Steuern und Messen über den Parallelport unter Delphi
Version 1.0 - 05.07.2002

Dieses Tutorial ist nur für Windows 95 / 98 (SE) geeignet, da unter Windows NT das Ansprechen der Ports ein „priviligierter Zugriff“ ist, der nur Treibern vorbehalten ist. Solche Treiber kann man im Internet leicht finden und die in ihnen bereitgestellten Routinen sehr leicht nutzen. Wenn man dieses Tutorial gelesen hat, sollte man auch leicht damit zurechtkommen.

Ich setze einige Grundlagen der Delphi-Programmierung und Digitaltechnik voraus, werde aber trotzdem eine kleine Einführung geben. Ein paar Assemblerkenntnisse können nicht schaden, sind aber nicht unbedingt notwendig.

Der Parallelport oder LPT-Port, an den ältere Drucker angeschlossen werden, wird offiziell „25-polige Centronics-Schnittstelle“ genannt. Von diesen 25 Leitungen stehen in der ursprünglichen von IBM spezifizierten Schnittstellenversion 8 Ausgänge und 5 Eingänge zur Verfügung. Diese kann man alle getrennt ansteuern bzw. auslesen.
Um diese Leitungen (Pins) ansprechen zu können, muss man die Portadresse des Parallelportes kennen. In den meisten Fällen ist sie $378 (das Dollarzeichen bedeute in Delphi, dass es sich um eine Hexadezimal Zahl handelt). Diese Portadresse ist im Grunde nur die Basisadresse von 2 weiteren. Unter der Basisadresse kann man auf die 8 Ausgabepins zugreifen, diese Leitungen sind die Datenleitung. Die Basisadresse plus 1 bietet uns 8 Einlesepins, von denen wir nur 5 ohne Probleme benutzen können. Ursprünglich wurde auf diesen Leitungen der Status des Druckers abgefragt. Die Basisadresse plus 2 ist zur Fehlerkorrektur und soll uns hier nicht interessieren.
Wie oben schon gesagt gilt diese Aufschlüsselung der Leitungen nur für den „normal Mode“ (auch SPP oder Standard) des Parallelports.
Die Basisportadresse und der Modus können im BIOS eingestellt und ausgelesen werden, meistens unter „Integrated Peripherals“ oder „Chipset Feature Setup“. Dies sieht etwa wie folgt aus:

...
```
Onboard Parallel Port 378/IRQ7
Onboard Parallel Mode normal
```
...
Weitere Informationen stehen im Mainboard-Handbuch.
Die Adresse des Parallelports kann man auch direkt im Delphiprogramm aus dem BIOS auslesen, zu finden ist sie unter den in der Tabelle aufgeführten Adresse.
Der Offset der Adressen beträgt $0040.

Adresse	**Port**
0008	Port-Adresse LPT1
000A	Port-Adresse LPT2
000C	Port-Adresse LPT3
000E	Port-Adresse LPT4 / nicht verwendet

Unter reinen Windowsprogrammen (wie Delphi es in den neueren Versionen ist) ist dies aber sehr kompliziert, da direkte Speicherzugriffe normalerweise nicht möglich sind, dies liegt am Multitasking Konzept von Windows. Unter Turbo-Pascal oder Delphi 1 wäre das sehr leicht über die *Mem* Funktion möglich. Im professionellen Bereich wird Windows zur Steuerung nur sehr selten eingesetzt, da es nur sehr beschränkt echtzeitfähig ist – aber für Hobby-Bastler ist es neben DOS die erste Wahl.

Um nun Daten aus dem Parallelport auszugeben, gibt es den Assemblerbefehl out.

