VFD-Studio

Hardware

Inhalt:

1. Das Display
2. Art des Anschlusses
3. LPT-Register
4. Anschluss des Displays
5. Hardwarezugriff auf den LPT-Port mit Delphi
6. Testgerät
7. Kommunikation mit LPT-Treiber
8. Zur Funktion von allowIO und PortTalk
9. Alternativen zu PortTalk
10. Benutzung der DL-PortIO-Komonente

1. Das Display

Ursprünglich war ich auf der Suche nach einem grafikfähigem LC-Display zum Anschluss an den PC. Beim Besuchen diverser Websites stieß ich dabei zufällig auf einen Testbericht über das GU128x64-800A VFD der Firma Noritake-Itron (Quelle: www.moddingtech.de).
Sehr angetan von den Vorteilen der VFDs gegenüber LC-Displays (höher Kontrast, größerer Ablesewinkel, selbstleuchtend, ...) machte ich mich von nun an auf die Suche nach einem günstigen VFD und wurde bei www.eBay.us fündig.

Das erworbene Display ist vom Typ GU256x64-372 und ebenfalls von der Firma Noritake
Es hat, wie bereits die Bezeichnung vermuten lässt, mit 256 x 64 Pixeln den doppelten Anzeigebereich wie das GU128x64-800A, was einem Textausgabebereich von 8 Zeilen zu je 42 Zeichen entspricht.
Das Display benötigt eine Spannung von 5V bei einer Stromaufnahme zwischen 1.5 und 2 A.
Enthalten sind neben dem eigentlichen Display auf der Platine die Displaycontroller, DC/DC-Spannungswandler, 8 kB Flash-RAM und alle Steuerungsschaltkreise zum Betrieb, sowie zwei Stecker für Stromversorgung und Anschluss an einen PC/µC.

Das Display kann Grafiken und/oder Zeichen darstellen. Dazu gibt es intern einen 'Grafikbereich' und einen 'Zeichenbereich' im RAM-Adressraum, welche Screen1 (Grafik)
und Screen2 (Zeichen) genannt werden.
Beide Screens können auch gleichzeitig angezeigt werden und dabei mit AND, OR und XOR logisch verknüpft werden.
Es können sowohl Daten in den RAM geschrieben als auch gelesen werden.
Ein Zeichen (Char) besteht aus 6x8 Pixeln.

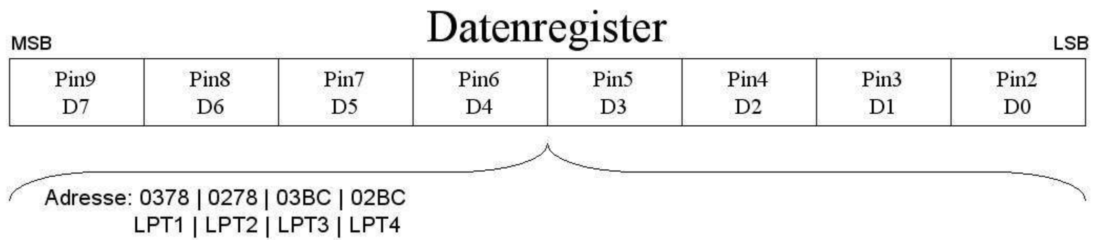
Weitere Details zum Display finden sich im beiliegenden Datenblatt.

