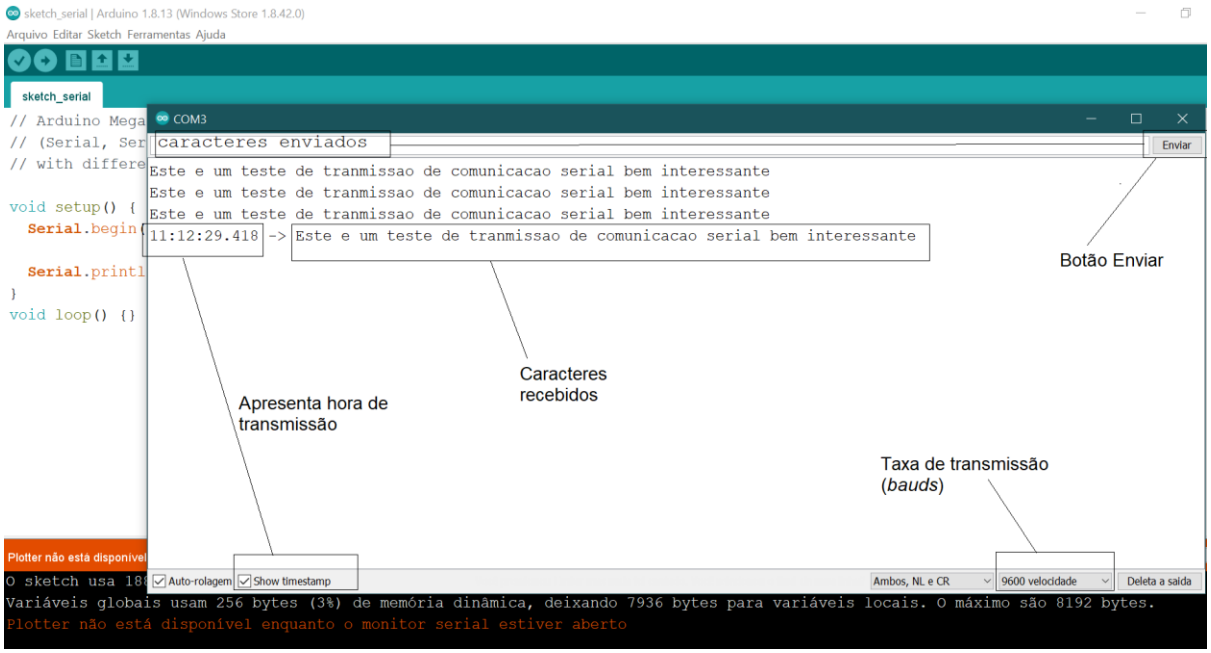

Figura 3 – Janela do Monitor Serial

O Monitor Serial simula a comunicação serial entre um dispositivo de I/O e a placa Arduíno, permitindo realizar testes antes da prototipação física. Permite simular o envio de dados para a placa do Arduíno e observar os dados transmitidos por ela, recebidos pelo dispositivo de I/O.

Lembrando, conforme tratado na nota 1, que para comunicação serial a placa do Arduíno utiliza os seguintes pinos, aos pares,

- 0 (RX), 1 (TX) – *Serial*;
- 19 (RX), 18 (TX) - *Serial1*;
- 17 (RX), 16 (TX) - *Serial2*; e
- 15 (RX), 14 (TX) - *Serial3*.

Embora a placa Arduíno MEGA 2560 disponha dessa pinagem, o Monitor Serial permite simulação somente com a porta *Serial*. Os códigos podem ser elaborados com os objetos *Serial1*, *Serial2* e *Serial3*, dependendo da pinagem utilizada no circuito a ser implementado, mas esses objetos não são simulados no Monitor Serial.

Para a comunicação serial utilizando os protocolos I2C e SPI (ambos detalhados na nota 1) são utilizados pinos específicos.

Se a comunicação serial se basear no protocolo I2C, são usados os pinos

- Pino 20 (SDA); e
- Pino 21 (SCL).

Se a comunicação serial se basear no protocolo SPI, são usados os pinos

- pino 50 (MISO);
- pino 51 (MOSI);

- pino 52 (SCK); e
- pino 53 (SS).

Em relação à operação do Monitor Serial, para enviar dados à placa deve-se digitar o texto e clicar no botão *Enviar* (ou teclar *enter*). Escolher a taxa de dados no objeto *drop-down* da parte inferior da janela. As velocidades apresentadas são estabelecidas em *bauds*.

A taxa de transmissão (em *bauds*) deve ser a mesma que a taxa passada para o *sketch* na função *Serial.begin()* (As funções de comunicação serial do Arduíno podem ser encontradas em https://www.arduino.cc/reference/en/language/functions/communication/serial/).