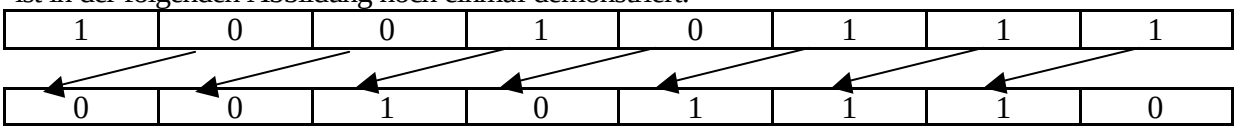
Diese Befehl benötigt als Parameter die Nummer des Ports (wie oben angegeben) und natürlich die Daten über die Zustände der Pins die ausgeben werden sollen.
Beide Parameter müssen sich in Registern (dem „Arbeitsspeicher“ des Prozessors) befinden. Dahin bekommt man sie mit dem *mov*(e) Befehl.
Hier mal eine Prozedur zum Ausgeben von Daten:

```
// Signal ausgeben
procedure WritePort(Port: word; Daten: byte);
begin
     asm // Befehle für Inlineassembler folgen
        MOV AL,Daten // Variable Daten in AL-Register bewegen
        MOV DX,Port // Variable Port in DX-Register bewegen
        OUT DX,AL // Ausgaben der Daten(AL) auf dem Port(DX)
     end;
end;
```

Mit dem Einlesen von Daten ist es sehr ähnlich, nur das der Befehl dort *in* heißt.

```
// Signal einlesen
function ReadPort(Port: word): byte;
begin
     asm // Befehle für Inlineassembler folgen
        MOV DX,Port // Variable Port in DX-Register bewegen
        IN  AL,DX // Einlesen der Daten(AL) vom dem Port(DX)
        MOV Result, AL // Daten(AL) in die Rückgabevariable
                       // bewegen
     end;
end;
```

Später müssen wir beim Aufruf der *Readport*-Funktion darauf achten das die Portadresse um 1 höher als die Basisadresse sein muss. Mit diesen beiden Funktionen haben wir nun das Grundwerkzeug zum Einlesen und Ausgeben von Daten auf jedem beliebigen Port und speziell für den LPT-Port.
Jeder Pin ist entweder 1 (High=Strom) oder 0 (Low=kein Strom). Es gibt maximal 8 Leitungen mit je 2 Zuständen. Diese Zustände werden als 0 oder 1 hintereinander geschrieben und ergeben somit eine 8 Bit Binärzahl.
Für die Ausgaben müssen wir nun die 8 Zustände der Pins in eine Byte-Variable (8*1 Bit) verschlüsseln. Das geht am besten über *Shift*-Befehle, sowie die logischen *or* und *and* Befehle. So ist es möglich die Variable bitweise zu bearbeiten.
Wir benötigen die zwei Shift-Befehle *Shl* und *Shr* (*Shl* für die Ausgabe und *Shr* für das Einlesen). *Shl* (Shift left) schiebt die Variable, die aus 8 Bit besteht um eins nach links, wobei das erste (linkste) Bit verschwindet und das letzte (rechtste) Bit leer (null) wird. Das Prinzip ist in der folgenden Abbildung noch einmal demonstriert:



Shr (Shift right) macht das Gleiche nur in entgegengesetzte Richtung, also von links nach rechts.

a	b	a and b
0	0	0
0	1	0
1	0	0
1	1	1

Mit dem *and* Befehl verknüpft man zwei Bits zu einem Ergebnis. Dabei ist das Ergebnis positiv (1), wenn beide Argumente auch positiv sind, ansonsten ist das Ergebnis negativ (0). Verwendete Zeichen für die *and*-Operation sind: $\cap$ oder das eigentlich korrektere $\wedge$.

a	b	a or b
0	0	0
0	1	1
1	0	1
1	1	1

Bei *or* Operation reicht schon ein positives Argument aus, um ein positives Ergebnis zu erhalten. Oft werden auch folgende Zeichen gebraucht: $\cup$ oder das zutreffendere $\vee$.