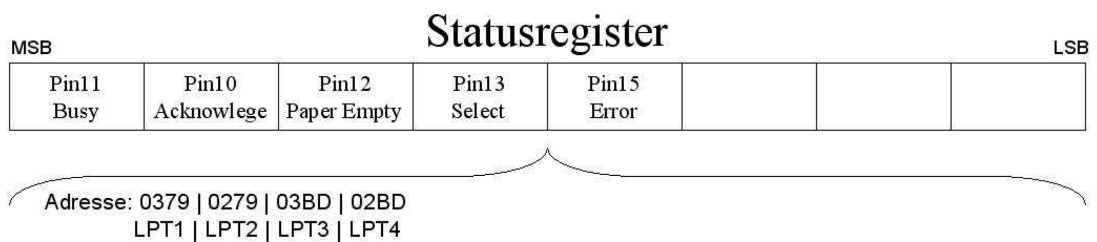
2. Art des Anschlusses

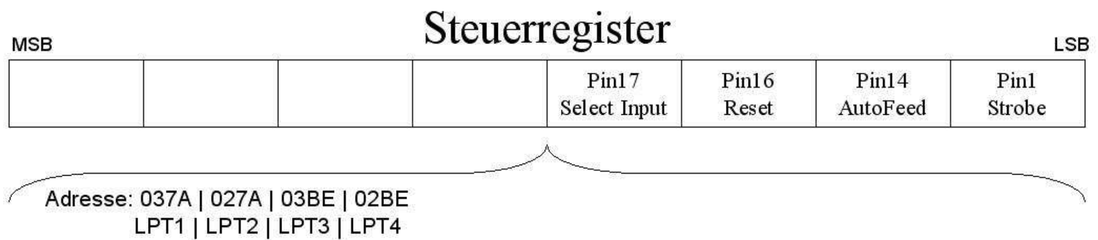
Bei der Entwicklung des Projektes stellte sich anfangs zunächst die Frage, über welchen Anschluss das Display mit dem PC kommunizieren sollte.
Zur Wahl stehen u.a. COM-Port, LPT und USB.
Der COM-Port hat den Vorteil programmiertechnisch leicht beherrschbar zu sein und einer hardwareseitigen Unempfindlichkeit gegen Kurzschlüsse. Allerdings müssen die seriellen Daten des COM-Ports mittels Schieberegister o.ä. in ein paralleles Format gewandelt werden, da das Display ja mindestens elf Datenleitungen aufweist (drei Steuerleitungen + D0..D7).
Für den USB-Port gilt das selbe Problem der seriellen Datenübertragung, so dass als Anschlussmöglichkeit der LPT-Port übrigbleibt. Dieser hat von sich aus acht Datenleitungen und vier bidirektionale Ein-/Ausgänge, womit zusätzlicher Schaltungsaufwand entfallen kann.
Allerdings ist der LPT-Port nicht kurzschlussfest und bei Anlegen einer Fehlspannung kann nicht nur der entsprechende Treiberbaustein Schaden nehmen, sondern auch das Mainboard zerstört werden!
Der LPT-Port kann bei kurzer Kabellänge dank seiner parallelen Arbeitsweise eine Datenübertragung von bis zu 1Mbit erreichen.
Angesprochen wird der LPT-Port über drei aufeinanderfolgende Adressen. Die erste, die Basisadresse, repräsentiert das Datenregister mit den Leitungen D_0 – D_7 an den Pins 2-9. Der Wert 255_d bedeutet z.B., dass alle Datenleitungen auf High-Pegel sind – 128_d entspräche 10000000_2 womit nur D_7 high wäre.
Bei den meisten PCs hat die Basisadresse des LPT1 den Wert 378_h.
Die nächste Adresse (Basisadresse + 1) repräsentiert das Statusregister. Dieses nur-lesen-Register hat 5 Eingangsleitungen *Busy*, *Acknowlege*, *Paper Empty*, *Select* und *Error*.
Das letzte Register (Basisadresse + 2) ist das Steuerregister mit den Leitungen *Strobe*, *Autofeed*, *Reset* und *Select Input*.

3. LPT-Register

Zur Übersicht hier die LPT-Register in grafischer Form:







4. Anschluss des Displays

Der LPT-Stecker wurde wie folgt mit dem 26poligen Displaystecker verbunden:

LPT-Stecker	**VFD 26pin Stecker**
1 STROBE	17 /WR
2 D0	15 D0
3 D1	13 D1
4 D2	11 D2
5 D3	9 D3
6 D4	7 D4
7 D5	5 D5
8 D6	3 D6
9 D7	1 D7
10 –13 n.c.	-
14 AUTO	/RD
15 n.c.	-
16 INIT	-
17 SELECT	C/D
18 – 25 GROUND	2,4,6,..24 GROUND und 23 (/CS)

Diese Anschlussbelegung wurde aus dem „PRINTER PORT INTERFACE.pdf“ der Fa. Noritake, enthalten in Support for 300/800/3xxx/7xxx Series Displays - Modified Promotion CD v2.01, übernommen.

