Guido Dampf

Grafik mit dem 7220 von NEC

Direkter Zugriff in Turbo-Pascal

Der Grafikprozessor 7220 von NEC ist ein sehr mächtiger Chip und wird bei vielen Computern eingesetzt, zum Beispiel beim Epson QX-10, QX-16 und beim mc-CP/M-Plus-Computer. Auch für den Computer-Selbstbauer ist er, nicht zuletzt durch den rapiden Preisverfall bei Halbleitern, ein attraktiver Prozessor. Der 7220 stellt eine Menge mächtiger Grafikbefehle zur Verfügung. Wenn man allerdings das NEC-Datenblatt zu diesem Baustein durchliest, kann man schnell den Mut verlieren. Es ist nicht gerade leicht verständlich. Daher beschreiben wir die wichtigsten Befehle. Der Autor ist Mitglied des WDR-Computerclubs.

Zum Datenaustausch zwischen der CPU und dem 7220 werden zwei I/O-Kanäle benötigt, die durch eine Adreßleitung und die Read/Write-Leitung ausgewählt werden. Ist der 7220 im Speicherbereich eingebunden, so belegt er zwei Adressen. Nachfolgend seien sie mit A0 und A1 (z. B. beim QX-10 sind A0 und A1 die I/O-Kanäle $38 und $39) bezeichnet. Liest man von A0 ein Byte ein, so erhält man die Statusinformationen des 7220. Wenn man von A1 ein Byte einliest, so erhält man einen Wert aus der internen Warteschlange (W.S.). Ein solcher Wert wird vom 7220 bei bestimmten Befehlen in die W.S. eingetragen. Da die W.S. jedoch auch als Puffer für die Befehle und deren Parameter dient, sollte darauf geachtet werden, daß bei einem Befehl, der einen Wert zurückliefert, alle sich noch in der W.S. befindlichen Befehle gelöscht werden. Die W.S. steht dann nur für die Ausgabe der Werte zur Verfügung. Wird ein neuer Befehl an den 7220 geschickt, bevor alle Werte, die ein vorhergehender Befehl in die W.S. hineingeschrieben hat, ausgelesen wurden, so gehen diese Werte ebenfalls verloren. Durch Überprüfung des Statusregisters lassen sich ungewollte Effekte dieser Art vermeiden.

Es ist weiterhin zu beachten, daß Befehle an den 7220 durch Schreiben in A1, die dazugehörigen Parameter aber durch Schreiben in A0 in die Warteschlange gesetzt werden. Bei einigen Befehlen ist es auch möglich, Parameter wegzulassen. Da die Zuordnung der einzelnen Werte zu den Parametern jedoch von der Reihenfolge abhängt, kann man nur Parameter weglassen, die sich am Ende der Liste der möglichen Parameter befinden. In diesem Fall beginnt die Ausführung des Befehls, sobald der 7220 das nächste Kommando aus der Warteschlange erkennt.

```
procedure putcmd(b : byte);

begin
  while (port(.A0.) and 2) <> 0 do;
  port(.A1.) := b
end;

procedure putval(b : byte);

begin
  while (port(.A0.) and 2) <> 0 do;
  port(.A0.) := b
end;

procedure putdval(i : integer);

begin
  while (port(.A0.) and 2) <> 0 do;
  port(.A0.) := lo(i);
  while (port(.A0.) and 2) <> 0 do;
  port(.A0.) := hi(i)
end;

function inpval : byte;

begin
  while (port(.A0.) and 1) = 0 do;
  inpval := port(.A1.)
end;
```

Bild 1. Mit diesen Funktionen und Prozeduren kann man den 7220 über ein Port-Array (A0,A1) direkt ansprechen

```
function lightpendetect : boolean;

begin
  lightpendetect := (port(.A0.) and $80) <> 0
end;

function horiblank : boolean;

begin
  horiblank := (port(.A0.) and $40) <> 0
end;

function vertblank : boolean;

begin
  vertblank := (port(.A0.) and $20) <> 0
end;

function isdrawing : boolean;

begin
  isdrawing := (port(.A0.) and 8) <> 0
end;

function isready : boolean;

begin
  isready := (port(.A0.) and 4) <> 0
end;

function dmaccess : boolean;

begin
  dmaccess := (port(.A0.) and $10) <> 0
end;
```

Der 7220 enthält die komplette Videologik zur Ansteuerung eines Monochrom- oder Farbbildschirmes. Auf die Farbtauglichkeit sei hier allerdings nicht näher eingegangen. Ich verweise in diesem Zusammenhang auf die Datenblätter von NEC (wenn's auch schwerfällt).

Das Statusregister

Die einzelnen Bits des Statusbytes enthalten Informationen, die zur sicheren Kommunikation mit dem 7220 unbedingt erforderlich sind.
Bit 0 zeigt an, ob der 7220 Daten für die CPU in der Warteschlange bereit hält (Bit gesetzt). Vor dem Lesen aus der Warteschlange sollte dieses Bit unbedingt geprüft werden.
Bit 1 zeigt an, ob die Warteschlange voll ist. Sollte dieses Bit gesetzt sein, so dürfen keine weiteren Daten an den 7220 gesendet werden.
Bit 2 ist gesetzt, wenn sämtliche Befehle aus der Warteschlange abgearbeitet wurden.
Bit 3 ist gesetzt, wenn der 7220 gerade mit der Ausführung eines Befehls beschäftigt ist. Die Warteschlange ist jedoch auch dann noch aufnahmebereit.
Bit 4 ist während der DMA-Zugriffe gesetzt.
Bit 5 zeigt im gesetzten Zustand an, daß der 7220 gerade den Kathodenstrahl der Bildröhre abgeschaltet hat, damit man dessen Rückführung in die linke obere Ecke des Bildschirms nicht sieht. Läßt man Bildspeicherzugriffe (spez. DMA-Zugriffe) außerhalb dieser Zeit zu, so kann es zu kleinen Unsauberkeiten des Bildes kommen. Während der Dunkelphase ist das jedoch ausgeschlossen.
Bit 6: Der Kathodenstrahl der Bildröhre wird nicht nur während seiner vertikalen Rückführung ausgeschaltet, sondern auch während der horizontalen Rückführung auf den Beginn einer neuen Bildschirmzeile. Diese Dunkelphase ist jedoch kürzer als die vertikale Rückführung. Während dieser Phase ist Bit 6 gesetzt.
Bit 7 dient als Strobe-Signal, falls man einen Licht-Griffel angeschlossen hat. Ist es gesetzt, so hat der 7220 die Position des Licht-Griffels erkannt, und man kann sie über einen entsprechenden Befehl abfragen.