É bom esclarecer que *baud* é a unidade que mede o número de mudanças de estado no canal de comunicações por segundo. Se a comunicação admite a informação representada por um sinal que pode apresentar N estados diferentes em um segundo, a taxa de transmissão medida em *bauds* será N *bauds*, ou seja, N estados/seg.

Por exemplo, suponha que um sinal possa assumir 4 estados diferentes {s_0, s_1, s_2, s_3}, tal que, s_0 representa 0, s_1 representa 1, s_2 representa 2 e s_3 representa 3. Admitindo que durante um intervalo de 1 segundo podem ser observados 1024 estados, a taxa será 1024 *bauds*.

A taxa de transmissão não é necessariamente igual à taxa em bps, ou bits/seg. A taxa em bps indica o número de bits, 0s e 1s, observados em uma janela de 1 segundo. No exemplo acima, admitindo que cada estado seja representado por dois bits, por exemplo, {s_0=00, s_1=01, s_2=10, s_3=11} a taxa de transmissão seria dada por 2 bits/estado x 1024 estados/seg = 2048 bps.

Função *Serial.begin()*

A função *Serial.begin()* configura o *sketch* para operar em uma taxa (em *bauds*) para comunicação serial. Se estiver usando dispositivos conectados aos pinos de comunicação serial da placa deve ser usada a taxa compatível com o dispositivo. Opera com um argumento opcional adicional, que representa as configurações de tamanho da palavra, paridade usada e bits de *stop*. O padrão de configuração de porta é SERIAL_8N1 – sem uso do argumento opcional – 8 bits, sem paridade, 1 bit *stop*.

Para compreensão melhor desse ponto é conveniente tratar da comunicação assíncrona, que utiliza o esquema apresentado na Figura 4.

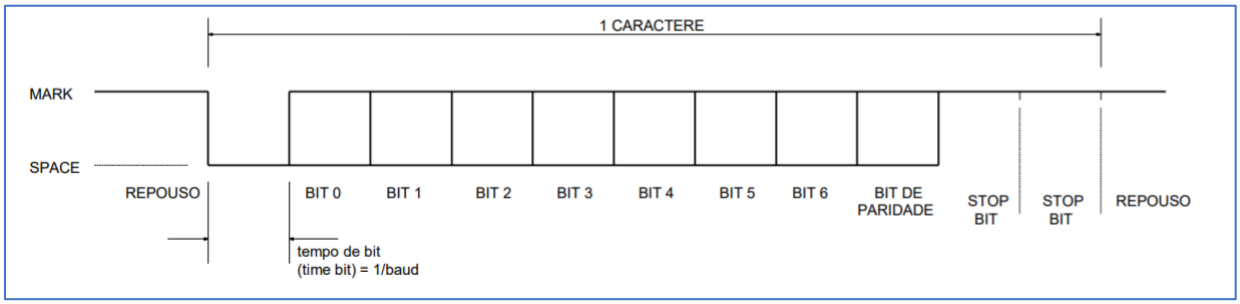

Figura 4 – Protocolo de comunicação serial assíncrona

Na figura, está apresentada uma sequência de 10 bits em uma transmissão. Os bits b_0 a b_6 (7 bits) representam a informação propriamente dita.

A figura apresenta um esquema que utiliza um bit de paridade. O bit é inserido pelo transmissor de modo a assegurar que o número de bits 1 de informações mais o bit de paridade seja par (se o código usa paridade par) ou impar (se o código usa paridade ímpar).

Os bits de paridade são inseridos após cada palavra binária transmitida serialmente, ou seja, é inserido após os bits de informação, a fim de possibilitar a detecção e correção de erros no receptor.

Os erros são causados por imperfeições do canal de comunicações. Dependendo do número de bits utilizados é possível somente detectar erros ou detectar e corrigi-los na palavra recebida, alterada no processo de transmissão sem requerer novo envio dos bits.

Se não for possível corrigir a palavra enviada, após detectar o erro o receptor solicita ao transmissor a retransmissão somente daquela palavra.

No caso da placa do Arduíno, se for utilizada a configuração de transmissão SERIAL_8O1 (8 bits, paridade ímpar, 1 bit de Stop), por exemplo, a palavra de 8 bits 0x3A (em binário 0011 1010), o bit de paridade deverá ser 1 (totalizando 5 bits 1).

Os bits de start e stop indicam respectivamente, em uma transmissão assíncrona, a marcação de início e final de um novo trem de bits de informação transmitidos. Na figura, a marcação de repouso marca o início e os dois bits de *stop* marcam o final.

Voltando ao exemplo de transmissão com configuração SERIAL_8O1 (8 bits, paridade ímpar, 1 bit de Stop) da palavra 0x3A (em binário 0011 1010), o bit de paridade deverá ser 1 e o bit de Stop severa ser 1, logo o trem de bits transmitidos deverá ser 0011 1010 1 1.

