Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi for .NET

Controle de Hardware com Delphi

Usando um LCD no Delphi

Resumo DevMan

Durante a época do MS-DOS e Windows 3.1, 95 e 98 tínhamos como desenvolver facilmente programas para envio de informações ao hardware utilizando diversas linguagens. Uma delas era o próprio Delphi em sua primeira versão. A utilização da porta paralela para interfacear hardware era comum, porém, hoje, nos deparamos com um grande problema: a dificuldade em se enviar comandos para a porta paralela usando programas de modo usuário.

Nesse artigo veremos como fazer uso do Delphi e do componente LPTPort para criar um exemplo que manipula um display de LCD.

Nesse artigo veremos

- Breve histórico de acesso ao hardware em diferentes versões do Windows;
- Acesso ao hardware através de funções em Assembly;
- Acesso ao hardware através de dlls como o InpOut32.dll;
- Detalhes sobre LCDs (Arquitetura e Funcionamento);
- Construção de uma classe de controle de LCDs (THD44780);
- Projeto de um programa de Controle de LCDs.

Qual a finalidade

- Controle e manipulação de LCDs do tipo HD44780 pela porta paralela.

Quais situações utilizam esses recursos?

- CaseModding, hobbistas de eletrônica e projetos de interfaceamento de dispositivos eletrônicos.

Thiago Batista Limeira

tblimeira@gmail.com

É engenheiro da computação e trabalha com Delphi há mais de 4 anos, atualmente desenvolve programas e interfaces homem-máquina que envolvam comunicação de dados. Pode ser contatado pelo endereço tblimeira@gmail.com.

Engenheiros, *hobbistas* e outros interessados em utilizar o computador para manipular *hardware* externo, durante muito tempo utilizaram o *Turbo Pascal* e o *Delphi 1*, pois estas ferramentas possuíam uma combinação de poder e velocidade de compilação, facilitando o interfaceamento de circuitos e outros dispositivos eletrônicos. Esta flexibilidade e poder de programação eram ótimos em *DOS*, também no *Windows 3.1 e Windows 95* com o *Delphi 1*, que possuía comandos de acesso as portas do PC. Mas a partir do *Delphi 2* em diante, juntamente com o *Windows NT*, a manipulação direta do *hardware* não é mais permitida e devido a isto o acesso nativo ao *hardware* pelas linguagens

visuais modernas foi retirada.

Neste artigo faremos uma revisão dos métodos mais utilizados para acesso ao *hardware* através do *Delphi*, e mostrar todas as formas utilizadas de acesso ao *hardware* através da evolução do *Windows* e do *Delphi*. E para demonstrar as capacidades de controle do *Delphi* controlaremos um *LCD* de *16x2* pela porta paralela (**Figura 1**) utilizando o componente *LPTPort*.

A saga do acesso ao hardware

No final dos anos 80, caso alguém desejasse acender *LEDs* através do computador teria de ligá-los a um cabo de impressora, e escrever um programa que alterasse o valor armazenado em um endereço específico de memória (no caso o endereço da porta paralela). Em *Turbo Pascal* era possível utilizar o comando *Port* a seguir:

```
Port[Endereço] := Valor;
```

Este comando funcionava sem dificuldades nas primeiras versões do *Windows e Delphi 1*, já que o *Delphi 1* era *16 bits* e até meados de 1995 ainda possuía um comando *Port* de *8 bits* e *PortW* de *16 bits*. Então o *Windows* ficou "melhor". Acesso direto a localizações específicas se tornou uma má idéia, e a partir do *Delphi 2* o comando foi retirado. Apesar de não ser mais comercializado, o *Delphi 1* acompanha versões subseqüentes do *Delphi*. Em sua primeira versão é possível produzir programas que rodam nas versões *3.1, 95* e *98* do *Windows*, porém nada de *Windows NT 4.0*.

A partir da versão 2 em diante, ainda era possível acessar o *hardware* do computador de forma direta através de instruções em *assembly* como na **Listagem 1**.

Nota: Essas funções, *WritePort* e *ReadPort*, funcionavam apenas nas versões do Windows 95 e 98.

A título de curiosidade, veja que o cabeçalho das funções é Delphi, mas o corpo em si é todo *Assembly*. Inclusive diversas funções que utilizamos no dia-a-dia são desenvolvidas em *Assembly*, como *IntToStr*, por exemplo.

A era das restrições

Como vimos, escrever programas para controlar a porta paralela no tempo do velho *MS-DOS* era muito simples, assim como na época do *Windows 95/98*. Podíamos utilizar as funções *WritePort* e *ReadPort* sem nenhuma restrição. Mas ao entrar na nova era do *Windows NT, 2000* e *XP*, toda esta simplicidade foi retirada. E com certeza, se em alguma ocasião você já tentou controlar as portas do PC em *Windows NT*, deve ter se deparado com problemas.

Quando tentamos rodar um programa que utiliza funções convencionais de acesso ao *hardware* nestas versões *Windows*, o sistema operacional mostrará uma mensagem de exceção de instrução privilegiada (**Figura 2**).

Sendo um sistema operacional seguro, o *Windows NT* confere alguns privilégios e restrições aos diferentes tipos de programa que operam sobre ele. Ele classifica todos os programas em duas categorias: modo *usuário* e modo *kernel* (**Figura 3**).

Os programas que geralmente criamos estão na categoria do modo usuário. Programas de modo usuário são restritos a certas instruções como entrada, saída, etc. Toda vez que um programa de modo usuário tenta executar tais instruções, o sistema operacional interrompe a execução deste programa e mostra uma mensagem de erro.