Wenn man genaueres über die Operationen *and* und *or* wissen will, sollte man sich über „Boolesche Algebra“ informieren, da es noch viele andere Operatoren und Gesetze, sowie logische Zusammenhänge gibt.

Da man Bits (auf normalen PC-Prozessoren) nicht so einfach direkt setzten kann, gehen wir Bit für Bit vor.
Um die Schaltzustände der Pins genau definieren zukönnen, habe ich die Funktion *EditPort* geschrieben, dort wird mit Hilfe des *or*-Operators das letzte Bit gesetzt und danach mit *Shl* nach links geschoben. Dies wird solange wiederholt bis alle Bits den richtigen Zustand haben.

0	0	0	0	0	0	0	0
or							
0	0	0	0	0	0	0	1
=							
0	0	0	0	0	0	0	1
shl							
0	0	0	0	0	0	1	0

Die folgende Funktion führt genau das eben beschriebene aus.

```
// Umwandeln in Ausgabesignal
function EditOut(l0,l1,l2,l3,l4,l5,l6,l7: boolean): byte;
begin
     result := 0;
     if l7 then result := (result or 1);
     result := result shl 1;
     if l6 then result := (result or 1);
     result := result shl 1;
     if l5 then result := (result or 1);
     result := result shl 1;
     if l4 then result := (result or 1);
     result := result shl 1;
     if l3 then result := (result or 1);
     result := result shl 1;
     if l2 then result := (result or 1);
     result := result shl 1;
     if l1 then result := (result or 1);
     result := result shl 1;
     if l0 then result := (result or 1);
end;
```

Man könnt die Bits natürlich auch direkt setzen, indem man Binärzahlen 10000000, 01000000, usw. direkt mit *or* verknüpft.

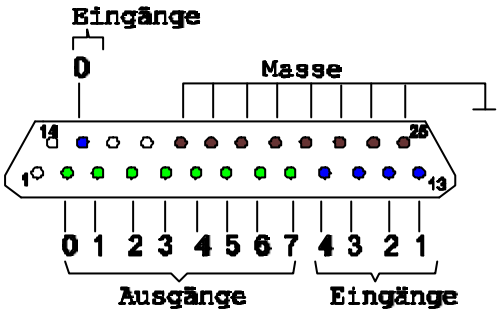
```
// Umwandeln in Ausgabesignal
function EditOut(l0,l1,l2,l3,l4,l5,l6,l7: boolean): byte;
begin
     result := 0;
     if l7 then result := (result or 1);
     if l6 then result := (result or 2);
```

```
     if l5 then result := (result or 4);
     if l4 then result := (result or 8);
     if l3 then result := (result or 16);
     if l2 then result := (result or 32);
     if l1 then result := (result or 64);
     if l0 then result := (result or 128);
end;
```

Zum Dekodieren, also beim Einlesen, muss dies genau anders herum geschehen. Hier wird auch Bit für Bit getestet, nur die ersten 3 Bit sind, da man ja nur 5 Bit nutzen kann uninteressant und werden daher gleich „rausgeshiftet“. Nicht wundern braucht man sich über den Tausch von l3 und l4, da diese beiden Pins ja ursprünglich als acknoledge und busy des Druckers geplant waren und somit nicht der „logischen“ Reihenfolge der Pins entsprechen. Eine weiter Besonderheit des Pins 11 also l3 (busy) ist, das er negiert ist, also bei logischer 1 eine 0 liefert und umgedreht.
In der folgenden Einlese-Funktion sind diese Unnormalitäten schon beseitigt.

```
// Dekodieren des Ausgabesignals
procedure EditIn(data: byte; var l0,l1,l2,l3,l4: boolean);
begin
     l0:=false;
     l1:=false;
     l2:=false;
     l3:=false;
     l4:=false;
     data := data shr 3; //die ersten 3 Statuspins sind nicht
                         //benutzbar
     if (data or 1) <> data then l0 := true;
     data := data shr 1;
     if (data or 1) <> data then l1 := true;
     data := data shr 1;
     if (data or 1) <> data then l2 := true;
     data := data shr 1;
     if (data or 1) <> data then l4 := true;
     data := data shr 1;
     if (data or 1) = data  then l3 := true; //Busy-Pin wird
                                             //negiert
end;
```