A sintaxe da função *Serial.begin()* é

Serial.begin(taxa)
Serial.begin(taxa,configuração)

Utiliza os parâmetros *taxa* e *configuração*. O argumento *taxa* representa a taxa de transmissão utilizada, em *bauds*. As taxas padrões são:

Tabela 1 – Taxas (em bauds) de transmissão serial utilizadas

50	75	110	134	150	200	300	600	1200	1800
2.4 k	9.6k	19.2k	38.4k	57.6k	115.2k	230.4k	460.8k	500k	576k
921.6k	1M	1.152M	1.5M	2M	2.5M	3M	3.5M	4M	

Permite dados do tipo *long*. Dados do tipo *long* armazenam 32 bits, representando os valores -2^{31} a $+2^{31}$.

O argumento *configuração* estabelece a configuração da porta serial que deve ser usada na transmissão.

O padrão da configuração é dado por SERIAL_8N1: 8 bits – Sem paridade - 1 bit Stop. As demais configurações admitidas (consultada em https://nvsl.github.io/PiDuino_Library/function/2000/02/02/Serial-begin-config.html).

A Tabela 2 apresenta as configurações de comunicação serial da placa Arduíno Mega 2560.

Tabela 2 – Configuração da comunicação serial do Arduíno

Configuração	Tamanho de dados	Paridade	Bits de Stop
SERIAL_5N1	5 bits	Nenhum	1 bit
SERIAL_6N1	6 bits	Nenhum	1 bit
SERIAL_7N1	7 bits	Nenhum	1 bit
SERIAL_8N1	8 bits	Nenhum	1 bit
SERIAL_5N2	5 bits	Nenhum	2 bits
SERIAL_6N2	6 bits	Nenhum	2 bits
SERIAL_7N2	7 bits	Nenhum	2 bits
SERIAL_8N2	8 bits	Nenhum	2 bits
SERIAL_5E1	5 bits	Par	1 bit
SERIAL_6E1	6 bits	Par	1 bit
SERIAL_7E1	7 bits	Par	1 bit
SERIAL_8E1	8 bits	Par	1 bit

Configuração	Tamanho de dados	Paridade	Bits de Stop
SERIAL_5E2	5 bits	Par	2 bits
SERIAL_6E2	6 bits	Par	2 bits
SERIAL_7E2	7 bits	Par	2 bits
SERIAL_8E2	8 bits	Par	2 bits
SERIAL_5O1	5 bits	Impar	1 bit
SERIAL_6O1	6 bits	Impar	1 bit
SERIAL_7O1	7 bits	Impar	1 bit
SERIAL_8O1	8 bits	Impar	1 bit
SERIAL_5O2	5 bits	Impar	2 bits
SERIAL_6O2	6 bits	Impar	2 bits
SERIAL_7O2	7 bits	Impar	2 bits
SERIAL_8O2	8 bits	Impar	2 bits

A função *Serial.begin()* não retorna valor.

Outras funções de comunicação serial na IDE do Arduíno

Algumas outras funções são importantes para explorar os recursos de comunicação serial do Arduíno. Estão descritas de forma sucinta na Tabela 3. Acessar https://www.arduino.cc/reference/pt/language/functions/communication/serial/ para consultar detalhes e funções não apresentadas na Tabela 3.

Algumas observações cabem nesse ponto, a respeito da linguagem:

- Tipo *size_t*: tipo de dados próprio para representar o tamanho de qualquer objeto em bytes;
- Objeto String: instância de uma classe String que permite construir sequências de caracteres com dados de vários tipos;
- Tipo string: cadeia de caracteres declarada na forma *char idt[dim]* – da mesma forma da declaração na linguagem C;
- Observar que as *strings* em linguagem C são vetores de caracteres, em que o identificador (*idt*) segue a regra geral para definir identificadores de variáveis e *dim* é a dimensão do vetor. Lembre-se que todo vetor é finalizado com \0, ou seja, o último caractere é reservado a \0.
- É importante observar os tipos definidos tanto para os argumentos das funções quanto para os valores que elas retornam.