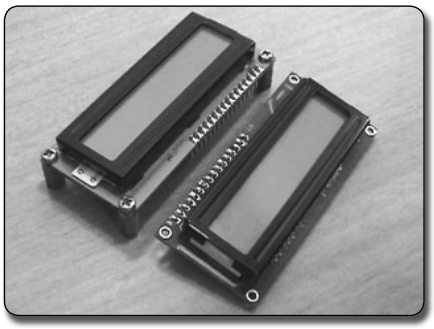


Figura 1. Interfaceamento de módulos LCD

Figura 2. Mensagem erro durante uma tentativa de acesso ao hardware

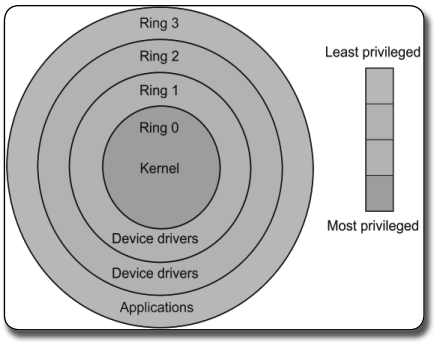


Figura 3. Arquitetura de acesso ao hardware em Windows NT/2000/XP

Nota do DevMan

A porta paralela é uma ferramenta de controle simples e barata para construção de dispositivos e projetos. A simplicidade e facilidade de programação fazem da comunicação paralela a mais popular dos *hobbistas* eletrônicos, ela também é utilizada em robôs controlados por computador, programadores de micro controladores, automação doméstica, etc, por isso sua utilização nesse artigo é indispensável, visto que conseguiremos elaborar um exemplo simples, mas bastante detalhado.

Listagem 1. Código para escrita e leitura na porta paralela nas versões do Windows 95 e 98

```
procedure WritePort(Port: word; Value: byte);
asm
 push word ptr Port
 mov al, Value
 pop dx
 out dx, al
end;

function ReadPort(Port: word): byte;
asm
 mov dx, Port
 in al, dx
end;
```

Em resumo: nossos programas de interfaceamento são interrompidos quando executam instruções de entrada ou saída para leitura ou escrita de dados na porta paralela. Mas ao mesmo tempo programas de modo *kernel* não são restritos de nenhuma forma ao executar tais instruções. *Device drivers* são capazes de rodar em modo *kernel*. Então para solução do problema de acesso ao *hardware* se resume em escrever um *driver* de modo *kernel* capaz de ler e escrever dados na porta e permitir ao programa de modo usuário comunicar-se com ele.

Só que escrever um *driver* não é tarefa nada fácil, mesmo para os programadores mais experientes. Então para obter acesso ao hardware necessitamos utilizar *drivers* (*DLLs*). Sinceramente uma das vantagens, ou pontos fortes do *Delphi* é que, normalmente, não usamos *DLLs*, mas realmente não existe outra forma de acesso ao *hardware* que não seja feita através delas.

Mesmo os componentes de acesso ao *hardware* como o *LPTPort* fazem uso de *drivers* para obter acesso direto ao equipamento.

Acessando o hardware com o uso de DLL

Uma das *DLLs* mais utilizadas para acesso direto ao *hardware* é a *InpOut32.dll*, que pode facilmente ser baixada no site do desenvolvedor em *www.logix4u.net*. A **Listagem 2** apresenta um exemplo de utilização da *InpOut32.dll* em uma *unit* no *Delphi*.

A *InpOut32.dll* possui as seguintes características:

- Funciona nas versões *do Windows 98/NT/2000* e *XP*;
- Usa um *driver kernel* encapsulada em uma *DLL*;
- Não requer nenhuma instalação de *software* ou *driver*;
- O *driver* é automaticamente instalado e configurado quando a *DLL* é carregada;
- Nenhuma *API* especial é necessária, pois são apenas duas funções principais (*Inp32* e *Out32*);
- Pode ser facilmente utilizada com *Delphi*.

Para importar métodos ou funções de *DLLs* devemos utilizar a palavra reservada *external* para informar que se trata de uma função externa e a diretiva *stdcall* é um comando que permite que outras linguagens possam fazer chamadas às funções.

Agora que fizemos uma breve recapitulação sobre formas de acesso ao *hardware* iremos programar nosso projeto de controle de *hardware*.

Listagem 2. Código exemplo de utilização da InpOut32.dll no Delphi

```
unit InpOut32;

interface

uses
SysUtils, Windows, Messages, Classes;

implementation

{ Definição das funções da InpOut32.dll }
function InPortb(EndPorta: integer): byte stdcall; external 'inpout32.DLL' name 'Inp32';

procedure OutPortb(EndPorta: integer; Valor:byte); stdcall;
  external 'inpout32.DLL' name 'Out32';
end.
```

Pino	Nome	Função
1	Vss	Terra
2	Vdd	5V
3	Vee	Tensão de Contraste
4	RS	Seleção de Registro (Register Select)
5	R/W	Leitura/Escrita (Read/Write)
6	E	Habilitar (Enable/Clock)
7	D0	Bit de dados 1
8	D1	Bit de dados 2
9	D2	Bit de dados 3
10	D3	Bit de dados 4
11	D4	Bit de dados 5
12	D5	Bit de dados 6
13	D6	Bit de dados 7
14	D7	Bit de dados 8
15	A	Anodo do BackLight (se houver)
16	K	Catodo do BackLight (se houver)

Tabela 1. Tabela Pinagem LCD HD44780 ou compatível

Detalhes importantes sobre LCDs

LCDs são muito utilizados em interfaces eletrônicas, para indicar o estado de diversos equipamentos. Além disso, eles têm sido extensivamente utilizados em *CaseMod*.