Der Bildwiederholspeicher und das Parameter-RAM

Der Bildwiederholspeicher (Video-RAM) ist bis zu 256 K Worte groß und in Worten zu 16 Bit organisiert. Die genaue Größe und Organisation muß beim Re-

```
procedure gdcreset;

begin
  putcmd(0);
(* Eigentlicher Reset-Befehl, der 7220 geht in den Leerlauf. *)
(* Hier  können noch Hardware-Parameter stehen (siehe sync Befehl)
   *)
  putcmd($6F)
(* Vsync-Master-Befehl  :  der  7220 erzeugt selbst den     Vsync-
   Impuls.  Eine Alternative wäre : $6E = Slave-Modus, d.h. Zuführung von außen. *)
end;
```

Bild 2. Ein Beispiel (Reset)

```
type   dmtyp = (MIXED,GRAPHIC,CHARACTER);
     modetyp = (REPLACE,COMPLEMENT,PRESET,PSET);
      figtyp = (CHARANDDOT,STRAIGHTLINE,GRAPHICSCHAR,
                CIRC,RECT,SLANTED);

const GDCCONST : array(.figtyp.) of byte = (0,8,$10,$20,$40,$90);
      xmp1 = 640; ymp1 = 400; (* Auflösung z.B. QX-10 *)
      xmax = 639; ymax = 399; (* Auflösung - 1 *)
      gdcparam : record (* Alle Parameter sollten nur über ihre
                           proceduren geändert werden *)
        displaymode : dmtyp;
        drawwhiledisp, screenenable,
        displaycursor, blinkcursor : boolean;
        dispzoom, grzoom, blinkrate, curtop, curbot : byte;
        mode : modetyp;
        pattern : integer;
        dynamicram,interlaced,repeatfieldchar : boolean;
        linesperchar,pitchvalue,wordsperline : byte;
        activdispword,activlines : integer;
        horisyncwidth,vertsyncwidth,horfrontporch,horbackporch,
        verfrontporch,verbackporch : byte;
        vsyncmaster : boolean
      end = (displaymode    : MIXED;    (* Text & Grafik-modus *)
             drawwhiledisp  : FALSE;    (* Nur bei ausgeschaltetem
                                           Kathodenstrahl malen *)
             screenenable   : TRUE;     (* Anzeige an *)
             displaycursor  : TRUE;     (* Cursor an *)
             blinkcursor    : TRUE;     (* Cursor blinkt *)
             dispzoom       : 0;        (* Kein Text-Zoom Faktor *)
             grzoom         : 0;        (* Kein Grafik-Zoom *)
             blinkrate      : 1;        (* Blinkgeschwindigkeit *)
             curtop         : 0;        (* Obere Kannte/Cursor *)
             curbot         : 15;       (* Untere Kannte/Cursor *)
             mode           : REPLACE;  (* Replace-Modus *)
             pattern        : $FFFF;    (* Pattern-Register *)
                                        (* Ab hier ist's Hardware-
                                           abhängig : *)
             dynamicram     : TRUE;     (* Dynamische / statische
                                           Rams *)
             interlaced     : FALSE;    (* kein interlaced-Modus
                                           beim Bildaufbau. *)
             repeatfieldchar : FALSE;   (* nur bei interlaced : TRUE
                                           wenn im Zeichen-Modus das
                                           gleiche Bild beim zweiten
                                           Halbbild wiederholt wer-
                                           den soll. *)
             linesperchar   : 16;       (* Zeilenzahl pro Zeichen-
                                           Zeile. *)
             pitchvalue     : 80;       (* Differenz zw. den Adres-
                                           sen zweier Zeichen, die
                                           direkt untereinander
                                           stehen. *)
             wordsperline   : 40;       (* Differenz zw. den Adres-
                                           sen zweier Punkte (unter-
                                           einander). Ist im Mixed-
                                           Modus die Hälfte von
                                           pitchvalue, sonst gleich
                                           pitchvalue *)
             activdispword  : 80;       (* Breite des Schirm's. Nor-
                                           malerweise gleich pitch-
                                           value *)
             activlines     : 400;      (* Höhe des Schirm's in Zei-
                                           len *)
             horisyncwidth  : 3;        (* Folgende Parameter wer-*)
             vertsyncwidth  : 16;       (* den benötigt,den Schirm*)
             horfrontporch  : 7;        (* zu synchronisieren (Er-*)
             horbackporch   : 19;       (* fahrungswerte, können *)
             verfrontporch  : 1;        (* je nach Hardware leicht*)
             verbackporch   : 4;        (* differieren) *)
             vsyncmaster    : TRUE);    (* 7220 generiert sein Vsync
                                           selber *)
```

Bild 3. Der Voreinstellungs-record

```
procedure sync(displaymode : dmtyp; drawwhiledisp,screenenable
                : boolean);    (* Setze Hardware-abh. Parameter *)

var b : byte;

begin
  gdcparam.displaymode := displaymode;
  gdcparam.drawwhiledisp := drawwhiledisp;
  gdcparam.screenenable := screenenable;
  with gdcparam do begin
    if screenenable then putcmd($F)
                    else putcmd($E);
    if dynamicram then b := 4 else b := 0;
    if displaymode = CHARACTER then b := b or $20
    else if displaymode = GRAPHIC then b := b or 2;
    if interlaced then begin
      b := b or 8;
      if not repeatfieldchar then b := b or 1
    end;
    if not drawwhiledisp then b := b or $10;
    putval(b); putval(activdispword - 2);
    putval((vertsyncwidth shl 5) or (horisyncwidth - 1));
    putval(((horfrontporch - 1) shl 2) or (vertsyncwidth shr 3));
    putval(horbackporch - 1);
    putval(verfrontporch);
    putval(activlines);
    putval((verbackporch shl 2) or (activlines shr 8));
    repeat until isready
  end
end;

procedure start;      (* Starte Display & Beende Leerlauf *)

begin
  putcmd($6B)
end;

procedure bctrl(screenenable : boolean);

(* Ein/Ausschalten des Schirms *)

begin
  gdcparam.screenenable := screenenable;
  if screenenable then putcmd($D)
                  else putcmd($C)
end;

procedure zoom(dispzoom,grzoom : byte); (* Setze Zoom-Faktoren *)

begin
  gdcparam.dispzoom := dispzoom; (* Faktor für Zeichen *)
  gdcparam.grzoom   := grzoom;   (* Faktor für Grafik *)
  putcmd($46);
  putval((dispzoom shl 4) or (grzoom and $F))
end;

procedure cchar(displaycursor, blinkcursor : boolean;
                blinkrate, curtop, curbot : byte);

(* Setze Cursor & Zeichen Characterisierung *)
```