Tabela 3 – Funções de comunicação serial da IDE do Arduíno

Sintaxe	**Descrição**	**Parâmetros**	**Retorno (*tipo*/valor)**
Inicialização e término			
Serial.begin(taxa) *Serial.begin(taxa,configuração)*	Configura o *sketch* para operar em uma taxa (em bauds) para comunicação serial	*taxa*: taxa de transmissão (*bauds*) *configuração*: Ver Tabela 2	Não retorna valor
Serial.end()	Desativa a comunicação serial, permitindo os pinos RX e TX serem usados novamente como entradas e saídas digitais.	Nenhum	Não retorna valor
Leitura de dados			
Serial.read()	Lê dados recebidos na porta.	Nenhum	*int* / quantidade de bytes recebidos; -1 se não há dados
Serial.readBytes(buffer,lenght)	Lê caracteres recebidos e carrega em *buffer*.	*buffer*: *char[dim]* ou *byte[dim]* *lenght*: *int* número de bytes lidos	*size_t* / número de bytes armazenados no *buffer*
Serial.readString()	Lê caracteres recebidos e carrega em *buffer*.	Nenhum	*String* / caracteres lidos
Escrita de dados			
Serial.print(val) *Serial.print(val, formato)*	Imprime caracteres como texto ASCII	*val*: valor impresso de qualquer tipo *formato*: NO FORMAT, DEC, HEX, OCT, BIN	*size_t* / número de bytes escritos
Serial.println(val) *Serial.println(val, formato)*	Imprime caracteres como texto ASCII seguido pelo caractere de retorno (ASCII 13, ou '\r') e pelo caractere de nova linha (ASCII 10, ou '\n').	*val*: valor impresso de qualquer tipo *formato*: NO FORMAT, DEC, HEX, OCT, BIN	

Sintaxe	**Descrição**	**Parâmetros**	**Retorno (*tipo*/valor)**
Serial.write(val) *Serial.write(str)* *Serial.write(buf, len)*	Escreve dados binários na porta serial.	*val: byte* *str*: *char[ ]* *buf*: *byte[ ]* *len*: *int*	*size_t* / número de bytes escritos
Estrutura de controle			
If(Serial)	Indica se a porta está pronta	Nenhum	*True*, se a porta estiver pronta. *False*, caso contrário
Serial.available()	Retorna o número de bytes, tipo *int*, disponíveis para leitura da porta	Nenhum	*int*, Número de bytes disponíveis
Serial.find(target) *Serial.find(target, length)*	Lê dados do *buffer* de recebimento até a *string* especificada ser encontrada	*target*: *char[ ]* *length*: int, número de dados lidos	*True*, se encontrar. *False*, caso contrário.
Serial.flush()	Espera a transmissão de dados enviados terminar.	Nenhum	Não retorna valor
Serial.setTimeout(tempo)	Configura o número máximo de milissegundos a se esperar por dados seriais. Valor padrão: 1000 mseg	*tempo*: limite (mseg)	Não retorna valor

Aplicação

O primeiro código objetiva discutir algumas particularidades das funções de comunicação serial da placa, além de apontar alguns aspectos da linguagem da IDE.

Código 1)

A placa lê de um dispositivo serial um bloco de 200 caracteres. Cada bloco tem a seguinte estrutura, em sequência:

nome: <nome> código: <código> idade: <idade> descrição: <descrição> balcão: <balcão>, conforme apresentado na

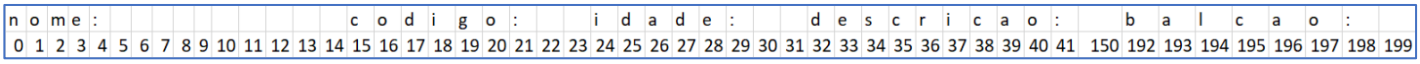

Figura 5 – Exemplo: estrutura de dados enviados por dispositivo de comunicação serial

A estrutura possui os campos com as dimensões

- <nome>: 10 caracteres;
- <código>: 2 caracteres;
- <idade>: 2 caracteres;
- <descrição>: 150 caracteres;
- <balcão>: 1 caractere;

totalizando 200 caracteres com os rótulos.

O programa devolve o nome e o código da pessoa do bloco de 200 caracteres ao dispositivo serial. Em seguida, apresenta o balcão em que será atendida.

Código 1 – Comunicação serial: leitura e filtragem de dados

```
// O programa ilustra a comunicação serial usando o Arduíno
// Integra a nota 4 produzida a respeito da utilização da placa Arduíno MEGA 2560

String bloco; //Declara a variável bloco para receber os caracteres da estrutura
char nome[10]; //Declara o nome do usuario
char codigo[2]; //Declara o código do usuário
char balcao; //Declara o balcão de atendimento

void setup() {
  Serial.begin(9600); //Define a taxa de 9600 bauds com a serial - pinos 0 (RX) 1 (TX)
}

void loop() {
  int i;
  /*Envia dados de uma porta serial - pinos 0 e 1 da placa:*/
  if(Serial) // Verifica se a porta está pronta para receber os dados
    bloco=Serial.readString(); //Carrega a estrutura lida na variável bloco
  /*Identifica os dados de nome no bloco lido*/
  for (i=0;i<=9;i++)
    nome[i]=bloco[i+5];
  /*Identifica os dados de código no bloco lido*/
    codigo[0]=bloco[22];
    codigo[1]=bloco[23];
```

```
  /*Identifica o balcão de atendimento*/
    balcao=bloco[199];
  /*Apresentação do resultado*/
    for (i=0;i<=9;i++)
      Serial.print(nome[i]);
    Serial.print(codigo[0]);
    Serial.print(codigo[1]);
    Serial.print(balcao);
}
```