Existem diversos projetos de como se utilizar um *LCD* para mostrar diversas informações do PC como velocidade e temperatura do processador, o horário local, a lista de músicas do *Winamp*, etc. Inclusive existem programas especializados em controle de *LCD* como o *LCDSmartie* que é *opensource* e que foi totalmente desenvolvido em *Delphi*.

Porta Paralela	LCD HD44780
D0 (Pino 2)	D0 (Pino 7)
D1 (Pino 3)	D1 (Pino 8)
D2 (Pino 4)	D2 (Pino 9)
D3 (Pino 5)	D3 (Pino 10)
D4 (Pino 6)	D4 (Pino 11)
D5 (Pino 7)	D5 (Pino 12)
D6 (Pino 8)	D6 (Pino 13)
D7 (Pino 9)	D7 (Pino 14)
Strobe (Pino 1)	Enable (Pino 6)
Error (Pino 14)	Read/Write (Pino 5)
Select In (Pino 16)	Register Select (Pino 4)

Tabela 2. Tabela de ligação Porta Paralela - LCD HD44780

Desenvolveremos um programa para controlar um *LCD 16x2* com o qual poderemos realizar a inicialização do *display*, escrever textos, enviar comandos, etc. Tudo isto através da porta paralela do PC e utilizando o componente *LPTPort* que na minha opinião oferece o método mais simples e fácil de acesso à porta paralela.

Antes de iniciarmos a programação do *LCD*, devemos antes conhecer alguns detalhes importantes que serão necessários para a execução deste projeto, entre eles qual tipo de *LCD* utilizaremos, qual a *pinagem* do *display*, esquema de ligação, comandos, etc.

Utilizaremos um módulo *LCD* alfanumérico com um controlador *Hitachi HD44780* (ou compatível), pois este tipo de módulo pode ser mais facilmente encontrado. O controlador do *display* é encarregado de controlar a matriz de cristal líquido. É óbvio que este tipo de módulo não oferece as mesmas vantagens de módulos mais modernos que são utilizados em celulares, *laptops* e muitos outros equipamentos, mas sua simplicidade será uma vantagem no momento da montagem do mesmo. Os módulos de *LCD* possuem um padrão de pinos. Normalmente os que não têm *backlight* (luz de fundo) possuem 14 pinos e os que possuem *backlight* têm 16 pinos. Esses dois pinos a mais são responsáveis pela ativação da luz de fundo. A **Tabela 1** mostra em detalhes os pinos de um módulo *LCD 44780* e suas respectivas funções.

Os pinos 1 e 2., *Terra* e *5V* respectivamente, são utilizados para energizar o módulo. O pino 3 é utilizado para controlar o contraste do *display*. O ideal é que este pino esteja conectado a uma fonte variável de tensão, para que dessa forma possamos variar a tensão e conseqüentemente o contraste. O pino 4 é o *Seletor de Registro* (*Register Select*) que é o primeiro de 3 pinos de controle.

Quando este pino está em baixa tensão (0V) então todos os *bytes* enviados ao *display* são tratados como comandos, e *bytes* lidos indicam seu estado. Ao colocarmos este pino em nível alto (5V) os dados de caracteres podem ser transmitidos do/para módulo.

O pino 5 é o *Leitura/Escrita* (*Read/Write*). Este pino deve ser colocado em nível baixo para escrever comandos ou caracteres no *display*, em nível alto para ler caracteres ou informação de status dos registros. O pino 6 é o *Habilitar* (*Enable*), utilizado para iniciar a transmissão de comandos ou caracteres dos pinos de dados. Quando escrevemos no *display*, os dados apenas são transmitidos na transição de nível alto para baixo deste pino. Porém quando lemos, os dados estarão disponíveis depois da transição de alto para nível baixo e outra transição de baixo para alto novamente.

E para finalizar, os pinos de 7 a 14 são os pinos de dados que podem ser transmitidos como um *byte* (8 bits) ou dois *nibbles* (4bits), veja o diagrama elétrico do display na **Figura 4**.

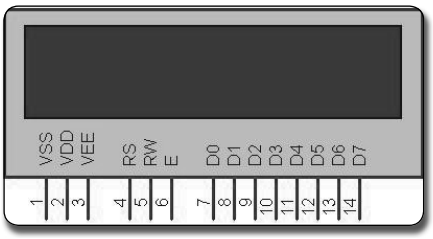


Figura 4. Diagrama elétrico de um LCD HD44780

Dicas de montagem

Para aqueles que não possuem nenhuma ou pouca experiência em eletrônica, recomendo a leitura dos tutoriais do site *www.casemodbr.com* onde é possível encontrar uma lista de materiais e ferramentas necessárias para a construção de um cabo para interfacear o *LCD* a porta paralela. A **Tabela 2** mostra as ligações necessárias que o leitor deve fazer para ligar e comandar o *LCD* pelo *Delphi*.

Nota: A montagem que utilizei neste artigo é uma montagem simplificada, visto que não iremos ler nenhum dado do *display*, sugiro que o leitor em sua montagem coloque o pino *Read/Write* (pino 5 do *LCD*) no terra. Desta forma não precisaremos nos preocupar em alterar o valor do pino 14 da porta paralela. Nosso exemplo se baseia em apenas escrever comandos e dados no *LCD* e só funcionará corretamente se o *Read/Write* estiver ligado ao terra.