Zudem wurde Pin 26 (/BL) des Displays mit +5V aus der Stromversorgung belegt und zwischen den Masseanschlüssen der Stromversorgung und der Stecker eine Verbindung hergestellt.
Zur Stromversorgung wurde ein PC-Y-Stromkabel an den Spannungsstecker des Displays gelötet, so dass es nun mit einem Computernetzgerät betrieben werden kann.

Abkürzungen:
D0 – D7: Datenleitungen
/WR Schreibsignal (lowsensitiv)
/RD Lesesignal (lowsensitiv)
/CS ChipSelect (lowsensitiv)
/BL DisplayBlanking (lowsensitiv) – ein Lowpegel an diesem Pin bewirkt, dass Grafik- und Charakter-Screen abgeschaltet werden

[„lowsensitiv“ bedeutet, dass der entsprechende Eingang ein Lowpegel aus logische ‚1’ interpretiert – es handelt sich also um invertierte Eingänge]

5. Hardwarezugriff auf den LPT-Port mit Delphi

Das Display sollte wie bereits erwähnt an den LPT-Port angeschlossen werden und die Software mit Delphi geschrieben werden.
Delphi kennt allerdings keine Methoden um direkt auf den Druckerport zu zugreifen um einzelne Bits zu manipulieren und auch die Windows-API bietet hier keine Lösung.
Eine Möglichkeit wäre nun natürlich zu diesem Zweck im Internet nach einer DLL oder API für Delphi zu suchen – eine einfache Lösung bietet Delphi jedoch selbst durch die Möglichkeit Assemblerroutinen einzubinden.
Mit Assemblerbefehlen ist es möglich Hardwareregister direkt zu schreiben oder zu lesen.
Zur Demonstration, wie Assemblerbefehle in Delphi eingebunden werden können, hier die Prozedur, die anfangs verwendet wurde um Daten über den LPT-Port auszugeben:

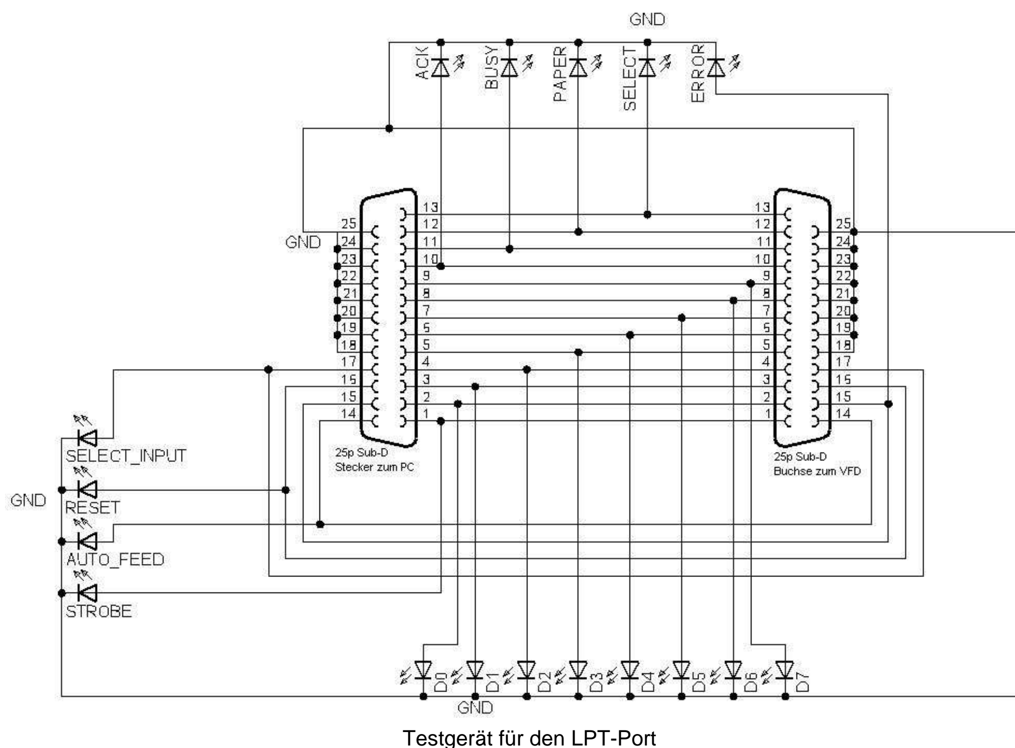
```
{Outport}
// Diese Assemblerfunktion schreibt einen Bytewert in die angegebene Adresse
//
// Name:                  outport
// Parameter:             Adresse: smallint - Adresse des Ports (z.B. $378)
//                        wert: byte        - Ausgabewert
// Rückgabewert:          -
procedure TVFD.outport(Adresse: smallint; wert: byte);    { Portausgabe }
begin
 asm
  push dx              // DX Register auf Spapel legen
  push ax              // AX Register auf Spapel legen
  mov dx, Adresse      // Adresswert in DX abspeichern
  mov al, Wert         // Wert in AL Register ablegen
  out dx, al           // Portausgabe
  pop ax               // AX von Stapel laden
  pop dx               // DX von Stapel laden
 end;
end;
```