Bild 4. Die 7220-Befehle, aus den Prozeduren in Bild 1 aufgebaut

```
var b : byte;
begin
  gdcparam.displaycursor := displaycursor;
  gdcparam.blinkcursor   := blinkcursor;
  gdcparam.blinkrate     := blinkrate;
  gdcparam.curtop        := curtop;
  gdcparam.curbot        := curbot;
  putcmd($4B);
  if displaycursor then putval($80 + gdcparam.linesperchar - 1)
                   else putval(gdcparam.linesperchar - 1);
  b := (curtop and $1F) or (blinkrate shl 6);
  if blinkcursor then putval(b)
                 else putval(b or $20);
  putval((curbot shl 3) or (blinkrate shr 2))
end;
procedure ccurs(i : integer);

(* Cursor-Positionierung  über  Adresse (EAd - Reg.) *)

begin
  putcmd($49);
  putdval(i)
end;

procedure gcurs(x,y : integer);

(* Cursor-positionierung  über Koordinaten für Grafik (EAd & dAd)-
   Reg.,  setzt alle Mask-Reg.-Bits bis auf eines auf 0  (abhängig
   von x-Koordinate (dAd)) *)

begin
  ccurs(y * gdcparam.wordsperline + x shr 4);
  putval((x and $F) shl 4)
  (* Falls mehr als 64 KW adressiert werden,
     sollte stattdessen diese Zeile Einsatz finden :
     putval((x and $F) shl 4 + trunc((int(y) *
                  gdcparam.wordsperline + x shr 4) / 65536.0)) *)
end;

procedure pram(adr : byte);

(* Bereitet das Schreiben in den Pram vor. Die folgenden Parameter
   (durch putval zu übergeben) werden ab Pram-adr eingetragen *)

begin
  putcmd($70 or adr)
end;

procedure pitch(pitchvalue : byte);

(* Setze  Pitchvalue,  also die Anzahl der angezeigten Wörter  pro
   Zeile  (Im Mixed-Modus orientiert sich Pitchvalue an der Anzahl
   der Zeichen pro Zeile) *)

begin
  gdcparam.pitchvalue   := pitchvalue;
  if gdcparam.displaymode = MIXED then
       gdcparam.wordsperline := pitchvalue div 2
  else gdcparam.wordsperline := pitchvalue;
  putcmd($47);
  putval(pitchvalue)
end;
```

set- und/oder Sync-Befehl als Parameter einmal angegeben werden. Im Zeichen-Modus entspricht ein Wort einem Zeichen mit bis zu 8 Attributen (wie blinkend, invers, halbe Helligkeit usw.). Im Grafik-Modus steht jedes Bit für ein Pixel auf dem Bildschirm.
Das Parameter-RAM (PRAM) ist ein schneller, interner, 16 Byte großer Speicher, dessen Inhalt durch den PRAM-Befehl von der CPU geändert werden kann.
In den ersten 8 Byte stehen Informationen über die Aufteilung des Video-RAMs. Im Zeichen-Modus gilt das auch für die restlichen 8 Byte.
Eine Teilbereichsdefinition benötigt vier Byte:

1. *Byte*: Absolute Startadresse im Video-RAM (Low-Byte)

2. *Byte*: Absolute Startadresse im Video-RAM (Middle-Byte)

3. *Byte*: Bit 0,1 sind (nur bei Grafik- und Misch-Modus) die höchstwertigsten Bits der Startadresse, falls mehr als 64 K Worte als Video-RAM angesprochen werden sollen.
   Bit 2,3 haben keine Bedeutung.
   Bit 4...7 sind das niedrigste Nibble der Länge dieses Teilbereichs.

4. *Byte*: Bit 0...5 sind der höherwertige Teil der Länge.
   Bit 6 muß für Textbereiche 0 sein, für Grafik dagegen 1.
   Bit 7 sollte i. a. 0 sein.

Die Länge des Bereiches wird in Bildschirmzeilen angegeben, und zwar unabhängig davon, ob es sich dabei um einen Text oder einen Grafik-Bereich handelt. Die Startadresse dagegen wird in Worten angegeben. Durch Erhöhen der Startadresse um die Zeilenlänge läßt sich ein schnelles Rollen des Bildschirms bewerkstelligen. Der 7220 berücksichtigt diese Daten wie folgt:

Beim Bildaufbau liest er zuerst die Startadresse des ersten Teilbereiches ein. Er malt mit den ab dieser Adresse im Video-RAM stehenden Daten so viele Bildschirm-Zeilen, wie sie für den Teilbereich angegeben worden sind. Ist er danach mit dem Bildaufbau noch nicht fertig (d. h., es sollen insgesamt mehr Zeilen angezeigt werden, als in diesem Teilbereich definiert wurden), so sieht er im nächsten Teilbereich nach – und so fort.

Im Zeichen-Modus können demnach vier Teilbereiche definiert werden. Im Misch- und Grafik-Modus können dagegen nur zwei Teilbereiche definiert werden, da die Parameter-RAM-Werte 8 bis 15 als Muster für Linien, Grafik-Zeichen und Flächen dienen (siehe figd und gchrd).

Der Speicherzugriff (Read Modify Write Cycle)

Jede Veränderung, die im Video-RAM ausgeführt werden soll, läuft nach foldendem Schema ab: Zuerst wird das Wort eingelesen, dessen Adresse im internen Register EAd abgelegt ist. Es wird dann mit dem Wert verknüpft, der sich aus einer UND-Verknüpfung des Pattern-Registers mit dem Mask-Register ergibt. Die Art der Verknüpfung hängt von dem gewählten Modus (ERSETZE, INVERTIERE, SETZE oder LÖSCHE) ab. Das Ergebnis wird wieder an seine alte Position zurückgeschrieben. Für die vier Modi ergeben sich folgende Bit-Verknüpfungen ((EAd) ist der Inhalt der durch das EAd-Register adressierten Video-RAM-Speicherstelle):

ERSETZE : (EAd) := Pattern and Mask
INVERTIERE : (EAd) := (EAd) xor (Pattern and Mask)
SETZE : (EAd) := (EAd) or (Pattern and Mask)
LÖSCHE : (EAd) := (EAd) and (not (Pattern and Mask))

Je nachdem, welcher Befehl an den 7220 gegeben werden soll, ist auch die Art und Weise verschieden, wie der Modus zu wählen ist und die Inhalte von EAd-, Pattern- und Mask-Register zu bestimmen sind (siehe curs, pram, wdat, mask, figs und rdat).

Der DMA-Kanal

Der 7220 besitzt die Fähigkeit, der CPU über einen DMA-Controller vom Typ 8257 oder 8237 eine direkte Zugriffsmöglichkeit auf sein Video-RAM zu geben. Vor jedem DMA-Zugriff muß durch einen DMAR- oder DMAW-Befehl die Art des Zugriffs definiert werden und der 7220 in den DMA-Transfer-Modus gebracht werden. Er verläßt diesen erst, nachdem die durch figs spezifizierte Anzahl von Bytes übertragen wurde.

Einbinden in Turbo-Pascal

Nun wird es praktisch. Die folgenden Routinen können (evtl. mit kleinen Abänderungen) auf allen Turbo-Pascal-tauglichen Computern mit dem 7220 zur Ansteuerung desselben benutzt werden. Bei Computern, bei denen der 7220 über Speicherstellen anstatt I/O-Ports angesprochen wird, müssen alle Zugriffe auf das PORT-Array gegen MEM-Array-Zugriffe ausgetauscht werden.
Zunächst einige fundamentale Routinen in *Bild 1*.
Diese Prozeduren und Funktionen stellen die Basis für den gesamten Datenaus-

```
procedure wdatlb(b :  byte);    (* Wdat : Höherwertiges Byte = 0 *)

begin
  putcmd($30 or byte(gdcparam.mode));
  putval(b)
end;

procedure wdathb(b : byte);    (* Wdat : Niederwertiges Byte = 0 *)

begin
  putcmd($38 or byte(gdcparam.mode));
  putval(b)
end;

procedure wdatw(i : integer);        (* Wdat : Beide Bytes aus i *)

begin
  putcmd($20 or byte(gdcparam.mode));
  putdval(i)
end;

procedure mask(i : integer);

(* Setze Mask-register. Es wird ferner verändert durch gcurs, figd
   und gchrd *)

begin
  putcmd($4A);
  putdval(i)
end;
```

Bild 5. Die wdat-Befehle: Transfer in das Video-RAM

tausch zwischen CPU und 7220 dar: putcmd(b) schiebt das Kommando mit der Nr. b auf die Warteschlange; putval(b) schiebt den Ein-Byte-Parameter b auf die Warteschlange; putdval(i) schiebt den Zwei-Byte-Parameter i auf die Warteschlange; die Funktion inpval wartet auf ein Byte vom 7220 für die CPU und liefert es als Funktionswert. Die übrigen Funktionen prüfen nur jeweils ein Bit des Statusbytes ab und liefern den boolschen Wert (TRUE/FALSE) zurück.

Mit diesen Prozeduren und Funktionen können alle Befehle des 7220 angesprochen werden. Ein paar Befehle dienen der Steuerung der Videologik selbst (reset, sync, start, bctrl), bestimmen die Abbildung des Video-RAMs auf dem Bildschirm (zoom, cchar, pitch) oder beeinflussen das Video-RAM. Die Befehle, die den Inhalt des Video-RAMs verändern, unterscheiden sich noch dahingehend, daß einige lediglich Voreinstellungen für spätere Befehle sind (curs, pram, figs, mask), andere wiederum direkte Veränderungen bewirken (wdat, figd, gchrd). Und schließlich gibt es noch diejenigen, die Werte in Abhängigkeit vom Inhalt des Video-RAMs oder interner Zustände liefern (rdat, curd).
Ein Reset, mit dessen Hilfe man den 7220 in einen definierten Zustand bringen kann, kann z. B. so aussehen wie in *Bild 2*.

Die Einstellung der hardware-abhängigen Parameter hierbei wurde weggelassen, weil es vollkommen ausreicht, sie mit dem sync-Befehl nachträglich zu setzen. Wurden sie schon einmal gesetzt, so ist es erst nach einem Hardware-Reset notwendig, sie neu zu setzen. Um mit den hardware-abhängigen Parametern zurechtzukommen, ist es sinnvoll, sich einen Record mit Voreinstellung zu definieren (ferner dient er als Gedächtnis für die letzten Einstellungen). *Bild 3* zeigt ihn.

```
procedure figs(figur : figtyp; dir : byte;
               dc,d,d2,d1,dm : integer);

(* Definiere Figure-drawing mit allen Parametern  *)

begin
  putcmd($4C);
  putval(GDCCONST(.figur.) or dir);
  putdval(dc); putdval(d); putdval(d2); putdval(d1); putdval(dm)
end;

procedure figs3(figur : figtyp; dir : byte; dc : integer);

(* Definiere Figure-drawing mit 3 Parametern *)

begin
  putcmd($4C);
  putval(GDCCONST(.figur.) or dir);
  putdval(dc)
end;

(* Entsprechend kann hier noch die für DMAW benötigte figs4-
   Prozedur eingefügt werden usw. *)

procedure figs5(figur : figtyp; dir : byte; dc,d,d2 : integer);

(* Definiere Figure-drawing mit 5 Parametern *)

begin
  putcmd($4C);
  putval(GDCCONST(.figur.) or dir);
  putdval(dc); putdval(d); putdval(d2)
end;

procedure figs6(figur : figtyp; dir : byte; dc,d,d2,d1 : integer);

(* Definiere Figure-drawing mit 6 Parametern *)

begin
  putcmd($4C);
  putval(GDCCONST(.figur.) or dir);
  putdval(dc); putdval(d); putdval(d2); putdval(d1)
end;
```

Bild 6. Figuren aufbauen

```
procedure figd;

(* Starte  Figure-drawing  für Figurtypen STRAIGHTLINE,  CIRC  und
   RECT *)

begin
  putcmd($6C)
end;

procedure gchrd;

(* Starte  Graphic-character-drawing (Figurtypen GRAPHICSCHAR  und
   SLANTED) *)

begin
  putcmd($68)
end;
```

Bild 7. Figuren malen

Der direkte Kontakt mit dem 7220

Mit diesen Voreinstellungen (evtl. leicht modifiziert) ist es einfach, die Möglichkeiten des 7220 voll auszunutzen. Zunächst in *Bild 4* die Prozeduren, die direkt die Befehle des 7220 repräsentieren (den Reset hatten wir ja schon).
Die wdat-Befehle zeigt *Bild 5*: Direktes Schreiben in das Video-RAM. Die einzelnen Befehle unterscheiden sich nur in der Interpretation der Parameter. Bei allen wird der Schreibmodus aus dem gdcparam-Record geholt. Die zu schreibenden Daten werden nicht, wie es bei figd und gchrd der Fall ist, aus dem Pattern-Reg. bzw. dem RAM geholt, sondern direkt als Parameter angegeben. Es gelten jedoch bezüglich der Verknüpfungsart die gleichen Regeln. Im Grafik-Modus interessiert nur das niedrigste Bit (alle anderen Bits werden auf den gleichen Wert gesetzt). Abhängig vom Inhalt des Mask-Reg. und des Schreibmodus wird dann das Video-RAM geändert. Im Zeichenmodus werden alle Bits beachtet. Im Mixedmodus unterscheidet der wdat-Befehl zwischen Grafik- und Zeichenmodus anhand des dc-Parameter-Bits 14 im zuletzt vorausgegangenen fig-Befehl (siehe figs). Daher sollte dieser zumindest einmal vorher als figs3 (CHARANDDOT, dir, dc) (mehr Parameter werden für wdat nicht benötigt) aufgerufen worden sein. Die dc-Parameter-Bits 0–13 werden als Wiederholungsfaktor-1 interpretiert. Mit dc = 0 wird wdat nur einmal ausgeführt. Je nachdem, welche Richtung man mit dir angegeben hat, wird die Cursor-Position nach jedem wdat um ein Wort verschoben. Es ist

```
function rdatlb : byte;          (* Hole niederwertiges (lo) Byte *)

begin
  putcmd($B0);
  rdatlb := inpval
end;

function rdathb : byte;          (* Hole höherwertiges (hi) Byte *)

begin
  putcmd($B8);
  rdathb := inpval
end;

function rdatw : integer;                    (* Hole ganzes Wort *)

var b : byte;

begin
  putcmd($A0);
  b := inpval;
  rdatw := (inpval shl 8) or b
end;
```

Bild 8. Zurücklesen aus dem Video-RAM

nicht nötig, den wdat-Befehl immer wieder neu aufzurufen, wenn sich lediglich die Daten, die in die foldenden Adressen zu schreiben sind, nicht aber die Richtung und der Schreibmodus ändern. Alle folgenden Worte sind dann lediglich als weitere Parameter mittels putval/putdval zu übergeben. Man achte hierbei darauf, daß der dc-Wert nach einmaligem Aufruf von wdat auf 0 heruntergezählt worden ist und sich auch nicht ändert.

Die einzelnen figs-Befehle (*Bild 6*), unterscheiden sich lediglich in der Anzahl der übergebenen Parameter. Dabei kommt den Parametern folgende Bedeutung zu:

figur: Art der zu definierenden Figur

-CHARANDDOT = Zeichen im Zeichenmodus, DMA-Zugriffe, WDAT und RDAT (auch für Einzelpunkte).