Criando a classe THD44780

Agora que já entendemos o funcionamento básico do *display* iremos iniciar a construção de nossa aplicação de controle de *LCDs*. Para isso abra o *Delphi*, crie uma nova aplicação, que batizaremos

Nota do DevMan

A Porta Paralela está ligada diretamente à placa mãe de seu computador, por isso recomendo aos leitores que tenham muito cuidado ao conectar circuitos eletrônicos a essa porta, pois, uma descarga elétrica ou um componente com a polaridade invertida, poderá causar danos irreparáveis ao seu computador.

Todos nós da ClubeDelphi, não nos responsabilizamos por nenhum dano causado tanto no equipamento, como na montagem/instalação inadequada da interface.

Listagem 3. Definição da classe THD44780

```
type
  THD44780 = class
    private
    public
      Columns : integer;
      Lines   : integer;
      LPTPort : TLPTPort;
      procedure InitializeLCD;
      procedure ClearDisplay;
      procedure PosCursor(X, Y : integer);
      procedure Cursor_Display(CursorOn, DisplayOn, BlinkCursor : Boolean);
      procedure ShiftCursor_Message(ShiftText, ToRight : Boolean);
      procedure WriteText(Text : string);
      procedure SendCommand(Command : byte);
      procedure SendData(Data : byte);
      procedure WritePort(Address : word; Value : byte);
      constructor Create(LPT: TLPTPort);
      destructor  Destroy; override;
  end;
```

de *LCD.dpr*, e salve a *unit* como *uMain.pas*, coloque um componente *LPTPort* no *form* principal que será renomeado para *frmMain*. Criaremos inicialmente uma classe com a qual poderemos comandar o *LCD* mais facilmente. Crie uma nova *unit*, e salve-a com o nome de *uHD44780.pas*, nesta *unit* criaremos a classe de controle do *LCD* conforme a **Listagem 3**.

Com isso iremos encapsular todas as funcionalidades de controle de um *LCD* na classe *THD44780*, mais adiante, estaremos abordando cada um dos métodos desta nossa classe. Reparem que nossa classe possui algumas variáveis que podem ser visualizadas em seguida:

```
public
  Columns : integer;
  Lines   : integer;
  LPTPort : TLPTPort;
```

As variáveis *Columns* e *Lines* serão utilizadas para definir respectivamente o número de colunas e linhas de nosso *LCD*, nesse caso, em específico, estamos utilizando um *LCD 16x2* (16 colunas e 2 linhas). E para finalizar a variável *LPTPort*, que é uma variável do tipo *TLPTPort*, que irá realizar todo acesso à porta paralela.

Por enquanto declare a classe *THD44780* e tecle *Ctrl+Shift+C* para que o Delphi crie o cabeçalho das funções declaradas. A seguir, a **Listagem 4,** onde encontramos os métodos *Create* e *Destroy* da nossa classe *THD44780*.

Nota: O componente *LTPPort* encontra-se disponível juntamente com o código-fonte desse artigo no site da ClubeDelphi.

Como podemos observar, esta classe quando criada, recebe o parâmetro *LPT*, que na verdade é apenas um ponteiro para uma variável do tipo *TLPTPort* que será o componente utilizado para acessar a porta paralela como já dito anteriormente. Depois configuramos o número de Colunas e Linhas através das variáveis *Columns* e *Lines*.

Quanto ao método *WritePort*, que pode ser visto nas duas últimas linhas do método *Create*, é apenas uma subrotina para configurar o endereço e o valor da porta, desta forma evitaremos utilizar 2 linhas de comando repetitivamente, o método *WritePort* pode ser visto a seguir:

```
procedure THD44780.WritePort(Address : word;
Value : byte);
begin
  LPTPort.SelectPort(Address);
  LPTPort.Out(Value);
end;
```

Comando	Valor Hexadecimal	Tempo de execução mínimo
Limpar display	01	1,6 milissegundos
Cursor no início	02 ou 03	40 microssegundos
Modo de Funcionamento	04 a 07	40 microssegundos
Display On/Off e Cursor	08 a 0F	40 microssegundos
Desloca Cursor/Display	10 a 1F	40 microssegundos
Modo de Utilização	20 a 3F	40 microssegundos
Configurar DDRAM	80 a FF	40 microssegundos

Tabela 3. Tabela de comandos do LCD HD44780

Listagem 4. Métodos Create e Destroy da classe THD44780

```
constructor THD44780.Create(LPT: TLPTPort);
begin
  LPTPort := LPT;
  Columns := 16;
  Lines := 2;
  WritePort($378, 0);
  WritePort($37A, 0);
end;

destructor THD44780.Destroy;
begin
  LPTPort.Free;
  inherited Destroy;
end;
```