Bei der Ausgabe-Funktionen steht l0 für Pin 0 und so weiter bis l7 für Pin 7, wo die Pins zu finden sind, besprechen wir weiter unten, wenn wir uns mit einigen Beispielschaltungen beschäftigen. Bei der Einlese-Funktion habe ich auch der Zeichnung entsprechend, die Pins l0 bis l4 benannt. Es kann gut sein, dass ihr sie in anderen Texten als s3 bis s7 benannt findet, da eben die ersten 3 Leitungen nicht zur Verfügung stehen, s steht dabei für Status. Leitungen die mit dem Buchstaben d beginnen sind die Data-Leitungen, also zur Ausgabe.
Nun kommen wir zum Hardware-Teil und da möchte man natürlich zu erst wissen, welche Pins des Parallelportes welche Datenleitung darstellen und welche Leistung bzw. Spannungen zu nutzen sind.

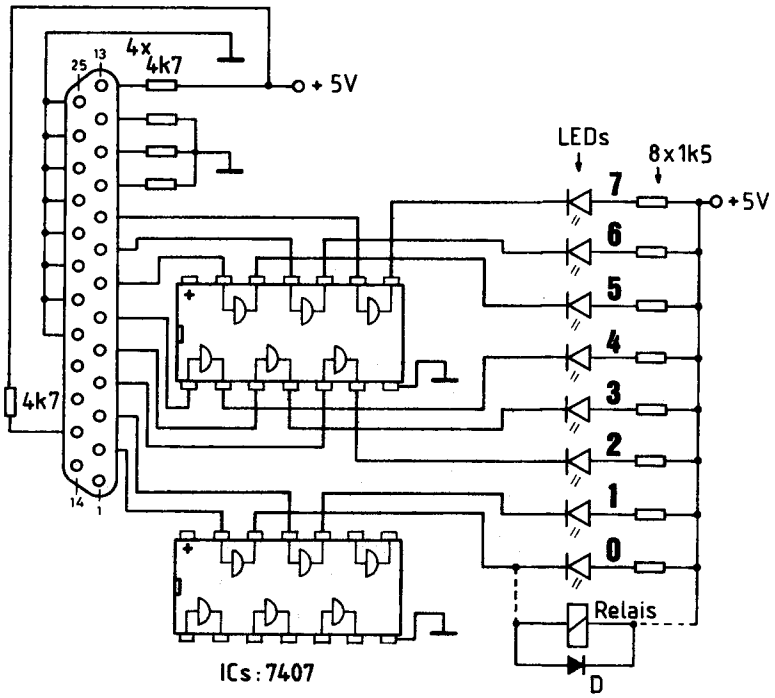


Ausgabe	l0	l1	l2	l3	l4	l5	l6	l7
HW-Pin	2	3	4	5	6	7	8	9

Einlesen	l0	l1	l2	l3	l4
HW-Pin	15	13	12	11	10
Bezeichnung	Error	Select	Paper empty	Busy	Ack-now-ledge

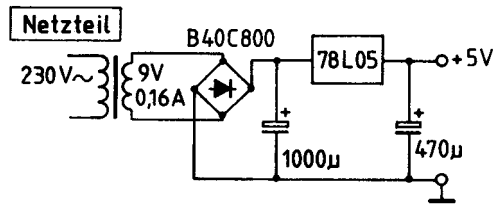
Für einen schnellen Einlesetest sollte man die Pins nicht einfach mit Masse kurzschließen (wenn einem sein Parallelport lieb ist), sondern sie mit einem 350 - 500 Ω Widerstand verbinden. Für einen Ausgabetest sollte man eine einfache LED nutzen.