A respeito do código, seguem alguns comentários:

- O código utiliza objeto String e tipos string.
- São utilizadas algumas das funções de comunicação serial descritas na Tabela 3. É importante respeitar a compatibilidade de tipos.
- Utiliza-se o objeto Serial, que significa a utilização dos pinos 0 e 1 da placa Arduíno MEGA 2560. Na implementação do circuito (que não é o foco na presente nota).
- A mesma porta de entrada serial de dados é a porta de saída. Sendo que a entrada é dada pelo pino 0 (RX) e a saída pelo pino 1 (TX).
- Não foi utilizado caractere acentuado. Lembre-se que esse aspecto é relevante, uma vez que a função *Serial.print(val)* imprime caracteres como texto ASCII;

Para testar o código pode-se utilizar a ferramenta Monitor Serial. O procedimento para uso do monitor é:

1. Na aba *Sketch* da IDE, selecionar a opção Carregar;
2. Na aba de Ferramentas da IDE, selecionar a opção Monitor Serial;

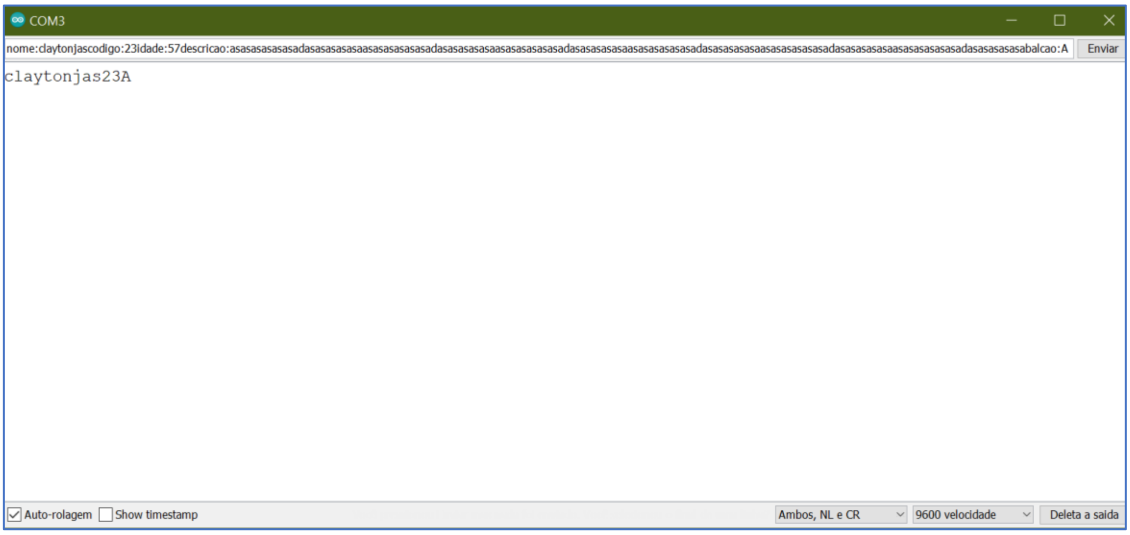

Figura 6 – Simulação da execução do código com a ferramenta Monitor Serial

3. Certificar-se que a taxa de transmissão/recepção é a mesma da configuração da placa, definida com a função *Serial.begin()*;

4. Inseri no campo de texto do Monitor Serial a mensagem a ser enviada – conforme a estrutura definida no enunciado (ver a Figura 6) ;
5. Observar a sequência de caracteres recebidos, com o tratamento realizado pelo código carregado na placa do Arduíno (ver a Figura 6).

Componentes I2C

Para aprofundar a compreensão da comunicação serial com a placa do Arduíno, vamos discutir também um circuito que utilize protocolo I2C, muito comum para diversos tipos de sensores do mercado.

Nesta nota será simulada a comunicação, no entanto não serão apresentados os esquemas elétricos de utilização dos componentes com os quais será estabelecida a comunicação serial do componente com a placa Arduíno MEGA 2560. Não serão apresentados também as prototipações em placa de *protoboard*. O foco da nota é utilizar os recursos da ferramenta Monitor Serial para testar os códigos da linguagem da IDE e os conceitos de comunicação aqui discutidos.

Utilizaremos para fazer as simulações um módulo serial I2C para display LCD Arduíno. A Figura 7 apresenta a visão do componente segundo quatro perspectivas.