Listagem 5. Constantes que serão utilizadas na classe THD44780

```
const
{ Comandos }
LCD_LIMPA     = $01;  // Limpa o Display                          00000001
LCD_HOME      = $02;  // Coloca o cursor na 1. linha e 1. coluna  00000010
LCD_ONOFF     = $08;  // Controle do display:                     00001000
LCD_ONOFF_CP  = $01;  // Cursor piscante ou não                   00000001
LCD_ONOFF_C   = $02;  // Liga cursor ou não                       00000010
LCD_ONOFF_D   = $04;  // Liga Display ou não                      00000100
LCD_DESLOCA   = $10;  // Deslocamento do Cursor:                  00010000
LCD_DESLOCA_D = $04;  // Desloca pra direita ou esquerda          00000100
LCD_DESLOCA_M = $08;  // Deslocar Mensagem ou o cursor            00001000
LCD_FUNCAO    = $20;  // Modo de Utilização:                      00100000
LCD_FUNCAO_M  = $04;  // Usar Matriz 5x10 ou 5x7                  00000100
LCD_FUNCAO_L  = $08;  // 2 ou mais linhas ou 1 linha              00001000
LCD_FUNCAO_B  = $10;  // 8 bits ou 4 bits                         00010000
LCD_DDRAM     = $80;  // Estabelece endereço da DDRAM             10000000
```

Estamos, portanto utilizando o método *WritePort* para colocar o endereço de dados e controle da porta em zero. E no método *Destroy* liberamos o *LPTPort*, que está associado com o *HD44780*, da memória além de destruir nossa classe de controle.

Ligando o LCD

Energize o *LCD*, e repare que o mesmo deve estar apresentando uma série de quadrados na linha superior do *LCD* (**Figura 5**).

Neste estado inicial do *LCD*, mesmo que enviemos algum texto, ele não poderá ser visualizado. É necessário enviar algum comando ao display para que ele possa inicializar. Agora necessitamos criar uma rotina que servirá para enviar um comando para o *LCD*, a **Tabela 3** mostra uma série de comandos possíveis do *HD44780*.

Para auxiliar a utilização do *LCD*, também utilizaremos diversas constantes que facilitarão o envio dos comandos. A lista das constantes pode ser vista na **Listagem 5.**

Enviando Comandos ao LCD

Agora faremos algumas rotinas básicas para envio de comandos e dados. Conheceremos em detalhes como manipular o *LCD*. Para enviar um comando necessitaremos de seguir alguns passos:

1. Colocar o pino *Register Select* em baixa (para que o *display* interprete o sinal enviado como comando ou instrução), neste momento o pino *Enable* deverá estar em alta (pois apenas durante a transição de nível alto para baixo deste pino é que o *display* irá de fato processar o comando) e aguardar certo tempo até que o *LCD* possa processar novos comandos ou dados (uma vez que existe um tempo mínimo necessário para cada tipo de comando (**Tabela 4**));

2. Enviar o comando que desejamos e novo *delay* (espera);

3. E para finalizar o envio do comando colocamos o *Enable* em baixa e aguardamos outro *delay* cujo tempo é baseado no comando processado.

Agora que já sabemos como enviar um comando ao *display*, podemos implementar o método *SendCommand* que está na **Listagem 6**.

Nota: Um detalhe muito importante, é que para cada comando recebido pelo LCD existe um tempo mínimo que devemos aguardar para que o LCD possa processar novos comandos ou dados, normalmente este tempo de resposta varia desde microssegundos até alguns milissegundos. No nosso caso não necessitamos de tanta precisão e resolução de tempo, podemos utilizar o método Sleep do Delphi que interrompe a execução de um programa por um determinado tempo especificado em milissegundos.

Os pinos de controle do *LCD* estão sendo controlados pelo registro de controle da porta, por isso inicialmente escrevemos o valor 0 no endereço $37A da porta, desta forma teremos o pino *Enable* em nível lógico alto (5V) e o pino *Register Select* em nível lógico baixo (0V). Assim o *display* interpretará o sinal enviado como um comando. Devemos aguardar 1 milissegundo para enviar o parâmetro *Command* ao registro de dados que corresponde ao registro de dados da porta, definido pelo endereço $378 e aguardamos mais 1 milissegundo para finalizar o envio do comando, fazendo com que o *Enable* fique em nível baixo e concretize o envio do comando. Repare que dependendo do comando enviado (se é um limpar *display*, ou cursor no início) existe a necessidade de aguardar um tempo de 1 ou 2 milissegundos.

Listagem 6. Método de envio de comandos do THD44780

```
procedure THD44780.SendCommand(Command : Byte);
begin
  {Register Select em baixo e Enable em alto}
  {Obs: o pino 1 (Strobe) é invertido no hardware da porta}

  { Envia 00000000 para registro de controle }
  WritePort($37A,0);
  { 1 milisegundo }
  Sleep(1);
  { Envia comando para registro de dados }
  WritePort($378, Command);
  Sleep(1);
  {Register Select em baixo e Enable em baixo}
  { Envia 00000001 para registro de controle }
  WritePort($37A,1);
  if (Command = LCD_LIMPA) or (Command = LCD_HOME) then
    { 2 milissegundos }
    Sleep(2)
  else
    Sleep(1);
end;
```

Listagem 7. Método de envio de dados para o THD44780

```
procedure THD44780.SendData(Data : Byte);
begin
  {Register Select em alto e Enable em alto}
  WritePort($37A,4); // envia 00000100 para registro de controle
  sleep(1);
  WritePort($378,Data); //envia dados para registro de dados
  sleep(1);
  {Register Select em alto e Enable em baixo}
  WritePort($37A,5); // envia 00000101 para registro de controle
  sleep(1);
end;
```

Para enviar dados ao *LCD*, necessitamos utilizar o pino *Register Select* em baixa para que o sinal enviado seja interpretado como um dado a ser mostrado no *display*. Segue o procedimento que devemos utilizar:

1. Colocar o pino *Register Select* em alta (para envio de dados), neste momento o pino *Enable* deverá estar em alta e aguardar certo tempo até que o *LCD* possa processar novos comandos ou dados;

2. Enviar os dados e novo tempo de espera;

3. E para finalizar o envio dos dados colocamos o *Enable* em baixa e aguardamos outro tempo.

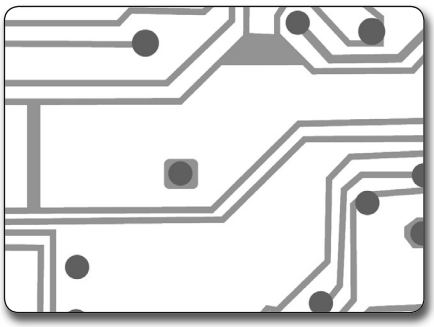
Agora que já conhecemos a forma de envio de dados, iremos construir nossa rotina *SendData* de acordo com a **Listagem 7**.

No método *SendData*, teremos basicamente o mesmo padrão do *SendCommand* sendo a principal diferença o fato de mantermos o *Register Select* em nível alto com o *WritePort* no endereço $37A (para que o sinal da porta seja interpretado

Nota do DevMan

O registro de controle da porta paralela é composto pelos pinos 1 (C0), 14 (C1), 16 (C2) e 17 (C3), com exceção do pino 16, todos os demais estão invertidos no hardware da porta paralela, em outras palavras, quando estiverem em nível lógico baixo, eles apresentam tensão de 5V.

Figura 5. Quando ligado o LCD mostra esse aspecto



Listagem 8. Método para configurar os parâmetros de controle do display

```
procedure THD44780.Cursor_Display(CursorOn, DisplayOn, BlinkCursor : Boolean);
var
  Temp : byte;
begin
  Temp := LCD_ONOFF;
  if CursorOn then Temp := Temp or LCD_ONOFF_C;
  if DisplayOn then Temp := Temp or LCD_ONOFF_D;
  if BlinkCursor then Temp := Temp or LCD_ONOFF_CP;
  SendCommand(Temp);
end;
```

Listagem 9. Método para posicionamento do cursor

```
procedure THD44780.PosCursor(X, Y : Integer);
var
  offset : Integer;
begin
  offset := 0;
  if X > Columns then
    begin
    X := 1;
    inc(Y);
    end;
  if Y > Lines then
    Y := 1;
  if Y = 1 then offset := 0;
  if Y = 2 then offset := 64;

  SendCommand(LCD_DDRAM + offset + (X-1));
end;
```

Listagem 10. Método para deslocar cursor e/ou texto

```
procedure THD44780.ShiftCursor_Message(ShiftText, ToRight : Boolean);
var
  Temp : byte;
begin
  Temp := LCD_DESLOCA;
  if ShiftText then Temp := Temp or LCD_DESLOCA_M;
  if ToRight then Temp := Temp or LCD_DESLOCA_D;
  SendCommand(Temp);
end;
```

como dados e não como comandos) . Depois do *delay* enviamos os dados ao *display* e finalizamos baixando o *Enable* e aguardando o tempo de processamento do *LCD*.

Inicialização do LCD

Finalmente utilizando a rotina de envio de comandos *SendCommand*, inicializaremos o *display* com o método *InitializeLCD* como mostrado a seguir:

```
procedure THD44780.InitializeLCD;
begin
  SendCommand(LCD_FUNCAO or LCD_FUNCAO_B or
LCD_FUNCAO_L);
  sleep(15);
end;
```

No *InitializeLCD* configuramos o modo de operação enviando *LCD_FUNCAO* que será a máscara de bits para o modo de utilização do *LCD*, ou seja, diremos ao *LCD* que queremos configurar o modo de utilização, portanto dentro desta máscara de bits principal, estaremos agregando valores que irão configurar outros parâmetros. Usaremos a constante *LCD_FUNCAO_B*, que configura o modo de envio de dados para 8 bits e *LCD_FUNCAO_L* que configura o modo de 2 linhas para o *display*.

Repare que utilizamos o operador lógico *or* para que possamos configurar todos estes modos de operação ao mesmo tempo. E aguardamos um tempo de aproximadamente 15 milissegundos para completar a inicialização do *display*.

Parâmetros de controle do display

Com o *display* já inicializado, devemos configurar alguns parâmetros de controle de *display*, são eles: *cursor ativo*, *display ligado* e *cursor piscando*.

No método *Cursor_Display* teremos 3 parâmetros booleanos *CursorOn*, *DisplayOn*, *BlinkCursor* que utilizaremos para configurar o estado do display, conforme mostra a **Listagem 8**.

Nesta rotina, teremos a variável *Temp* que será o byte que desejamos enviar com a máscara de bits para o controle do *display* (usando a constante *LCD_ONOFF*), com isso faremos uma varredura para sabermos em quais parâmetros temos o valor *True* e mediante o uso do operador *or* agregamos o valor da respectiva constante a máscara do controle do *display LCD_ONOFF*. Depois desta varredura enviamos o valor como um comando ao *display* usando *SendCommand*.

Limpando o LCD

Para que possamos limpar o LCD sempre que desejarmos, precisaremos criar agora o comando de limpeza do *display*, bastando apenas enviar o valor da constante *LCD_LIMPA*, utilizando o método *SendCommand*, conforme listado a seguir. Crie uma nova *procedure* com o nome *ClearDisplay* e digite o código como mostrado. Não há segredo. Depois bastará fazer uma chamada a esse método em algum controle de nosso formulário:

```
procedure THD44780.ClearDisplay;
begin
  SendCommand(LCD_LIMPA);
end;
```

Posicionando o cursor

É possível controlar também o posicionamento do cursor, para isso necessitamos utilizar a constante *LCD_DDRAM* que nos proporciona acesso ao endereço da posição de dados do *display* (*DDRAM*) veja a **Listagem 9**.