Grundsätzlich ist der Parallelport TTL kompatibel. TTL ist eine alte, aber immer noch recht verbreitete Logikgatterfamilie, die man in jedem Elektronikhandel leicht bekommt. Erkennen kann man Standard-TTL-Gatter an ihrer Nummer, sie beginnt immer mit 74. Häufig stehen auch Buchstaben in der ID-Nummer; etwa LS diese Bausteine sind nicht immer ohne weiteres nutzbar. Mit den Schaltkreisnummern, die man in den Zeichnungen deutlich erkennt, kann man die Bausteine bequem bestellen.
Die TTL Kompatibilität bedeutet, das der High-Zustand 5V ist und der Low-Zusand bei nahezu 0V. Theoretisch könnte man direkt an den LPT-Port TTL-Gatter anlöten und alles müsste funktionieren, aber da der LPT-Port von Rechner zu Rechner verschiedene Leistungen bereit stellt bzw. mal bessere und mal schlechtere Qualität hat, sollte man direkt hinter den Parallelport eine Treiberbaustein schalten. Ein solcher Baustein wäre das TTL-Treiber IC 7407.
Als erstes werde ich mal eine kleine Lichtorgelschaltung hier aufzeichnen. Es ist eigentlich immer das erste was man so zum Testen baut, wenn man mit dem Parallelport schalten will:



An Stelle der LEDs kann man natürlich auch Relais schalten und hat durch den IC 7407 den Vorteil, dass bei einer Überlastung nicht der Parallelport, sondern das IC kaputt geht (der IC kostet so um 0,50€ der Parallelport mindestens 25€). Mit diesen Relais kann man dann Motoren mit hoher Leistung oder auch nur Glühbirnen schalten. Um die ICs mit Strom zu

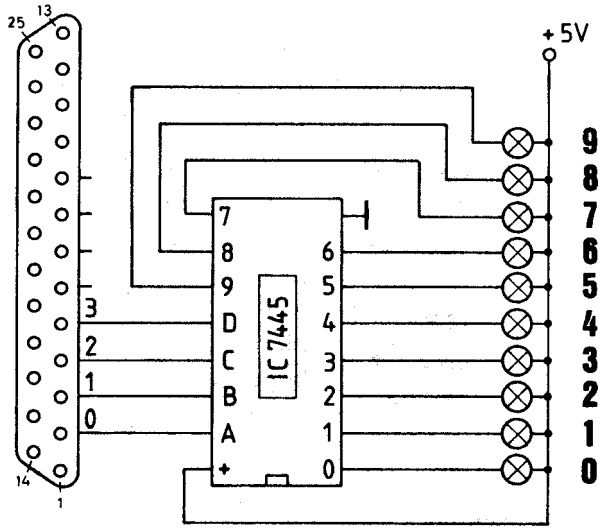
versorgen ist hier ein Spannungsregler aufgezeichnet. Als Netzteil kann man jedes normale 9V Netzteil nehmen, wie sie häufig für Discmans und ähnliches im Haushalt rumliegen.



Wenn jemanden die 8 Datenleitungen nicht reichen sollten, so kann er auch einen Decoder hinter den Port schalten. So kann man z.B. mit nur 4 Datenleitungen aus dem LPT-Port 10 Lampen schalten. Dazu könnte man einen BCD-Decoder nutzen. Wie dies funktioniert zeigt diese Tabelle:

Eingänge				**Ausgänge**									
A	**B**	**C**	**D**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	0	0	1	1	0	1	1	1	1	1	1	1	1
0	0	1	0	1	1	0	1	1	1	1	1	1	1
0	0	1	1	1	1	1	0	1	1	1	1	1	1
0	1	0	0	1	1	1	1	0	1	1	1	1	1
0	1	1	0	1	1	1	1	1	0	1	1	1	1
0	1	1	1	1	1	1	1	1	1	0	1	1	1
1	0	0	0	1	1	1	1	1	1	1	0	1	1
1	0	0	1	1	1	1	1	1	1	1	1	0	1
1	0	1	0	1	1	1	1	1	1	1	1	1	0