-STRAIGHTLINE = Gerade Linien malen.

-GRAPHICSCHAR = Zeichen im Grafikmodus, Flächen füllen (Muster).

-CIRC = Kreise und Kreisbögen malen.

-RECT = Rechtecke malen.

-SLANTED = wie GRAPHICSCHAR mit kursiver Zeichenrichtung.

dir: Zeichenrichtung
0 = unten
1 = rechts-unten
2 = rechts
3 = rechts-oben
4 = oben
5 = links-oben
6 = links
7 = links-unten

dc: Bit 0–13: Anzahl der auszuführenden Speicherzugriffe-1
Bit 14: Im Mixed-Modus: Unterscheidung zwischen Grafik- und Zeichenmodus.

d,d2,d1: Interne Parameter für bestimmte Figuren.

dm: Anzahl der Punkte, die bei Kreisen/Kreisbögen maskiert werden.

Die Befehle figd und gchrd (*Bild 7*) starten den durch figs definierten Malprozeß. Im Laufe dieses Prozesses werden das Mask-Register sowie die Cursor-Adresse verändert. Als Schreibmodus wird der im letzten wdat-Befehl definierte Modus verwendet (siehe auch setmode). Das Pattern-Register wird bei figd aus den PRAM-Adressen 8 (lo) und 9 (hi) geladen (siehe set_pattern). Das zu verwendende Muster für gchrd liegt als 8*8-Matrix im PRAM (8 bis 15, siehe define_char).

Die rdat-Befehle (*Bild 8*) bringen als Funktionswert ein Byte (Halbwort) oder ein ganzes Wort aus dem Video-RAM zurück. Ähnlich wie bei wdat ist es möglich, mehrere aufeinanderfolgende Worte des Video-RAMs auszulesen. Dafür muß bei rdat jedoch der dc-Parameter beim vorhergehenden figs-Befehl genau die Anzahl der zu lesenden Bytes enthalten und nicht, wie bei wdat, Anzahl-1. Die weiteren Daten werden dann über die inpval-Funktion ausgelesen. Bei der Übertragung ganzer Worte wird erst das niederwertige und dann das höherwertige Byte gelesen. Wie bei wdat beginnt der Transfer mit der Adresse, die durch die Cursorpositionierung festgelegt wurde.

curd liefert die Cursoradresse als absolute Punkt-Nr. in zwei Integer-Zahlen. Insgesamt sind je nach Video-RAM-Ausbau bis zu 22 Bits dafür vorgesehen. ih = höherwertiges Wort, il = niederwertiges Wort (*Bild 9*).

```
procedure curd(var ih,il : integer);

var i : integer;

begin
  putcmd($E0);
  il := inpval shl 4;
  ih := inpval;
  il := il or (ih shl 12);
  ih := (inpval shl 4) or (ih shr 4);
  i  := inpval;
  i  := i or (inpval shl 8);
  while (i and 1) = 0 do begin
    i  := i shr 1;
    il := il + 1
  end
end;
```

Bild 9. Die Cursoradresse feststellen

```
procedure lprd(var i : integer); (* Version 1, i = Adresse *)

begin
  putcmd($C0);
  i := inpval;
  i := (inpval shl 8) + i
end;

procedure lprd(var ih,il :  integer);

(* Version 2,  ih = Adresse Bit 16,17; il = Adresse Bit 0-15 *)

begin
  putcmd($C0);
  il := inpval;
  il := (inpval shl 8) + il;
  ih := inpval
end;
```

Bild 10. Position des Lichtgriffels lesen

```
procedure dmawlb; (* dmaw : Nur niederwertiges Byte *)

begin
  putcmd($34 or byte(gdcparam.mode))
end;

procedure dmawhb; (* dmaw : Nur Höherwertiges Byte *)

begin
  putcmd($3C or byte(gdcparam.mode))
end;

procedure dmaww;    (* dmaw : Beide Bytes werden übertragen *)

begin
  putcmd($24 or byte(gdcparam.mode))
end;

(* Ganz ähnlich zu rdat ist der Dmar - Befehl :
   Bei der Übertragung ganzer Wörter verlangt der dmar-Befehl
   einen Parameter mehr, als der dmaw-Befehl : figs5(CHARANDDOT,
   dir, dc1, dc2, dc2 div 2). Ansonsten sind die Voreinstellungen
   die gleichen. *)

procedure dmarlb;

(* Schicke niederwertiges (lo) Byte auf DMA - Kanal *)

begin
  putcmd($B4)
end;

procedure dmarhb;

(* Schicke höherwertiges (hi) Byte auf DMA - Kanal *)

begin
  putcmd($BC)
end;

procedure dmarw;

(* Schicke ganzes Wort (lo - Byte zuerst) auf DMA - Kanal *)

begin
  putcmd($A4)
end;
```

Bild 11. Mit DMA ins Video-RAM

```
procedure swap(var a,b : integer);
(* Hilfsprozedur zum Austausch zweier Integer-Variablen *)

var c : integer;

begin
  c := a;
  a := b;
  b := c
end;

procedure hline(x1,x2,y : integer);

(* Diese Prozedur nutzt den schnellen Zugriff des wdat-Befehls um
   eine durchgehende horizontale Linie von den Koordinaten x1,y zu
   den Koordinaten x2,y zu ziehen. Der im gdcparam-record einge-
   tragene Pattern-Wert hat keine Wirkung. Man kann jedoch den
   Schreibmodus (REPLACE/COMPLEMENT/PRESET/PSET) mittels set_mode
   einstellen. Die hier benutzte Cursorpositionierung funktioniert
   allerdings nur bis 64 KW Video-Ram. *)

var ead, h : integer;
    dad    : byte;

begin
  if x1 > x2 then swap(x1,x2);
  ead := y * gdcparam.wordsperline + x1 shr 4;
  ccurs(ead);
  dad := x1 and $F;
  h := (y * gdcparam.wordsperline + x2 shr 4) - ead;
  if h > 0 then begin
    if dad <> 0 then begin
      figs3(CHARANDDOT,2,$4000);
      mask($FFFF shl dad);
      wdatlb(1);
      h := h - 1
    end;
    dad := x2 and $F;
    if dad <> $F then h := h - 1;
    if h >= 0 then begin
      figs3(CHARANDDOT,2,h or $4000);
      mask($FFFF);
      wdatlb(1)
    end;
    if dad <> $F then begin
      figs3(CHARANDDOT,2,$4000);
      mask($FFFF shr (15 - dad));
      wdatlb(1)
    end
  end
  else begin
    figs3(CHARANDDOT,2,$4000);
    mask(($FFFF shl dad) and ($FFFF shr (15 - (x2 and $F))));
    wdatlb(1)
  end
end;

procedure plot(x,y : integer);

(* Setze Punkt abhängig vom Schreibmodus *)
```