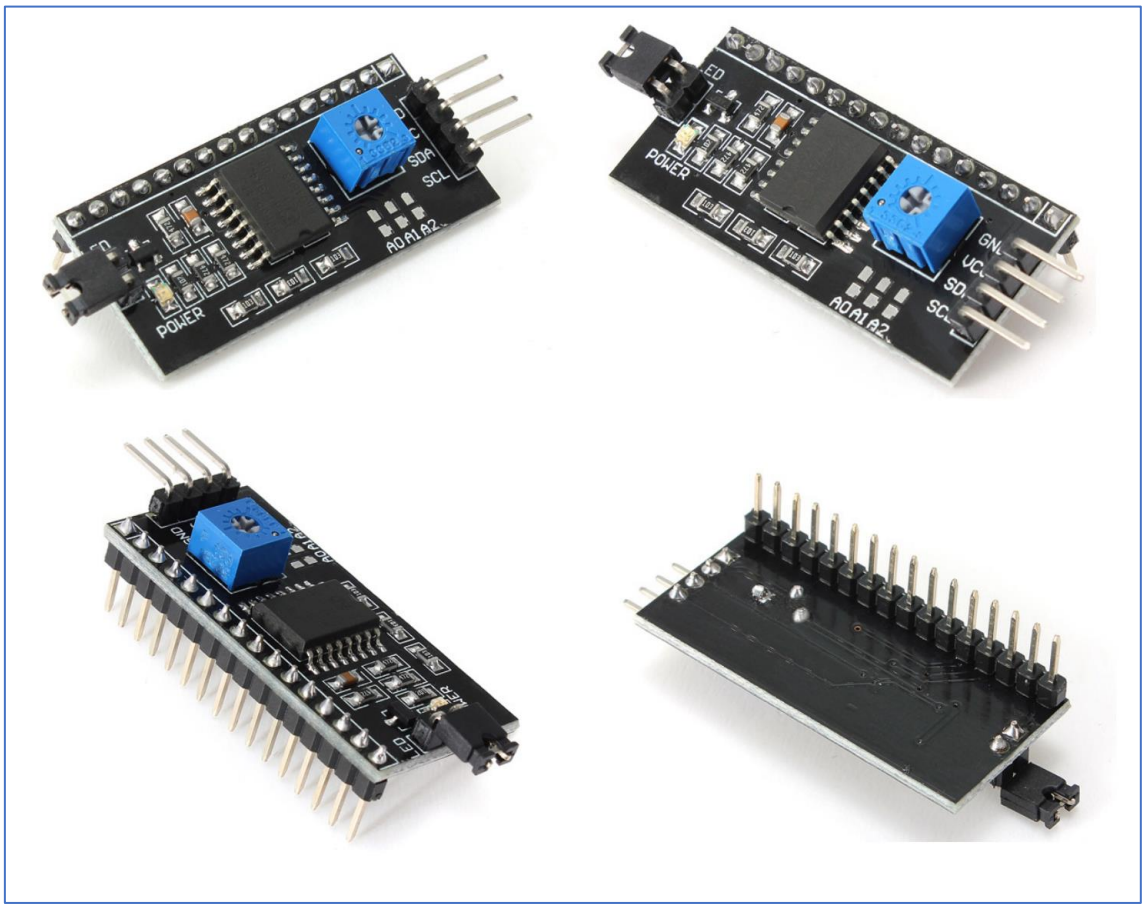

Figura 7 – Perspectivas de uma interface serial para LCD com Arduíno

A interface serial simplifica a ligação do painel LCD com a placa do Arduíno, pois reduz o número de ligações com a pinagem da comunicação. O painel LCD demanda um barramento paralelo que ocuparia vários pinos de dados da placa Arduíno, utilizando as saídas digitais. Utilizando a interface serial, entre o Arduíno e a interface somente uma linha de dados é utilizada. O barramento paralelo de N-bits é estabelecido entre a interface e o painel LCD.

A Figura 8 apresenta o diagrama em blocos das ligações.

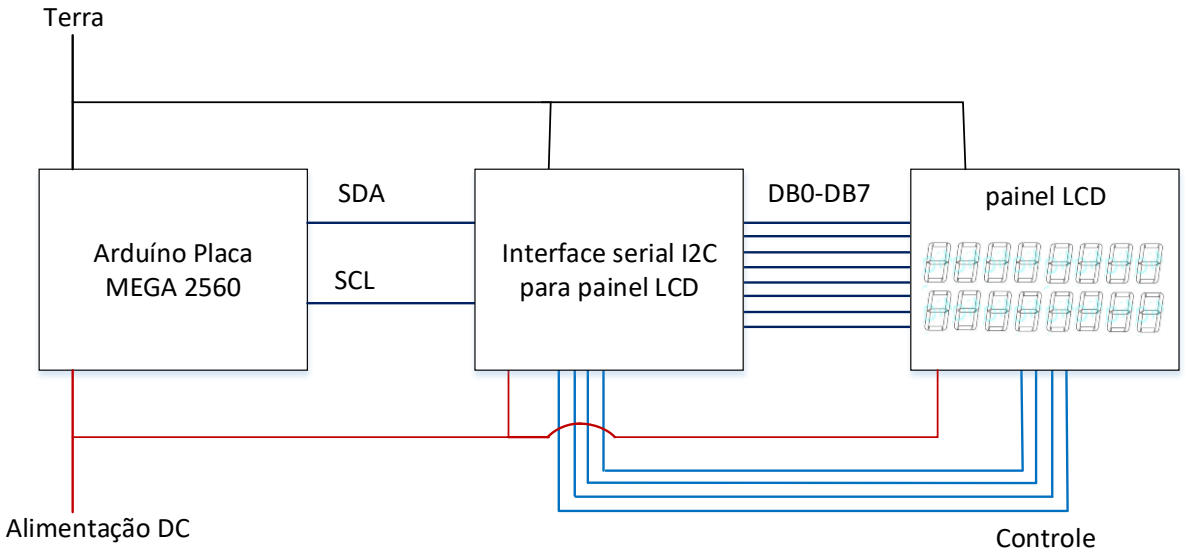

Figura 8 – Diagrama em blocos de ligação placa Arduíno -interface serial I2C-painel LCD

O diagrama em blocos das ligações considera uma interface I2C que pode ser acessada nos endereços I2C 0x20-0x27. O painel LCD considerado é um 16x2, ou seja, possui 16 caracteres em 2 linhas, utilizando uma pinagem de dados de 8-bits.

Toda a sinalização de controle do painel é gerada pela interface.

Biblioteca *Wire.h*

A biblioteca *Wire.h* permite conectar dispositivos de comunicação serial que utilizem protocolo I2C com as placas Arduíno. Como já tratado anteriormente, os pinos 20 (SDA) e 21 (SCL) da placa Arduíno MEGA 2560 destinam-se a essa comunicação.

Existem versões de endereços I2C de 7 e 8 bits, 7 bits para identificar o dispositivo e o oitavo bit para determinar se ele está sendo escrito ou lido. A biblioteca *Wire.h* usa endereços de 7 bits.

A Tabela 4 apresenta as funções da biblioteca *Wire.h*.

Tabela 4 – Funções da biblioteca Wire.h para comunicação I2C

Sintaxe	**Descrição**	**Parâmetros**	**Retorno (*tipo*/valor)**
Wire.begin(taxa) *Wire.begin(endereço)*	Inicia a biblioteca e configura o barramento I2C como *master* ou *slave*.	*endereço*: 7-bits de escravo; se não especificado o componente é mestre.	Não retorna valor
Wire.requestFrom(endereço,quantidade) *Wire.requestFrom(endereço,quantidade,stop)*	Solicita dados do *slave*.	*endereço*: 7-bits de escravo; *quantidade*: *int*, número de bytes *stop*: booleano, *true* libera o barramento após a requisição	*byte* / Número de bytes obtidos
Wire.beginTransmission(endereço)	Inicia uma transmissão para o dispositivo escravo I2C com o endereço fornecido.	*endereço*: 7-bits de escravo	Não retorna valor
Wire.endTransmission() *Wire.endTransmission(stop)*	Encerra uma transmissão para um dispositivo escravo e transmite os bytes que foram enfileirados.	*stop*: booleano, *true* libera o barramento após a transmissão	*byte* / Status da transmissão: 0-sucesso;1,2,3 e 4 erros (Ver detalhes)
Wire.write(val) *Wire.write(string)* *Wire.write(data, len)*	Grava dados de um *slave* em resposta a um *master*; ou enfileira bytes para transmissão de um *master* para o *slave*.	*val*: um *byte* *string*: *byte[ ]* *data*: *byte[ ][ ]* *len*: int, número de bytes	*byte* / Número de bytes escritos

Sintaxe	**Descrição**	**Parâmetros**	**Retorno (*tipo*/valor)**
Wire.available()	Retorna o número de bytes disponíveis para recuperação com *read()*.Deve ser chamado em um *master* após uma chamada para *requestFrom()* ou em um escravo dentro do manipulador *onReceive()*.	Nenhum	*byte* / Número de bytes disponíveis
Wire.read()	Lê um byte que foi transmitido de um *slave* para um *master* após uma chamada para *requestFrom()* ou foi transmitido de um *master* para *slave*.	Nenhum	*byte* / O próximo byte recebido
Wire.setClock (clockFreq)	Modifica a frequência do *clock* para comunicação I2C.	*clockFreq*: 100000 ou 400000	Não retorna valor
Wire.onReceive (handler)	Registra uma função a ser chamada quando um *slave* recebe uma transmissão de um *master*.	*handler*: função a ser chamada	Não retorna valor
Wire.onRequest (manipulador)	Registra uma função a ser chamada quando um master solicita dados a um *slave*.	*handler*: função a ser chamada	Não retorna valor

Aplicação

Códigos 2 e 3)

Considerando o mesmo problema apresentado no código 1, deseja-se que a placa do Arduíno leia a estrutura de dados com 200 caracteres a partir de um dispositivo serial de comunicação conectado através dos pinos 0 e 1 da placa – componente A. O código da placa dever ler os dados enviados pelo componente A, tratar os dados recebidos e enviar somente nome, código e balcão de atendimento para um painel de LCD através de uma interface serial I2C. Considerar o esquema de blocos apresentado na Figura 8.

Código 2 – Comunicação serial com painel LCD 16x2 através de interface I2C

```
// O programa ilustra a comunicação serial usando o Arduíno
// Integra a nota 4 produzida a respeito da utilização da placa Arduíno MEGA 2560
#include <Wire.h>

String bloco; //Declara a variável bloco para receber os caracteres da estrutura
byte nome[10]; //Declara o nome do usuario
byte codigo[2]; //Declara o código do usuário
byte balcao; //Declara o balcão de atendimento

void setup() {
  Serial.begin(9600); //Define a taxa de 9600 bauds do dispositivo serial A: 0 (RX) 1 (TX)
  Wire.begin();//Inicializa a biblioteca - define a placa Arduíno como master
}

void loop() {
  int i;
  byte erro;
  /*Envia dados de uma porta serial - pinos 0 e 1 da placa:*/
  if(Serial) // Verifica se a porta conectada ao componente A está pronta para receber os
dados
    bloco=Serial.readString(); //Carrega a estrutura lida na variável bloco
  /*Identifica os dados de nome no bloco lido*/
  for (i=0;i<=9;i++)
    nome[i]=bloco[i+5];
  /*Identifica os dados de código no bloco lido*/
    codigo[0]=bloco[22];
    codigo[1]=bloco[23];
  /*Identifica o balcão de atendimento*/
    balcao=bloco[199];
  /*Apresentação do resultado*/
    Wire.beginTransmission(0x20);
    erro=Wire.endTransmission();
    if (erro==0)
      Wire.write(nome[10]);
      Wire.write(codigo[2]);
      Wire.write(balcao);
      Wire.endTransmission();
}
```

A respeito dos códigos 2 e 3, seguem alguns comentários:

- Admite-se que o painel está no endereço 0x20;

- O código se ocupa exclusivamente do envio dos dados, no entanto, conforme apresentado na Figura 8, o painel LCD requer também sinais de controle;
- Para explorar todos os sinais entre a interface I2C e o painel LCD pode-se utilizar a biblioteca *LiquidCrystal_I2C.h,* conforme o código 3 detalha.
- O código se encarrega de verificar se há erro na transmissão, com o uso da função *Wire.endTransmission()*, porém não apresenta um caminho alternativo caso seja observado qualquer um dos erros possíveis;
- Os códigos 2 e 3 foram compilados corretamente, porém não puderam ser testados fisicamente pela falta dos componentes para construir o circuito, o que será realizado posteriormente.

Código 3 – Comunicação serial com painel LCD 16x2 através de interface I2C

```
// O programa ilustra a comunicação serial usando o Arduíno
// Integra a nota 4 produzida a respeito da utilização da placa Arduíno MEGA 2560
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

String bloco; //Declara a variável bloco para receber os caracteres da estrutura
char nome[10]; //Declara o nome do usuario
char codigo[2]; //Declara o código do usuário
char balcao; //Declara o balcão de atendimento
LiquidCrystal_I2C lcd(0x20,16,2); //Cria um lcd 16x2 no endreço 0x20

void setup() {
  Serial.begin(9600); //Define a taxa de 9600 bauds dos pinos de comunicação serial 0 (RX)
1 (TX)
  lcd.init(); //Inicializando o lcd
  lcd.backlight(); //Ligando o backlight do lcd
}

void loop() {
  int i;
  /*Envia dados de uma porta serial - pinos 0 e 1 da placa:*/
  if(Serial) // Verifica se a porta conectada ao componente A está pronta para receber os
dados
    bloco=Serial.readString(); //Carrega a estrutura lida na variável bloco
  /*Identifica os dados de nome no bloco lido*/
  for (i=0;i<=9;i++)
    nome[i]=bloco[i+5];
  /*Identifica os dados de código no bloco lido*/
    codigo[0]=bloco[22];
    codigo[1]=bloco[23];
  /*Identifica o balcão de atendimento*/
    balcao=bloco[199];
  /*Apresentação do resultado*/
    lcd.print(nome);
    lcd.print(codigo);
    lcd.print(balcao);
}
```