Es irritiert vielleicht etwas das die Ausgänge auf Low liegen, obwohl sie logisch auf High liegen müssten. Dies kann man mit einem kleinen „Schaltungstrick“ umgehen, oder man baut hinter diesen BCD-Decoder einen Inverter. Diese „aus 4 mach 10“ Schaltung wurde zum Beispiel in der folgenden Schaltung eingesetzt. Die dort freien Drucker-Port-Pins werden genauso wie oben beschaltet.



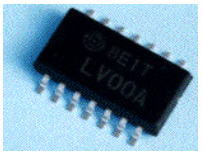
Wenn man einen solchen Decoder einsetzt oder auch mehrere davon, so kann man einfach die Funktion *EditOut* umschreiben und schon kann man wieder leicht Programme erstellen ohne die komplizierte Decodierung für jede Ausgabe überdenken zu müssen.

Dieses Tutorial ist natürlich nur eine Einführung. Es gibt aber leider zu diesem Thema unter Delphi bei Windows nur sehr wenig Bücher. Trotzdem sollte man sich nicht entmutigen

lassen und immer wieder mal was Neues bauen. Mit dem LPT-Port lässt sich sehr viel machen, das Spektrum reicht dort von Lichtorgeln und Ampelsteuerungen bis zu AD/DA-Wandler Anwendungen. Falls ihr weitere Informationen im Internet sucht, solltet ihr darauf achten in welchem Modus der Parallelport zu betreiben ist (SPP, EPP oder ECP).
Die hier beschriebenen Schaltungen sind für den normalen Umgang gedacht, also nicht bei über 100°C betreiben ;-)

Kleiner Glossar

TTL (Transistor-Transistor-Logik) = spezielle Baureihe von IC mit einheitlichen Kenndaten, daher kompatibel, es gibt die Baureihen: L-TTL, Standard-TTL, LS-TTL, ALS-TTL, S-TTL, AS-TTL die sich Verlustleistung und Durchlaufverzögerung unterscheiden (man kann nicht alle Baureihen mit einander kombinieren !)
LS-TTL (Low-Power-Schottky Transistor-Transistor-Logik) = spezielle TTL-Baureihe mit geringerer Verlustleistung bei geringer Durchlaufverzögerung
Gatter = Logischer Baustein zur Realisierung boolescher Logik (z.B.: or, and, xor, ...) in der Digitaltechnik, meistens werden mehrere Gatter in einem IC zusammengefasst
TTL-Treiber = Verstärkerbaustein in der TTL-Technik z.B.: 7407
LED (Light Emiting Diode) = eine Lichterzeugende Diode, benötigen wesentlich weniger Leistung als Glühbirnen
Relais = elektronischer Schalter, beim Anlegen einer Spannung wird ein elektromagnetisches Feld ein Schalter geschlossen
IC (Integrierter Schaltkreis) = miniaturisierter Schaltkreis der in meist in Plastik gegossen und mit Anschlussbeinchen bestückt worden; es gibt mehrere IC-Bauformen, normaler Weise DIL (Dual Inline Gehäuse), es gibt auch Pingrid, Flatpack, SOJ, PLCC und andere



AD-Wandler = Baustein der einen analogen Wert in eine Binärzahlumwandelt, es gibt verschieden Verfahren und Genauigkeiten bei AD-Wandlern z.B.: MAX191
DA-Wandler = wandelt digitale Werte in ein analog Signal (siehe AD-Wandler) z.B.: TLC7524
BCD-Decoder = Schaltung die eine BCD-Zahl in eine andere Codierung umwandelt z.B.: 7445