Bild 12. Komplexere Befehle, zusammengebaut aus den prozessornahen Unterprogrammen

```
begin
  gcurs(x,y);
  figs3(CHARANDDOT,2,$4000);
  wdatlb(1)
end;

function pointset(x,y : integer) : boolean;

(* Liefert den Wert TRUE, falls der Punkt mit den Koordinaten x,y
   gesetzt ist, sonst FALSE *)

begin
  gcurs(x,y);
  figs3(CHARANDDOT,2,$4001);
  pointset := (rdatw and (1 shl (x and $F))) <> 0
end;

procedure vline(x,y1,y2 : integer);

(* Arbeitet genau wie hline, mit dem Unterschied, daß die Linie
   veritkal von x,y1 nach x,y2 läuft. *)

var ead : integer;

begin
  if y1 > y2 then swap(y1,y2);
  gcurs(x,y1);
  figs3(CHARANDDOT,0,$4000 or (y2 - y1));
  wdatlb(1)
end;

procedure set_pattern(pattern : integer);

(* Definiert den Inhalt des Pattern-Registers. Dieses wird beim
   figd-Befehl aus dem Pram Adr. 8 & 9 geladen. *)

begin
  gdcparam.pattern := pattern;
  pram(8);
  putdval(pattern)
end;

procedure set_mode(mode : modetyp);

(* Definiert den Schreibmodus. Dieser wird vom wdat-Befehl direkt
   aus dem gdcparam-record geholt. Deshalb wird hier ein dummy-
   wdat ohne Parameter zur Festlegung für den figd- und gchrd-
   Befehl ausgeführt *)

begin
  gdcparam.mode := mode;
  putcmd($20 or byte(mode))
end;

procedure fill_screen;

(* Diese Routine füllt den gesammten Bildschirm abhängig vom
   Schreibmodus aus. *)

begin
  ccurs(0); mask($FFFF);
  figs3(CHARANDDOT,2,(xmp1 shr 4) * ymp1 or $4000);
  wdatlb(1)
end;

procedure cls;

(* Je nachdem, welcher Schreibmodus aktiv ist, füllt cls den Bild-
   schirm schwarz oder weiß. *)

var helpmode : modetyp;

begin
  helpmode := gdcparam.mode;
  case helpmode of
    REPLACE,COMPLEMENT,PSET : set_mode(PRESET);
    PRESET                  : set_mode(PSET);
  end;
  fill_screen;
  set_mode(helpmode)
end;

procedure graphmode;  (* Definiere gesammten Schirm als Grafik *)

begin
  pram(0);
  putdval(0);
  if gdcparam.displaymode = MIXED then
       putdval($4000 + ymp1 shl 4)
  else putdval(ymp1 shl 4);
(* Manche Rechner bieten die Möglichkeit, dem Betriebssystem mit-
   zuteilen, in welchem Modus man sich befindet, damit auch im
   Grafikmodus die Standardprozeduren zur Ein-/Ausgabe wie write
   und ClrScr benutzbar bleiben. Der QX-10/16 z.B. benötigt dazu
   nur ein Flag. Die folgende Zeile muß man hier einfügen :
   mem(.$FE50.) := 1 *)
end;

procedure textmode;     (* Definiere gesamten Schirm als Text *)

begin
  pram(0);
  putdval(0);
  putdval(ymp1 shl 4);
(* Evtl. hier einzufügen :
   mem(.$FE50.) := 0 *)
end;

procedure line(x1,y1,x2,y2 : integer);

(* Dies ist die Standard-Linien-Procedur. Sie zieht eine Linie von
   x1,y1 nach x2,y2. Da sie über figd startet, ist es möglich über
   set_pattern eine Strichelung zu erreichen. Die eingetragenen
   Pattern- und Mode-Werte werden sicherheitshalber bei allen
   diesen Prozeduren noch einmal aktualisiert. *)

var dir : byte;
    xd,yd,d : integer;
    dc,d1 : integer;

begin
  set_mode(gdcparam.mode);
  set_pattern(gdcparam.pattern);
  if x1 > x2 then begin
    swap(x1,x2);
    swap(y1,y2)
  end;
```

```
  gcurs(x1,y1);
  xd := x2 - x1;
  yd := y2 - y1;
  if yd > 0 then if xd < yd then begin
                                   dir := 0;
                                   dc := yd;
                                   d1 := 2 * xd
                                 end
                            else begin
                                   dir := 1;
                                   dc := xd;
                                   d1 := 2 * yd
                                 end
            else begin
                 yd := -yd;
                 if xd > yd then begin
                                   dir := 2;
                                   dc := xd;
                                   d1 := 2 * yd
                                 end
                            else begin
                                   dir := 3;
                                   dc := yd;
                                   d1 := 2 * xd
                                 end
               end;
  d := d1 - dc;
  figs6(STRAIGHTLINE,dir,dc or $4000,d,d - dc,d1);
  figd
end;

const wurzel2 = 1.414213562;

procedure circle(x,y,r : integer);

(* Hier  wird ein Kreis mit Mittelpunkt x,y und Radius r  gezeich-
   net. Ansonsten gilt das gleiche wie für line *)

var dir,dc,d,d2 : integer;

begin
  set_mode(gdcparam.mode);
  set_pattern(gdcparam.pattern);
  dc := trunc(r / wurzel2 + 0.99999999) or $4000;
  d  := r - 1;
  d2 := 2 * d;
  for dir := 0 to 7 do begin
    case dir of
      0,3 : gcurs(x-r,y);
      1,6 : gcurs(x,y-r);
      2,5 : gcurs(x,y+r);
      4,7 : gcurs(x+r,y);
    end;
    figs(CIRC,dir,dc,d,d2,$FFFF,0);
    figd
  end
end;

procedure arc(x,y,r,begw,endw : integer);

(* Malt einen Kreisbogen mit Mittelpunkt x,y, Radius r Anfangswin-
   kel  begw und Endwinkel endw.  Alle Winkel sind in Grad  ausge-
   drückt. Ein Kreis hat hier 4096 Grad. *)

const dir_for_seg : array(.0..7.) of byte = (1,4,7,2,5,0,3,6);
      faktor = 0.1917475985E-3;

var i,dc,d,d2,begseg,endseg : integer;

(* Die Segmente sind im Uhrzeigersinn von oben durchgezaehlt *)

procedure draw_segment(segment : integer);

(* Malt ein achtel Kreis mit angegebener Nummer *)

begin
  case segment of
    0,7 : gcurs(x,y-r);
    1,2 : gcurs(x+r,y);
    3,4 : gcurs(x,y+r);
    5,6 : gcurs(x-r,y);
  end;
  figs(CIRC,dir_for_seg(.segment.),dc,d,d2,$FFFF,0);
  figd
end;


procedure draw_beg;

(* Malt Teil eines achtel Kreises ab begw. *)

begin
  case begseg of
    1,2 : gcurs(x+r,y);
    3,4 : gcurs(x,y+r);
    5,6 : gcurs(x-r,y);
    7,0 : gcurs(x,y-r);
  end;
  begw := begw and $FFF;
  if begseg in (.1,3,5,7.) then figs(CIRC,dir_for_seg(.begseg.),
    trunc(r * sin(($FFF - begw) * faktor) + 0.99999999) or $4000,
                          d,d2,$FFFF,0)
  else figs(CIRC,dir_for_seg(.begseg.),dc,d,d2,$FFFF,
    trunc(r * sin(begw * faktor)));
  figd
end;

procedure draw_end;

(* Malt Teil eines achtel Kreises bis endw. *)

begin
  case endseg of
    1,2 : gcurs(x+r,y);
    3,4 : gcurs(x,y+r);
    5,6 : gcurs(x-r,y);
    7,0 : gcurs(x,y-r);
  end;
  endw := endw and $FFF;
  if endseg in (.0,2,4,6.) then figs(CIRC,dir_for_seg(.endseg.),
    trunc(r * sin(endw * faktor) + 0.99999999) or $4000,d,d2,
                          $FFFF,0)
  else figs(CIRC,dir_for_seg(.endseg.),dc,d,d2,$FFFF,
    trunc(r * sin(($FFF - endw) * faktor)));
  figd
end;
```

```
procedure draw_teil;

(* Malt Teil eines achtel Kreises von begw bis endw. *)

var h : integer;

begin
  case begseg of
    1,2 : gcurs(x+r,y);
    3,4 : gcurs(x,y+r);
    5,6 : gcurs(x-r,y);
    7,0 : gcurs(x,y-r);
  end;
  begw := begw and $FFF;
  endw := endw and $FFF;
  if begseg in (.1,3,5,7.) then begin
    h    := $FFF - begw;
    begw := $FFF - endw;
    endw := h
  end;
  figs(CIRC,dir_for_seg(.begseg.),
   trunc(r * sin(endw * faktor) + 0.99999999) or $4000,d,d2,$FFFF,
   trunc(r * sin(begw * faktor)));
  figd
end;

(* Eigentliche Kreisbogenroutine.  Setzt einen Kreisbogen aus Seg-
   menten zusammen. *)

begin
  set_mode(gdcparam.mode);
  set_pattern(gdcparam.pattern);
  dc := trunc(r / wurzel2 + 0.99999999) or $4000;
  d  := r - 1;
  d2 := 2 * d;
  begseg := (Hi(begw) and $70) shr 4;
  endseg := (Hi(endw) and $70) shr 4;
  if begw > endw then begin
    for i := 0 to endseg - 1 do draw_segment(i);
    for i := begseg + 1 to 7 do draw_segment(i);
    if (begw and $FFF) = 0 then draw_segment(begseg)
    else draw_beg;
    if (endw and $FFF) = $FFF then draw_segment(endseg)
    else draw_end
  end
  else if ((begseg + 1) and 7) = endseg then begin
    if (begw and $FFF) = 0 then draw_segment(begseg)
    else draw_beg;
    if (endw and $FFF) = $FFF then draw_segment(endseg)
    else draw_end
  end
  else if begseg <> endseg then begin
    for i := (begseg + 1) and 7 to (endseg + 7) and 7 do
      draw_segment(i);
    if (begw and $FFF) = 0 then draw_segment(begseg)
    else draw_beg;
    if (endw and $FFF) = $FFF then draw_segment(endseg)
    else draw_end
  end
  else draw_teil
end;
```

```
procedure rectangle(x1,y1,x2,y2 : integer);

(* Malt ein Rechteck mit den beiden Ecken x1,y1 und x2,y2. *)

begin
  set_mode(gdcparam.mode);
  set_pattern(gdcparam.pattern);
  if x1 > x2 then swap(x1,x2);
  if y1 > y2 then swap(y1,y2);
  gcurs(x1,y1);
  figs(RECT,0,$4003,y2 - y1,x2 - x1,$FFFF,y2 - y1);
  figd
end;

procedure define_char(a0,a1,a2,a3,a4,a5,a6,a7 : byte);

(* Definiert Grafik-Muster oder -Zeichen im Pram 8-15 *)

begin
  pram(8);
  putval(a7);
  putval(a6);
  putval(a5);
  putval(a4);
  putval(a3);
  putval(a2);
  putval(a1);
  putval(a0)
end;

procedure fill_rect(x1,y1,x2,y2 : integer);

(* Ausfüllen eines Rechtecks mit vorher definiertem Muster *)

begin
  set_mode(gdcparam.mode);
  if x1 > x2 then swap(x1,x2);
  if y1 > y2 then swap(y1,y2);
  gcurs(x1,y1);
  figs5(GRAPHICSCHAR,0,(x2 - x1) or $4000,y2 - y1 + 1,y2 - y1 + 1);
  gchrd
end;

procedure draw_char(dir : byte);

(* Zeichne definiertes Grafik-Zeichen in spez. Richtung *)

begin
  set_mode(gdcparam.mode);
  figs3(GRAPHICSCHAR,dir,$4007);
  gchrd
end;

procedure draw_slant_char(dir : byte);

(* Wie draw_char, Zeichen geneigt (kursiv) *)

begin
  set_mode(gdcparam.mode);
  figs3(SLANTED,dir,$4007);
  gchrd
end;
```

```
type str14 = string(.14.);

procedure save_rect(name : str14; x1,y1,x2,y2 : integer);

(* Speichert ein Rechteck unter dem angegebenen Namen auf Disk *)

var f    : file of integer;
    h,x,y : integer;
    ead1,ead2,dad1,dad2 : integer;

begin
  assign(f,name);
  rewrite(f);
  if x1 > x2 then swap(x1,x2);
  if y1 > y2 then swap(y1,y2);
  write(f,x1,y1,x2,y2);
  ead1 := x1 shr 4; dad1 := $FFFF shl (x1 and $F);
  ead2 := x2 shr 4; dad2 := $FFFF shr ($F - (x2 and $F));
  for y := y1 to y2 do begin
    ccurs(y * gdcparam.wordsperline + ead1);
    figs3(CHARANDDOT,2,$4001);
    h := rdatw and dad1;
    write(f,h);
    for x := ead1 + 1 to ead2 - 1 do begin
      ccurs(y * gdcparam.wordsperline + x);
      figs3(CHARANDDOT,2,$4001);
      h := rdatw;
      write(f,h)
    end;
    ccurs(y * gdcparam.wordsperline + ead2);
    figs3(CHARANDDOT,2,$4001);
    h := rdatw and dad2;
    write(f,h)
  end;
  close(f)
end;

procedure load_rect(name : str14; x,y : integer;
                    var x2,y2 : integer);

(* Lädt  Rechteck  mit angegebenen Namen von Diskette  mit  linker
   oberer Ecke x,y und übergebe rechte untere Ecke an x2,y2. *)

var f             : file of integer;
    h,x1,y1,dx,ead,wordsm1,rest : integer;

begin
  assign(f,name);
  (*$I-*) reset(f);
  if ioresult <> 0 then exit; (*$I+*)
  read(f,x1,y1,x2,y2);
  wordsm1 := (x2 shr 4) - (x1 shr 4);
  dx := x - x1;
  x2 := x2 + dx;
  x1 := x;
  dx := dx and $F;
  y2 := y2 + y - y1;
  y1 := y;
  ead := x1 shr 4;
  for y := y1 to y2 do begin
    rest := 0;
    for x := 0 to wordsm1 do begin
      ccurs(y * gdcparam.wordsperline + ead + x);
      figs3(CHARANDDOT,2,$4000);
      read(f,h);
      mask((h shl dx) or rest);
      wdatlb(1);
      rest := h shr (16 - dx)
    end;
    if rest <> 0 then begin
      ccurs(y * gdcparam.wordsperline + ead + wordsm1 + 1);
      figs3(CHARANDDOT,2,$4000);
      mask(rest);
      wdatlb(1)
    end
  end;
  close(f)
end;
```

Bild 13. Bildinhalte abspeichern

Um die Funktionstüchtigkeit all dieser Prozeduren/Funktionen zu testen, soll folgendes Testprogramm dienen:

```
program grafiktest;

(*$I GR*)    (* Einladen der Grafik-Routinen *)

var c : char;
    i,j,x,y : integer;

begin
  set_pattern($CCCC); set_mode(PSET);
  gdcreset;                              (* Dieser Teil ist nicht *)
  sync(MIXED,FALSE,FALSE);               (* nötig, wenn der 7220 *)
  cchar(FALSE,FALSE,1,13,14);            (* schon initialisiert *)
  zoom(0,0);                             (* ist *)
pitch(80);
graphmode;
start;
cls;
for i := 0 to ymax do hline(i,xmax - i,i);
read(kbd,c);
set_mode(COMPLEMENT);
for i := 0 to ymax div 2 do hline(0,i + 31,i);
read(kbd,c);
set_mode(PRESET); cls;
read(kbd,c);
vline(200,100,300);
read(kbd,c);
hline(100,300,100); hline(100,300,300);
read(kbd,c);
cls;
read(kbd,c);
for i := 0 to ymax div 2 do vline(i,i,ymax - i);
read(kbd,c);
set_mode(COMPLEMENT);
line(0,ymax div 2,xmax,ymax div 2);
line(xmax div 2,0,xmax div 2,ymax);
line(0,0,xmax,ymax);
line(0,ymax,xmax,0);
x := xmax div 2; y := ymax div 2;
read(kbd,c); cls;
for i := 0 to xmax do begin
  line(x,y,i,ymax);
  line(x,y,i,0)
end;
```

```
    for i := 0 to ymax do begin
      line(x,y,xmax,i);
      line(x,y,0,i)
    end;
    read(kbd,c);
    set_mode(PSET); cls; plot(100,100);
    if not pointset(100,100) then begin
      textmode; ClrScr;
      writeln('plot oder pointset defekt!',^G);
      halt
    end;
    read(kbd,c);
    set_mode(PRESET); plot(100,100);
    if pointset(100,100) then begin
      textmode; ClrScr;
      writeln('plot oder pointset defekt!',^G);
      halt
    end;
    read(kbd,c);
    set_mode(PRESET); cls; plot(100,100);
    if pointset(100,100) then begin
      textmode; ClrScr;
      writeln('plot oder pointset defekt!',^G);
      halt
    end;
    read(kbd,c);
    set_mode(PSET); plot(100,100);
    if not pointset(100,100) then begin
      textmode; ClrScr;
      writeln('plot oder pointset defekt!',^G);
      halt
    end;
    read(kbd,c);
    set_mode(PSET); cls;
    for i := 1 to 10 do circle(320,200,15 * i);
    read(kbd,c);
    cls;
    for i := 1 to 10 do rectangle(30 * i,18 * i,xmax - 30 * i,
                                  ymax - 18 * i);
    read(kbd,c);
    cls;
    for i := 1 to 20 do
      arc(320,200,10 * i,(18432 - i*512) and $7FFF,
                         (18432 + i*512) and $7FFF);
    read(kbd,c);
    cls;
    for i := 1 to 64 do
      arc(320,200,3 * i,i*512 and $7FFF,(16384 + i*512) and $7FFF);
    read(kbd,c);
    set_mode(REPLACE);
    define_char(8,$14,$22,$41,$82,$44,$28,$10);
    fill_rect(160,100,480,300);
    read(kbd,c);
    define_char($8C,$92,$62,$01,$01,$62,$92,$8C);
    gcurs(0,350);
    draw_char(0);
    read(kbd,c);
    cls;
    for j := 0 to 7 do begin
      gcurs(320,200);
      for i := 1 to 20 do draw_char(j)
    end;
    read(kbd,c);
    cls;
    for j := 0 to 7 do begin
      gcurs(320,200);
      for i := 1 to 20 do draw_slant_char(j)
    end;
    read(kbd,c);
    set_mode(PSET);
    save_rect('TEST',160,40,480,360);
    cls;
    hline(0,xmax,100);
    hline(0,xmax,200);
    hline(0,xmax,300);
    read(kbd,c);
    load_rect('TEST',168,48,x,y);
    read(kbd,c);
    with gdcparam do
      cchar(TRUE,blinkcursor,blinkrate,curtop,curbot);
    textmode;
    ClrScr
  end.
```

Wenn die Position des Lichtgriffels lokalisiert worden ist (siehe Status-Reg.), so kann über **lprd** (*Bild 10*) die Adresse des Wortes im Video-RAM, auf das der Lichtgriffel zeigt, ausgelesen werden. Sollten Sie mehr als 64 K Worte Video-RAM adressieren wollen, so müssen Sie Version 2, sonst Version 1 nehmen.

Die dmaw-Befehle (*Bild 11*): Diese Befehle sind den wdat-Befehlen sehr ähnlich. Sie schreiben jedoch nicht direkt in das Video-RAM, sondern lösen einen DMA-Zugriff aus, analog zum wdat-Befehl. Die vorher mit figs4(CHARANDDOT, dir, dc1, dc2) eingestellten Parameter haben jedoch zum Teil eine etwas andere Bedeutung: Per DMA wird jeweils ein Rechteck in das Video-RAM geladen oder aus ihm gelesen. Die Ausrichtung dieses Rechtecks ist abhängig von dir. Der dc1-Parameter gibt an, wieviele Wörter im rechten Winkel zur angegebenen Richtung zu beachten sind (Höhe des Rechtecks - 1). Der dc2-Parameter gibt die Breite des Rechtecks in Bytes an (-1).

Höhere Befehle

Mit dem geschilderten Satz an Funktionen und Prozeduren sind alle Befehle des 7220 direkt ansprechbar. Da diese zweite Schnittstelle zum 7220 (die erste waren die Prozeduren und Funktionen, die direkt auf die Portadressen zugriffen) aber immer noch nicht besonders bedienungsfreundlich ist, möchte ich noch einen Vorschlag für eine dritte Ebene machen. Diese letzte Schnittstelle soll dem Programmierer den Umgang mit dem 7220 soweit erleichtern, daß auch Ungeübte seine Fähigkeiten nutzen können (*Bild 12*).

Die letzten beiden Prozeduren zeigen, wie man in Turbopascal ein definiertes Rechteck aus dem Grafikbereich auf Diskette speichern und später wieder zurückholen kann (*Bild 13*).

Die Routinen sind nicht auf maximale Geschwindigkeit, sondern auf möglichst modularen Aufbau ausgerichtet. Für den einzelnen Anwendungsfall können sie weitgehend zusammengelegt werden, um maximale Effizienz zu erzielen.

Auch ist es unwahrscheinlich, daß in einem Programm alle Möglichkeiten Einsatz finden. Dem Programmierer sollen die Routinen eine prinzipielle Übersicht verschaffen und ihm ermöglichen, seine eigenen Vorstellungen zu verwirklichen. Viel Erfolg!