Observe o leitor que através dos parâmetros X e Y do *PosCursor* podemos apontar uma nova posição do cursor, caso o valor de X seja superior ao número de colunas. Então colocamos a variável X na primeira posição da próxima linha. E caso a variável Y seja maior que o número de linhas atribuímos a Y o valor de 1, ou seja, posicionamos o cursor na primeira linha.

Repare nas duas últimas instruções *if*. Estamos tratando da primeira linha, então a variável *offset* recebe o valor de 0, do contrário recebe o valor 64, este *offset* se refere a posição da *DDRAM*.

Com isso podemos enviar o comando através do *SendCommand* utilizando a constante *LCD_DDRAM* para que o *LCD* entenda que se trata do endereçamento da posição do cursor e agregaremos o

valor do *offset* (que representa a linha em que queremos posicionar o cursor) e a posição da variável X – 1 (pois a primeira linha de endereços da *DDRAM* vai de 0 a 15).

Deslocando o cursor e/ou o texto

O *LCD HD44780* é muito versátil, podemos deslocar o cursor, o texto ou ambos ao mesmo tempo. Para isso utilizaremos a constante *LCD_DESLOCA* como máscara de bits principais para acessar a instrução para deslocar o cursor e texto. Para isso digite o conteúdo da **Listagem 10** no método *ShiftCursor_Message*.

Neste método receberemos 2 parâmetros do tipo *Boolean*, o primeiro *ShifText* que se for *True* faz com que além do cursor o texto também seja deslocado e o segundo *ToRight* que se for *True* desloca o texto e/ou cursor para direita, caso contrário faz o deslocamento para esquerda. Repare que adicionamos o valor das constantes a nossa variável local *Temp* através de operações com *or*. Enviando em seguida o valor acumulado em *Temp* com o comando *SendCommand*.

Escrevendo caracteres no LCD

Creio que agora chegamos ao método mais aguardado de todos, enviar texto ao *LCD*. Para realizar esta tarefa faremos uso do método *SendData* criado anteriormente, podemos ver a implementação do método *WriteText* na **Listagem 11**.

Note que para que possamos enviar o parâmetro *Text*, que é uma *string*, ao *LCD* fazemos uso de um laço *for* do primeiro caractere até o número máximo de caracteres recuperados através do método *Length* da variável *Text*, enviando um caractere por vez (tendo sido convertido para seu valor da tabela *ASCII* através função *Ord*). Toda vez que receber um caracter, o *LCD* o escreve na tela e move o cursor um espaço para a direita. O cursor marca o local onde o próximo caracter vai ser escrito.

Listagem 11. Método escrever caracteres no display

```
procedure THD44780.WriteText(Text : string);
var
  i : byte;
begin
  for i := 1 to Length(Text) do
    SendData(Ord(Text[i]));
end;
```

Listagem 12. Clique do btnInicializar

```
procedure TfrmMain.btnInicializarClick(Sender: TObject);
begin
  LCD.InitializeLCD;
  LCD.ClearDisplay;
  sleep(1);
  LCD.WriteText('Controle de LCD');
  LCD.PosCursor(1,2);
  LCD.WriteText('em Delphi!');
  LCD.Cursor_Display(True,True,True);
end;
```

Agora que já temos nossa classe pronta, podemos voltar à construção de nossa aplicação exemplo, declarando na seção *uses* da *uMain* a *unit uHD44780* que contém a classe e os métodos de controle do *LCD*. Não se esqueça de declarar uma variável do tipo *THD44780* conforme segue:

```
LCD : THD44780;
```

Com essa variável teremos acesso a todas as funcionalidades já implementadas de nossa classe especial, e poderemos controlar o *LCD* sem maiores dificuldades. Agora dê um duplo clique no *frmMain* e no evento *OnCreate* digite o código a seguir:

```
LPTPort1.Port[$378];
LCD := THD44780.Create(LPTPort1);
```

Com este código associamos o componente *LPTPort1* que colocamos em nosso *form* principal ao endereço $378 da porta paralela que é endereço base da porta, e instanciamos a variável *LCD* do tipo *THD44780*. Passamos como parâmetro do método *Create* o *LPTPort1* do nosso *form*, associando assim o *LPTPort1* a nossa variável de acesso ao *display*.

Não se esqueça de no evento *Destroy* do *frmMain* liberar o *LCD* da memória com o seguinte comando:

```
LCD.Free;
```

Precisamos agora de um *Button* para abrir a lista das portas paralelas do micro e um *ComboBox* onde armazenaremos a lista de portas. Coloque um *Button* com a propriedade *Name* alterada para *btnAbrir* e um *ComboBox* renomeado para *cmbxLPT*. Dê um duplo clique no *btnAbrir* e digite o código que segue no evento *OnClick* do botão:

```
LCD.LPTPort.Open;
```

Utilizando o método *Open* do *LPTPort* carregamos todas as *dlls* necessárias para acesso ao *hardware*, a partir daqui já temos permissão para listar as portas do micro. Para isso no evento *DropDown* do *cmbxLPT* coloque a seguinte linha de código:

```
LCD.LPTPort.EnumPorts(cmbxLPT.Items);
```