Diese Prozedur beinhaltet einen Assemblerblock der mit dem Schlüsselwort **asm** beginnt und wie **begin** mit einem **end** enden muss.
Innerhalb dieses Blockes dürfen Assembleranweisungen stehen.
In diesem Block werden die Register AX und DX verwendet, daher ist es ratsam diese Register am Anfang mittels *push* auf den Stack zu legen und am Ende den Blockes via *pop* wieder von diesem zu nehmen.
Nach Sicherung der beiden Register wird in das DX-Register der Wert der Zieladresse aus dem ersten Parameter der Prozedur kopiert. Zu beachten ist, dass DX ein 16bit Register ist und der Parameter entsprechend ebenfalls ein Typ mit nicht mehr als 16bit sein darf.
Dann wird der Wert, der an der Adresse stehen soll, vom zweiten Prozedurparameter
In das AL-Register geladen. Man hätte ihn auch nach AX kopieren können, der Inhalt des AH-Registers ist für die Prozedur aber ohnehin nichtig.
Schließlich folgt der wichtigste Teil der ganzen Prozedur mit der out-Anweisung, welche der Zieladresse den Ausgabewert zuweist.
Die out-Anweisung benötigt Register als Parameter, weshalb es nicht möglich gewesen wäre zu schreiben „out adresse, wert“.

Auf diese Art auf Hardware zu zugreifen ist unter Betriebssystemen wie MS DOS die übliche Vorgehensweise.
Bei Multitasking-Betriebssystemen ist diese Vorgehensweise jedoch äußerst heikel.
Obige Methode funktioniert auf den DOS-basierten Windowsversionen Windows 95, 98, Me zwar noch, darf aber wegen dem direkten Zugriff auf Register statt der Verwendung von Treibern zurecht als unnötig kritisch eingestuft werden.
Und ab Windows NT ist direkter Hardwarezugriff ohnehin nur noch Treibern möglich.
Diese Prozedur würde dann eine Fehlermeldung „Privilegierte Instruktion“ verursachen und Windows würde den Zugriff verweigern.

6. Testgerät

Da die Methode unter Windows 98 jedoch funktioniert, wurde es zunächst dabei belassen und folgende Schaltung aufgebaut, die über LEDs die Low-/Highzustände der Leitungen des LPT Ports anzeigt:



Testgerät für den LPT-Port

Diese Schaltung diente nicht nur zum anfänglichen experimentieren mit dem LPT sondern später auch als nützliches Utensil zur Fehlersuche, da sie auch bei Betrieb mit dem Display in die Leitung eingeschleift werden kann.
Der Anschluss der LEDs ohne Vorwiderstand mag auf den ersten Blick Skepsis hervorrufen, verwendet man allerdings low-current LEDs, die der Spannung des LPT-Ports standhalten, darf man bezüglich der Lebensdauer der Leuchtdioden und des LPT-Controllers beruhigt sein.

Bild des Testgerätes

7. Kommunikation mit LPT- Treiber

Mit dem LPT-Port konnte nun also umgegangen werden.
Allerdings war die Beschränkung auf Windows 95/98 recht unbefriedigend.
Auf der Suche nach Lösungen fand sich das „300/800 Series Development Software – Kit“ der Fa. Noritake unter www.noritake-itron.com .
In diesem Software-Packet befindet sich auch der *Port Talk I/O Port Driver* von *Beyond Logic* (http://www.beyondlogic.org), welcher zusammen mit dem Programm *allowIO.exe* anderen Programmen Zugriffsrechte auf IO-Ports unter den Betriebssystemen Windows NT/2000/XP ermöglichen kann.
Die Vorgehensweise dazu ist folgende:
Benötigt werden die Dateien *allowIO.exe* und *Porttalk.sys* im gleichen Verzeichnis wie das Programm, welches Zugriff auf den/die Ports haben soll.
Dann ruft man *allowIO.exe* auf und übergibt als Parameter den Programmnamen und die IO-Adresse: a*llowIO <executable.exe> <Addresse(n)>*
Die Adresse(n) werden in der hexadezimalen 'C'-Darstellung angegeben (0x00).
Beispiel:

allowIO vfdstudio.exe 0x378
*g*ibt vfdstudio Zugriffsrechte auf LPT1.

Alternativ kann auch der Zugriff auf alle Adressen ermöglicht werden, wenn z.B. bei Programmstart noch nicht klar ist welche Adressen benötigt werden:

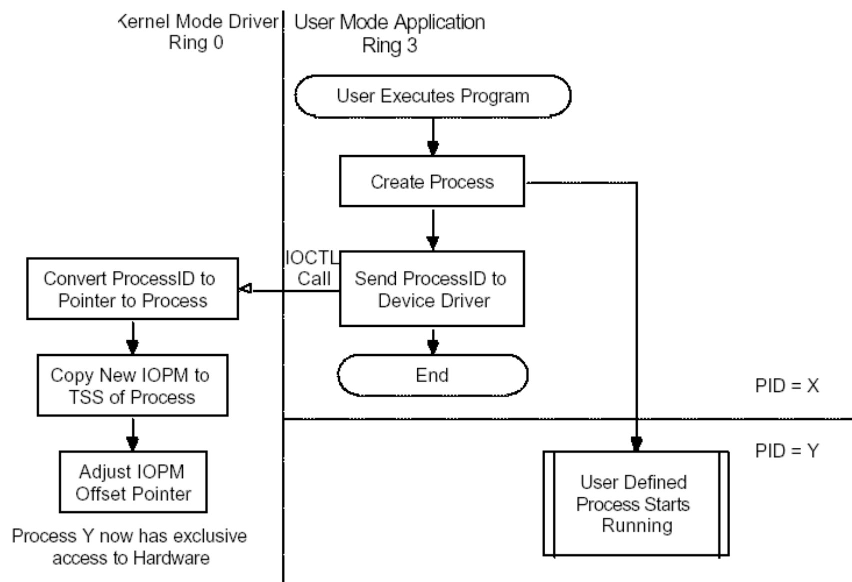
allowIO vfdstudio.exe /a

8. Zur Funktion von allowIO und PortTalk:

NT-basierte Windowsversion kennen zwei Modi für Zugriffsrechte auf Hardware:
Normale Benutzerprogramme laufen im ***ring3-Modus***, während der Kernel und Hardwaretreiber im ***ring0-Modus*** laufen.
Dadurch wird gewährleistet, dass Anwendersoftware keine Hardwarekonflikte verursachen können da sie mit einem Gerätetreiber kommunizieren müssen, welcher sich um den Hardwarezugriff kümmert.

Dass normale Programme unter Windows NT nicht direkt auf Hardwareregister zugreifen können liegt eigentlich jedoch nicht an Windows sondern vielmehr daran, dass der Prozessor im *protected mode* betrieben wird.
Ein solcher Prozessor wird, wenn auf eine IO-Register zugegriffen wird, zunächst prüfen, ob der Task die nötigen Privilegien hat. Diese stehen im sog. ***Task State Segment (TSS)***.
Sind die Pivilegien nicht vorhanden wird er prüfen ob im ***IO permission bitmap*** die Zugriffsrechte vermerkt sind. Trifft das zu wird er den IO-Befehl ausführen, andernfalls eine Exception auslösen.
Die IO permission bitmap ist eine Tabelle, die für jede IO-Adresse einen Eintrag besitzt, der entscheidet ob auf diese Adresse zugegriffen werden darf. Eine '1' bedeutet, dass kein Zugriff erlaubt ist; eine '0' bedeutet, dass Zugriff auf diese Adresse erlaubt wird.
Jeder Task besitzt natürlich seine eigene Tabelle.
Wenn allowIO ausgeführt wird, wird von diesem auch das eigentliche Programm gestartet.
Dabei wird Windows die ProcessID des neuen Programmes broadcasten, welche der porttalk-Treiber abfängt und anhand dieser ID die IO permission bitmap des neuen Programmes manipuliert.

Zur Veranschaulichung ein Auszug aus der Dokumentation des PortTalk-Treibers:



Zur detaillierten und tiefergehenden Erklärung der Vorgänge sei an dieser Stelle jedoch auf Sourcecode und Dokumentation des PortTalk-Treibers im oben genannten Noritake Development Software–Kit verwiesen.

Dieses Verfahren ist also sehr interessant, aber nicht ganz trivial und auch sicherheitstechnisch eher bedenklich.
Insbesondere letzterer Aspekt war es dann auch, der dazu bewog nicht den PortTalk-Treiber zu verwenden, sondern einen reinen LPT-Treiber.
Und da das VFD-Studio immer so entwickelt werden sollte, dass es später evtl. als Freeware verbreitet werden könnte, schied der PortTalk-Treiber somit aus.

9. Alternativen zu PortTalk

Beschäftig man die Google-Suchmaschine mit der Frage nach LPT-Treibern zusammen mit Delphi erhält man als eines der ersten Ergebnisse einen Verweis auf das **DLPortio-Packet** von John Pappas (http://diskdude.cjb.net).
Hierin enthalten ist neben dem ***DriverLINX*** LPT-Treiber von *Scientific Software Tools* praktischerweise auch gleich noch eine Komponente für Delphi und C-Builder, die die Kommunikation mit diesem Treiber übernimmt.
Der DriverLINX-Treiber ist übrigens ein sehr populärer Treiber, den viele Programme zur Displayansteuerung verwenden. So verwendet z.B. auch LCDInfo das DLPortIO-Packet und den DriverLINX-Treiber.

10. Benutzung der DLPortIO-Komponente

Neben der großen Verbreitung spricht auch die leichte Handhabung der Komponente für sich. Nach der Installation in die Delphi-Komponentenbibliothek erfolgt die Kommunikation mit dem LPT-Port einfach folgendermaßen:

- In die uses-Klausel der Fenster-Unit wird die Portio.pas aufgenommen:

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  PortIO, StdCtrls;
```

- Bei Programmstart muss zuerst der Treiber geöffnet werden:

```
//------------------------------------------------------------------------
// Constructor for TMain_Win
//------------------------------------------------------------------------
constructor TMain_Win.Create(Owner : TComponent);
begin
   inherited Create(Owner);
   // Driver is in the same directory as the demo.exe file!
   DLPortIO.DriverPath:=ExtractFileDir(ParamStr(0));
   // Open the DriverLINX driver
   DLPortIO.OpenDriver();
end;
```

(Beispielcode)

Und bei Programmende mit *CloseDriver* entsprechend geschlossen werden.

- Die eigentliche Datenausgabe über den LPT erfolgt dann mittels

`DLPortIO.Port[Adresse]:=Daten;`

wobei `Adresse` die LPT-Adresse angibt (z.B. `'$378'`) und `Daten` den zu schreibenden Wert (`$00..$FF`).

Fazit

Die DLPortIO-Komponente und der DriverLINX-Treiber haben sich während der Entwicklung des VFD-Studios bewährt und wurden inzwischen auch in einem anderen Projekt (CPULast-Anzeige mit Drehspulinstrument an LPT-Port) verwendet.