Com esta linha de código listamos todas as portas paralelas existentes no computador e armazenamos esta lista no *cmbxLPT*. Já temos agora todo controle sobre as portas paralelas e partiremos ao controle e inicialização do *LCD*. Coloque 3 *Buttons* cujas propriedades *Name* serão alteradas respectivamente para *btnInicializar*, *btnEscrever* e *btnLimpar*. No evento *OnClick* do *btnInicializar* iremos inicializar o *LCD* através do código da **Listagem 12.**

No clique do *btnInicializar*, faremos uso do método *InitializeLCD* que fará a inicialização do *display*, e só para nos certificarmos enviamos também um comando para limpar o *display* e fazemos um *delay* de 1 milissegundo. Depois

Nota do DevMan

O LCD HD44780 16x2 possui 32 posições para caracteres, ou seja, 16 posições para cada linha. O DDRAM ou Display Data RAM é uma memória volátil usada para armazenar os caracteres do display, possuindo 80 bytes, destes 80 apenas 32 posições são visíveis (as que são mostradas no display), caso escrevamos algum caractere fora destas 32 posições ele fica armazenado na DDRAM.

Listagem 13. Código de deslocamento de cursor ou texto (SpeedButtons)

```
procedure TfrmMain.spdbtnCursorEsqClick(Sender: TObject);
begin
  LCD.ShiftCursor_Message(False, False);
end;

procedure TfrmMain.spdbtnCursorDirClick(Sender: TObject);
begin
  LCD.ShiftCursor_Message(False, True);
end;

procedure TfrmMain.spdbtnTextoEsqClick(Sender: TObject);
begin
  LCD.ShiftCursor_Message(True, False);
end;

procedure TfrmMain.spdbtnTextoDirClick(Sender: TObject);
begin
  LCD.ShiftCursor_Message(True, True);
end;
```

disto com o método *WriteText* escrevemos o texto *Controle de LCD*, lembrando que assim que inicializado, o cursor do *LCD* já está apontando para a primeira posição da linha 1.

Colocamos o cursor na primeira posição da segunda linha usando o método *PosCursor* e nesta posição escrevemos na segunda linha o texto *em Delphi!* e ativamos o cursor, o *display* e cursor piscando com o método *Cursor_Display* passando todos os parâmetros como *True*, veja a tela do *LCD* após o clique do *btnInicializar* na **Figura 6**.

Voltando aos demais botões, digite no evento *OnClick* do *btnLimpar* o comando de limpeza do *display*:

```
LCD.ClearDisplay;
```

Para escrever texto através de nosso exemplo necessitamos de um *Edit*. Coloque um componente *Edit* no *form* principal e altere seu nome para *edtMensagem*, todo texto que digitarmos nele será enviado através do clique do *btnEscrever* que segue:

```
LCD.WriteText(edtMensagem.Text);
```

Agora colocaremos 3 *CheckBoxes* que utilizaremos para alterar a forma como o cursor está sendo mostrado. Depois de colocá-los, altere seus nomes para *chckbxCursorLigado*, *chckbxDisplayLigado* e *chckbxCursorPiscando*, respectivamente. Queremos que toda vez que alterar a propriedade *Checked* de qualquer um deste *CheckBoxes* o *LCD* atualize as configurações do cursor, para isso dê um duplo clique no *chckbxCursorLigado* e digite o código a seguir:

```
LCD.Cursor_
  Display(chckbxCursorLigado.Checked,
  chckbxDisplayLigado.Checked,
  chckbxCursorPiscando.Checked);
```

Aponte os eventos dos outros *CheckBoxes* para o evento *chckbxCursorLigadoClick*.

Movendo o cursor e/ou a mensagem

Para finalizar nosso exemplo necessitamos de 4 *SpeedButtons* (*spdbtnCursorEsq*, *spdbtnCursorDir*, *spdbtnTextoEsq*, *spdbtnTextoDir*) que serão utilizados para deslocar o cursor e a mensagem para a direita ou esquerda, utilizando o método *ShiftCursor_Message*. A **Listagem 13** apresenta o código dos *SpeedButtons*.

Repare que apenas alteramos os parâmetros do método *ShiftCursor_Message* que são utilizados para deslocar texto e cursor. Agora já temos nossa aplicação pronta para demonstrar algumas funcionalidades de controle de *LCD*. Na **Figura 7** temos uma sugestão de GUI que o caro leitor pode utilizar durante o desenvolvimento do programa.

Conclusão

Com nossa classe de controle de *LCDs* fica muito fácil interfacear e controlar *displays*, que são uma grande alternativa aos *hobbistas* e praticantes de *CaseModding*. É importante estudarmos essas metodologias que nos ajudam no desenvolvimento de diversas soluções. Recomendo a leitura de outros artigos e referências a esse tema. Fico por aqui, um forte abraço, e até a próxima. ●

Figura 6. Tela do LCD após clique do botão de inicialização

Figura 7. Sugestão de GUI para a aplicação exemplo

Links

Understanding User and Kernel Mode
http://www.codinghorror.com/blog/archives/001029.html

NT Drivers Faq
http://www.cmkrnl.com/faq01.html#1.2

LCD Interfacing
http://www.8051projects.net/lcd-interfacing/

Accessing Port Hardware
http://sheepdogguides.com/dt4j.htm

HD44780
http://pic.rocklizard.org/ReferenceDocs/HD44780HowTo.pdf

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback