Experimentierprogramm für Schrittmotoren

Tales from the crypt: Schrittmotorenansteuerung unter DOS mit Turbopascal

Vor ein paar Jahren bekam ich zum Geburtstag von einem begnadeten Modellbauer (sieht /6/) eine selbst gebaute Portalfräse mit Schrittmotorantrieb geschenkt (ja – Glück muss man haben). Als Schrittmotoren waren drei bipolare Motoren mit geringer Leistungsaufnahme verbaut. Für diese Motoren bietet die Firma Conrad Elektronik (siehe /4/) preisgünstige fertige 3-Achsen-Ansteuerungen an (SMC800 Artikel-Nr.: 967599-62 und SMC1500 Artikel-Nr.: 967785 - 62). Diesen Steuerung liegt eine genaue Beschreibung und freundlicher Weise auch gleich Demonstrationsprogramme in Pascal und C als Basis für die eigene Programmerstellung bei. Will man aber damit seine eigene Ansteuerung verwirklichen, wird es problematisch. Das Programm ist ziemlich monolithisch und unübersichtlich geschrieben. Was man zur Motorsteuerung braucht und warum, ist mühevoll zu erkennen. Natürlich bietet Conrad Elektronik auch die fertige Software zur Schrittmotorsteuerung an – aber ich wollte mich selbst damit beschäftigen.

Darum und in Erinnerung an eine Zeit, wo ich noch davon träumte, mit einem Computer mehr zu machen, als auf immer leistungsfähigeren Rechnern immer noch leistungshungrigere Betriebssysteme zu installieren und am Laufen zu halten, nahm ich mir vor, das Ansteuerprogramm so zu gestalten, dass ein einfaches und leicht zu änderndes Modulares 'Experimentalprogramm' entsteht und mit meiner Fräse zu experimentieren.

Aufbauend auf die Tatsache, dass für anspruchsvolle NCP (Numeric Controlled Programmiersysteme) 1995 noch PC Systeme mit 25 MHz, 4 bis 8 MB Speicher und 200 MB Harddisk gefordert wurden (/1/), war es für mich klar, zurück zu den Wurzeln zu gehen und unter DOS zu arbeiten. Dadurch kehrt die Freude am Computer und Programmieren wieder zurück – und die Kindheitsträume: Da ist man plötzlich in der Welt des kleinen Taschengelds: Eine DOS-Maschine, die den notwendigen 'damaligen Spitzenforderungen' genügt, bekommt man heute geschenkt oder erhält sogar noch Geld für die 'Entsorgung'. Programmieren kann man mit Turbo-Pascal 5 (TP5.5 siehe /3/) – das ist gratis und frei – oder mit Free-Pascal. Die so erstellten Programme sind klein und schnell. Und der permanente Update-Wahnsinn wegen Security und neuer nicht benötigten Features hat ein Ende. Natürlich ist für die Programmerstellung und die Dokumentation ein System mit grafischer Oberfläche (Windows oder LINUX) angenehm – aber nicht notwendig.

Architektur

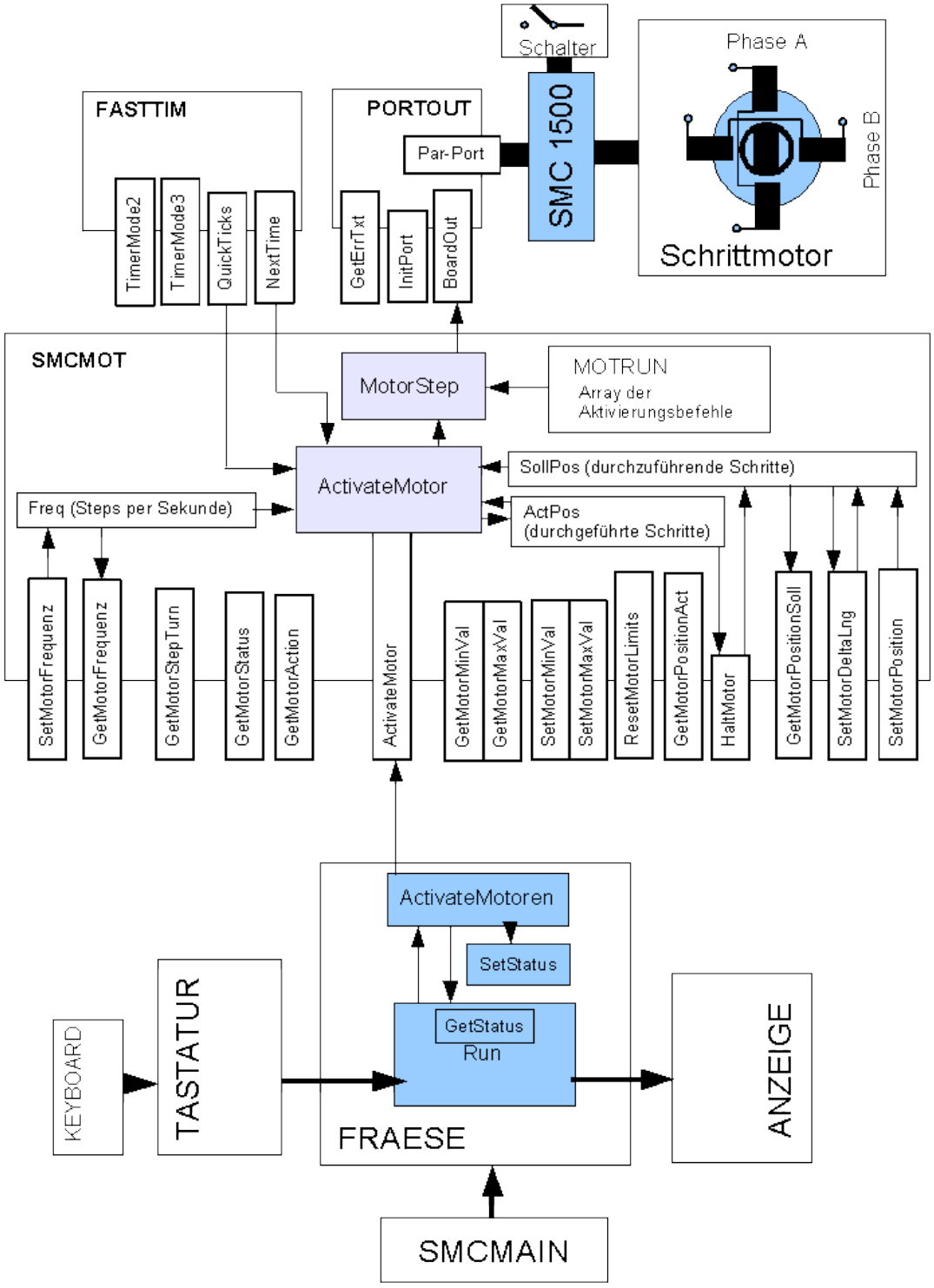
Für die Programmerstellung habe ich mich, für TP5.5 entschieden, da es noch die angenehmen Port-Befehle zur Hardwareadressierung enthält. In Freepascal fehlen dies Befehle. Dort müsste man sie als Assemblerunterprogramme einbringen. Die Programmstruktur nimmt auf solche Umbauten dadurch Rücksicht, dass diese Programmteile in eigene Module ausgelagert sind. Sie können so leicht ausgetauscht werden.

Bei der Programmgestaltung wurde auf Objektorientierung verzichtet – es wurde strikt strukturiert gearbeitet. Der Grund ist, dass die Objektorientierung mit ihrer Kapselung und Vererbungsstruktur Programmänderungen nur dann leicht ermöglicht, wenn diese

Änderungen eine Erweiterung des bestehenden Programmaufbaus (Generalisierungen oder Spezialisierungen der Klassen) sind – nicht jedoch, wenn sich der Aufbau selbst ändert. Diese Einschränkung aber widerspricht der Absicht, ein 'Experimental-Programm' zu haben, mit dem man leicht Ansteuerkonzepte erproben kann.

Generell ein Wort zur Programmgestaltung: Mit einem objektorientierten Ansatz zu starten, ist für ein Programm mit dem man noch 'spielen' will nur für sehr erfahrenen Programmierer empfehlenswert. Der Grund ist: Die Vererbung kann bei Änderungen in Programmteilen Auswirkungen zeigen, die man nicht erwartet hat – und: hat man sich zu Beginn für ein ungünstiges Objektmodell entschieden – ist entweder das ganze Programm ein Murks oder die Umstellung auf ein geändertes Modell sehr mühsam.

Der strukturieret, modulbasierte Ansatz ist da wesentlich leichter zu handhaben und zu testen.



Übersicht Programmarchitektur

Nun zur Programmrealisierung:

Das Logging

Zur Problemanalyse im laufenden Programm ist eine sehr einfach gehaltene Logging-Möglichkeit vorgesehen. Als Log wird eine Textdatei mit dem Namen 'SMCMAIN.LOG' im Verzeichnis des Programms angelegt und fortlaufend geschrieben (kein Umlaufpuffer). Bei Programmstart ist das Log abgeschaltet.

Handhabung der Log-Schnittstelle:

Aufruf :

```
WriteLog ( Modulname, Funktionsname, Stelle in der Funktion, Zusatztext)
```

Zur systematischen Verwendung soll in jedem Modul eine Stringkonstante 'ModNam' mit dem Namen des Moduls und in jeder Funktion oder Prozedur eine Prozedurkonstante 'ProcName' mit dem Namen der Prozedur angelegt werden.

Bsp.:

```
    Const ModNam = 'SMCMAIN';
```

Diese Konstanten werden dann der Log-Schnittstelle übergeben.

Bsp.:

```
    WriteLog (ModNam, ProcNam, 'Aufruf InitPort', 'Zusatztext');
```

Im allgemeinen soll in den Funktionen am Anfang und am Ende protokolliert werden. Als Zusatztext sollen die Übergabeparameter und die resultierenden Ergebnisse beinhalten.

Natürlich soll in Schleifen vorsichtig protokolliert werden (Datenmengen). Diese Log-Schnittstelle ist nicht für das Programm-Debuging gedacht, dafür bietet TP5.5 exzellent Möglichkeiten, sondern um später im fertigen Programm die Möglichkeit zu haben, bei auftretenden Problemen, erste Hinweise für eine Fehleranalyse zu bekommen.

Der Programmaufbau

Der Kernteil der Ansteuerung, das Modul SMCMOT.PAS wurde so gestaltet, dass nur direkt den Motor betreffende Kommandos in dem Modul vorhanden sind. Jegliche 'höhere' Verknüpfung oder Umrechnung muss vom Aufrufrahmen durchgeführt werden.

Die eigentliche Motorsteuerung besteht aus den 3 Komponenten

- Ansprache der Steuerelektronik angeschlossen am Druckerport: PORTOUT.PAS
- Erzeugen des Taktes für die Steuerung: FASTTIM.PAS
- Steuerung der Motorphasen: SMCMOT.PAS

Wobei SMCMOT der eigentliche Kern der Ansteuerung ist.

Die anderen Module dienen als Beispiel für einen möglichen Anwenderrahmen mit Bedieneingabe und Statusanzeige:

- Lesen Tastaturpuffer und Zeichenkonvertierung: KEYBOARD.PAS
- Behandlung der Eingaben: TASTARUR.PAS

- Visualisierung der Daten: ANZEIGE.PAS
- Steuerrahmen: FRAESE.PAS

Das Hauptmodul SMCMAIN.PAS erledigt den Anstoß der Programmschleife.

Auf die Verarbeitung von Koordinatendateien zu Ansteuerung und auf das Laden und Speichern der Kenndaten und Parameter habe ich fürs Erste verzichtet um das Programm klein zu halten. Alle Aktionen und Positionsangaben werden mittels einer sehr einfach gehaltenen Tastatureingabe getätigt.

Der Motor

Für die Erklärung der Steuerung von Schrittmotoren reicht das Modell eines 2 Phasen Permanentmagnet-Schrittmotor. Dieser ist im einfachsten Fall ein Gehäuse mit 4 Polen welche die Phasenspulen tragen und einem Permanentmagnet als Rotor. Dadurch, dass der Rotor ein Permanentmagnet ist, steht er in Ruhestellung unter einem der Pole mit einem gewissen restlichen Haltemoment (das spürt man, wenn man den Motor an der Achse dreht). Wird nun der Motor mit Strom beaufschlagt, entsteht an den Statorpolen durch die Polwicklungen (Phasen) ebenfalls ein Magnetfeld. Der Rotor wird nun zum Pol mit der dem Rotor entgegengesetzter Magnetisierung gezogen. Beaufschlagt man nun danach den nächste Pol (die nächste Phase) mit Strom derart, dass dieser nun den Rotor anzieht, dreht sich der Rotor um einen Schritt weiter. Werden nun die Phasen zyklisch der Reihe nach bestromt, dreht der Rotor und der Motor 'läuft'. (Für genauere Erklärung und Darstellung der unterschiedlichen Schrittmotoren und Ansteuerungsarten siehe /2/ oder sonstige Literatur über Schrittmotoren).

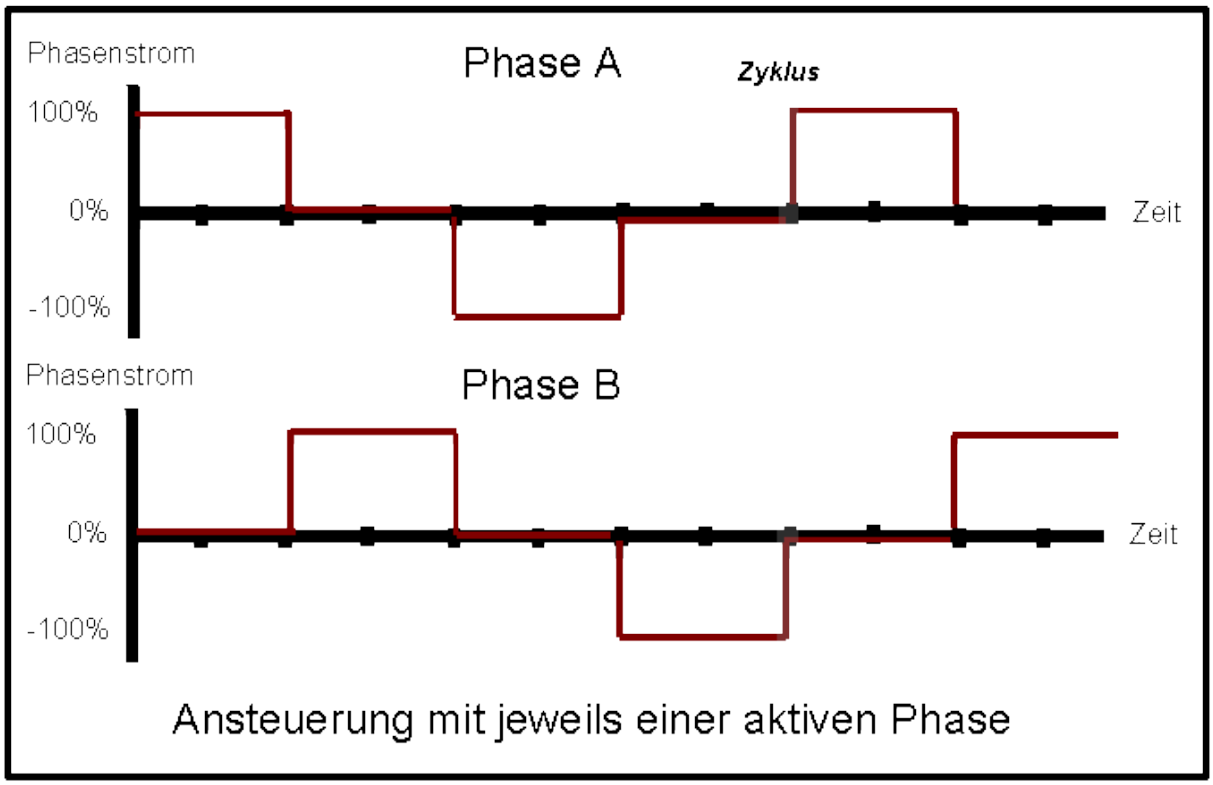
Die einfachste Art um eine Motor mit Strom zu beaufschlagen ist es, eine Spule einzuschalten und wieder abzuschalten und mit der nächste Spule dann das Gleiche zu machen und so weiter. Man fährt also im 'Vollschrittbetrieb', der Anker springt dabei von einem Pol zum nächsten.

Eine zweite Möglichkeit ist, kurz nach dem Einschalten der ersten Phase auch die zweite einzuschalten. Man erreicht damit, dass der Anker auch zwischen 2 Polen stabil stehen bleiben kann – der Motor läuft dann im 'Halbschrittbetrieb'. Im Halbschritt sollte man den Phasenstrom bei Beaufschlagung beider Phasen gleichzeitig absenken, da das Haltemoment zwischen den Phasen bei voller Strombeaufschlagung größer ist als unter den Polen – der Motor läuft dann unruhig.

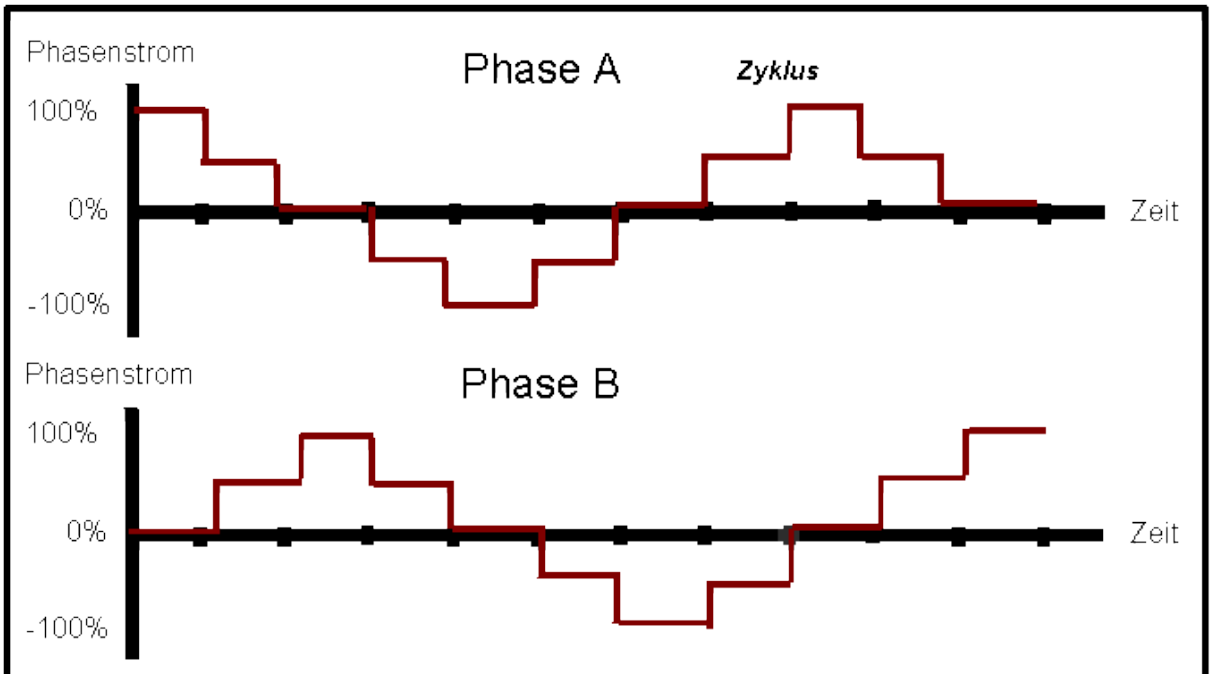
Die Motorsteuerung

Die SMC1500 Ansteuerkarte hat einen Befehlssatz zur Einstellung der Phasenströme in 4 Stufen (0, 20, 60 und 100% des Maximalstroms). Für diese Karte gibt es von Conrad Elektronik auch ein Zusatzmodul, welches die Steuerung einfach mit einem Takt und Richtungssignal ermöglicht. Das Modul SMCMOT.PAS ist zur Ansteuerung der SMC1500 ohne Zusatzkarte gedacht und stellt somit die Befehle zur Phasenstromeinstellung bereit. Die Motoren werden im 'Halbschrittbetrieb' angesteuert. Die einzustellenden Stromwerte werden in einer Konstantentabelle im Programm hinterlegt.

Im Stillstand kann der Motor bestromt oder unbestromt stehen. Soll der Motor bestromt stillstehen um ein stabiles Haltemoment zu haben, sollte man den Phasenstrom im Stillstand absenken, um eine unzulässige Erwärmung der Motoren zu verhindern. Diese



Ansteuerung mit jeweils einer aktiven Phase



Ansteuerung mit zwei aktiven Phasen (Halbschrittbetrieb)

'Stillstandsbestromung' erreicht man am einfachsten, in dem man eine zweite Befehlstabelle verwendet. Da meine Fräse über Gewindestangen angetrieben wird und dieser Antrieb von Haus aus ein relativ hohes Haltemoment hat, schalte ich die Motoren im Stillstand einfach ab. Für die Motorbewegung verwende ich die Ansteuerung im Halbschrittbetrieb mit den Werten 0%, 20% und 100% des Phasenstromes für alle 3 Motoren. Die zugehörigen Bitmuster sind im Modul SMCMOT in der Tabelle MOTRUN als Konstanten abgelegt.

Programmfragment Ansteuertabelle mit Kommentar:

```
// Parallelport SMC1500 ohne Zusatzplatine
// Steckerbelegung
//      Pin 2 - 10: Daten 1 - 8 (Data1 .. Data8)
// Datenbelegung
//       8    7    6    5 |  4    3    2    1
//    +----+----+----+----+----+----+----+----+
//                 |              |    x    x    = Control   Phase A
//                 |              |    1    1    =  0% max curr
//                 |              |    0    1    = 20%
//                 |              |    1    0    = 60%
//                 |              |    0    0    =100%
//                 |              +--------------- Direction Phase A
//                 |
//                 |     x    x                  = Control   Phase B
//                 |     1    1                  =  0% max curr
//                 |     0    1                  = 20%
//                 |     1    0                  = 60%
//                 |     0    0                  =100%
//                 +------------------------------ Direction Phase B
//
//      0      0                                 = Motor X  (  0, $00)
//      0      1                                 = Motor Y  ( 64, $40)
//      1      0                                 = Motor Z  (128, $80)
//
}
Const
STEPMAX = 7;
 MOTRUN:  Array[0..STEPMAX] of Byte =
 ( $03,    { -> 0: 0000 0011   B= 0  100%  A= 1    0%}
   $09,    {    1: 0000 1001   B= 0   20%  A= 1   20%}
   $38,    { -> 2: 0011 1000   B= 1   -0%  A= 1  100%}
   $29,    {    3: 0010 1001   B= 1  -20%  A= 1   20%}
   $27,    { -> 4: 0010 0111   B= 1 -100%  A= 0   -0%}
   $2D,    {    5: 0010 1101   B= 1  -20%  A= 0  -20%}
```

```
  $1C,    { -> 6: 0001 1100   B= 0    0%  A= 0 -100%}
  $0D     {    7: 0000 1101   B= 0   20%  A= 0  -20%}
);
```

Wie man sieht, wird die Stromstärke und die Richtung in 3 Bit pro Phase codiert. Diese Bitmuster werden in ein Array abgelegt und bilden die Ansteuertabelle, die zyklisch durchlaufen wird. Die 2 Bit, welche den Motor identifizieren, werden mittels OR beigefügt. Für den Stillstand werden die Motoren durch die Ausgabe von 0% Phasenstrom abgeschaltet. Dazu habe ich das Kommando 0% Strom für Phase A und B ebenfalls als Konstante abgelegt.

```
MOTZERO = $1B;  {OR-Mask für Strom 0% beide Phasen }
                { 0001 1011                         }
```

SMCMOT ist so aufgebaut, dass durch Vorgabe einer Sollposition die Motoren solange bestromt werden, bis die vorgegeben Position erreicht ist – oder ein Fehler auftritt. Die Vorgabe der Positionen erfolgt in SMCMOT immer in Schritten (genauer gesagt in Halbschritten).

Die Endschalter

Die Ansteuerkarte beinhaltet auch die Handhabung von Endschaltern. Es ist nur ein Eingang für einen Endschalter vorhanden. Alle angeschlossenen Endschalter sind an diesen Eingang parallel angeschlossen.

Zur Identifizierung der Schalter werden im Programm Wertetypen definiert.

```
type TSwitchPosition = (xmin, xmax, ymin, ymax, zmin, zmax, unknown);
```

Die Identifizierung der betätigten Endschalter muss durch die Programmlogik basierend auf den aktuellen Bewegungszustand erfolgen. In SMCMOT wird daher nach jeder Motorbewegung der Schalterstatus eingelesen. Aus der Motoridentifikation und der Bewegungsrichtung wird dann auf den Endschalter geschlossen. Man sollte sich im klaren sein, dass diese Schlussfolgerung falsch sein kann - z.B. wenn der Schalter bei Programmstart bereits geschlossen war.

Auswerten des Endschaltersignals:

```
        if ReadSwitch = TRUE then begin        { auf Schalter gefahren }
          switch.act :=CLOSE;
          if switch.pos = unknown then begin   { noch nicht gemeldet   }
            case motID of
              xmotor:
                if action = foreward then switch.pos := xmax
                else                      switch.pos := xmin;
              ymotor:
                if action = foreward then switch.pos := ymax
                else                      switch.pos := ymin;
              else
                if action = foreward then switch.pos := zmax
                else                      switch.pos := zmin;
```

```
            end;
          end;
        end
        else begin
          switch.act := open;
          switch.pos := unknown;
```

Sollen zusätzliche Geber an der Fräse installiert werden (z.B.: Werkzeuglängensensoren), müssen diese in gleicher Weise identifiziert werden – aus der Programmlogik heraus.

Der Takt

Die Steuerbefehle müssen nun fortlaufend an die Motorsteuerung übermittelt werden. Dies kann entweder durch einen von einem Timer ausgelösten Interrupt oder durch dauerndes Aufrufen der Befehlsausgabe in einer Schleife erfolgen – polling genannt. Ich habe mich entschieden, dies im 'Polling Mode' zu tun und nicht per Interruptfunktion. Der Grund liegt darin, dass ein Programm ohne Interrupt-Aktivierung einfacher zu handhaben ist - es ist unkritischer.

Die Ausgabe der Befehle muss aber in gewissen Zeitintervallen erfolgen. Erfolgt die Ausgabe zu schnell, können die Schritte vom Motor nicht durchgeführt werden – er beginnt zu brummen, wird warm und bewegt sich nicht. Erfolgt die Ausgabe zu langsam, dreht der Motor zu langsam. Der Ausgabetakt hat also in einer Geschwindigkeit zu erfolgen, die 'brauchbar' ist. Da die 'normalen' BIOS – Timer – Ticks (INT 08 hex) in ca 55 msec Abstand erfolgen – und damit nur eine maximale Schrittfrequenz von ca 18HZ (18 Schritte pro Sekunde) möglich sind, würden nur sehr langsame Motorbewegung möglich sein – überhaupt im Halbschrittbetrieb. Um schneller Takten zu können, wird der Timerbaustein 8253 der im PC für das Timing zuständig ist (siehe /8/) vom Mode 2 in den Mode 3 geschaltet - damit sind 0.2145524 Millisekunden per Tick, also ca. 4,661KHz (4661 Steps pro Sekunde), möglich. Dieser Wert ist allerdings nur rechnerisch gegeben da das Programm nicht auf Geschwindigkeit optimiert ist. Den tatsächlichen Wert habe ich nicht ermittelt.

Die Unterprogramme zur Timerbehandlung wurden aus Chip Spezial (siehe /5/) übernommen und im Modul FASTTIM.PAS zusammengefasst. Das Umschalten in den Timer Mode 3 erfoglt aus dem Initialisierungs-Teil des Moduls SMCMOT – für das Rückschalten wurde die Funktion MotEnd eingeführt. Somit sind alle Aufrufe zur Timeransprachen in SMCMOT gekapselt.

Um den Aufruf durchzuführen, wurde eine Endbearbeitung auch im Modul FRAESE.PAS aufgenommen – diese Prozedur wird beim Verlassen des Bedienrahmens aufgerufen (TASTATUR.PAS: HaltModul). Dadurch wird das Rückschalten in den Mode 2 vor verlassen des Programms durchgeführt.

Das Polling

Der Pollingrahmen wird durch die Eingabeschleife 'Eingabe' im Modul TASTATUR.PAS gebildet.

```
Procedure Eingabe;
```

```
Const ProcNam = 'Tastatur';
var keyNew, keyOld : char;
    repCtr, delt   : integer;

begin
 keyOld     := char(0);
 keyNew     := char(0);
 repCtr     := 0;

 SetStatus     (CNC_OK);               { Alles OK anfahren !   }
 SetSwitchMode (ignore);               { f□r offline-Erprobung }
 ShowFraese;                            { Bild aufblenden       }

 WriteLog (ModNam, ProcNam, 'Entry', '');
 while( GetStatus <> CNC_CANCEL ) do           { bis Abbruch   }
 begin
   if IsPressed then begin
     keyNew := InKey;
     if( keyNew = keyOld ) then begin          { Eingabewiederholung }
       if repCtr < 20 then inc( repCtr )
       else delt := 10;
     end
     else begin
       delt   := 1; repCtr := 1;
     end;

     DoKeyAction (keyNew, delt);        { Eingabe auswerten }
     ShowFraese;                        { Summe zeigen      }
     keyOld := keyNew;
   end
   else begin
     Run;
     If GetStatus = CNC_CANCEL then begin
       ShowMessage ('Abgebrochen');
       SetStatus (CNC_OK);              { im Bedienrahmen bleiben }
     end;
   end;
 end; { End --- while --- }
 FraesEnd;
end;
```

Im Modul FRAESE.PAS ist im Aufruf 'Run' eine zweite Schleife versteckt. Diese Funktion arbeitet die Bewegungsbefehle ab und ist der eigentlich Aufruf an SMCMOT in ActivateMotoren.

```
Procedure Run;
begin
  if GetStatus = CNC_RUN then begin
    cnc.display;                              { Anfangszustand Zeigen  }
    while (GetStatus = CNC_RUN) do
    begin
      SetStatus (ActivateMotoren);            { Fahren   }
      cnc.breakKey;                           { Abrage Bedienereingabe }
      cnc.showPos;                            { Anzeigen }
    end;
    cnc.display;                              { Endzustand zeigen      }
    HaltMotoren;
  end;
end;
```

Diese Konstruktion ist etwas unschön – schöner wäre es nur 'run' aufzurufen und von dort zu verzweigen, da die Bedieneingabe eigentlich als 'Dekoration' eingehängt ist – also über Funktionszeiger aufgerufen wird.

Anzeige und Eingabe:

Anzeige und Eingabe werden über Funktionszeiger in die Fräsensteuerung eingehängt. FRAESE.PAS selbst definiert nur die Schnittstelle und den Einhängepunkt.

Einhängepunkte:

```
type TDisplayProc= Procedure;     { Zeiger auf angemeldete Anzeige        }
type TBreakKey   = Procedure;     { Zeiger auf angemeldete Abbrucheingabe }
type Tcnc = Record       {Daten der Fräse ------- }
  .........
  .........
  display  : TDisplayProc;
  showPos  : TDisplayProc;
  breakKey : TBreakKey;
end;
```

Schnittstelle:

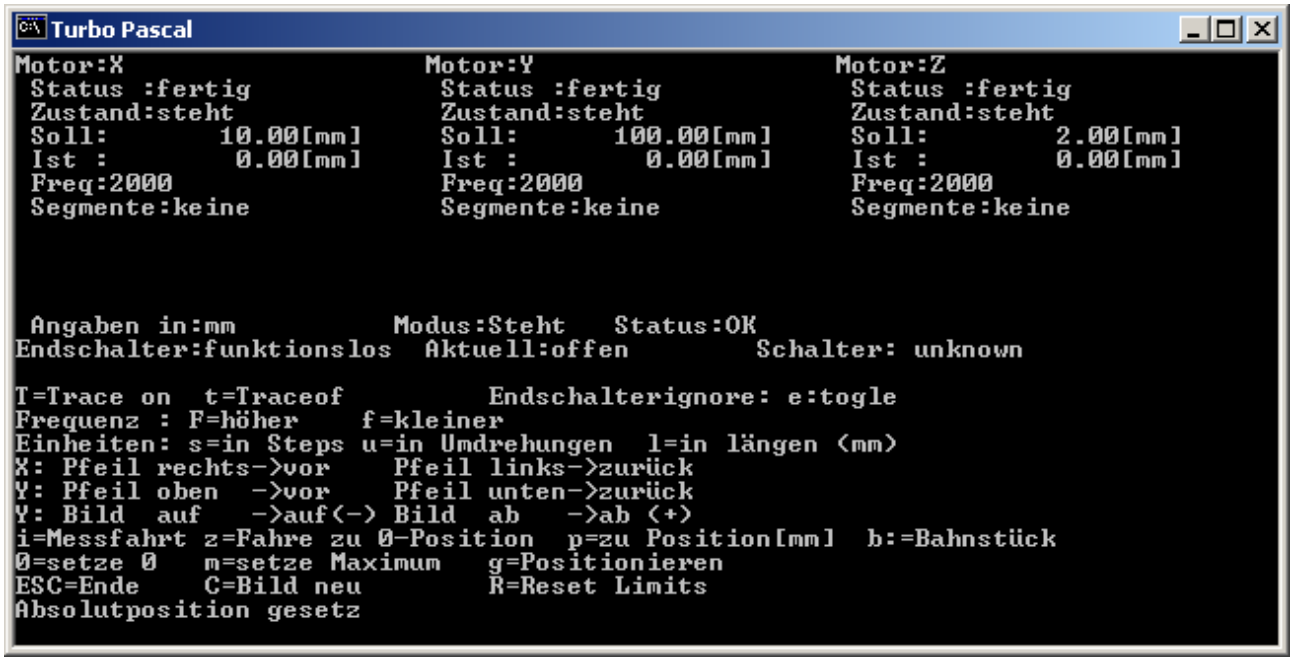
```
Procedure SetDisplay  (proc : TDisplayProc);
Procedure SetShowPos  (proc : TDisplayProc);
Procedure SetBreakKey (proc : TBreakKey   );
```

Das Einhängen erfolgt jeweils im Initialisierungsteil der Modulen TASTATUR.PAS und ANZEIGE.PAS.

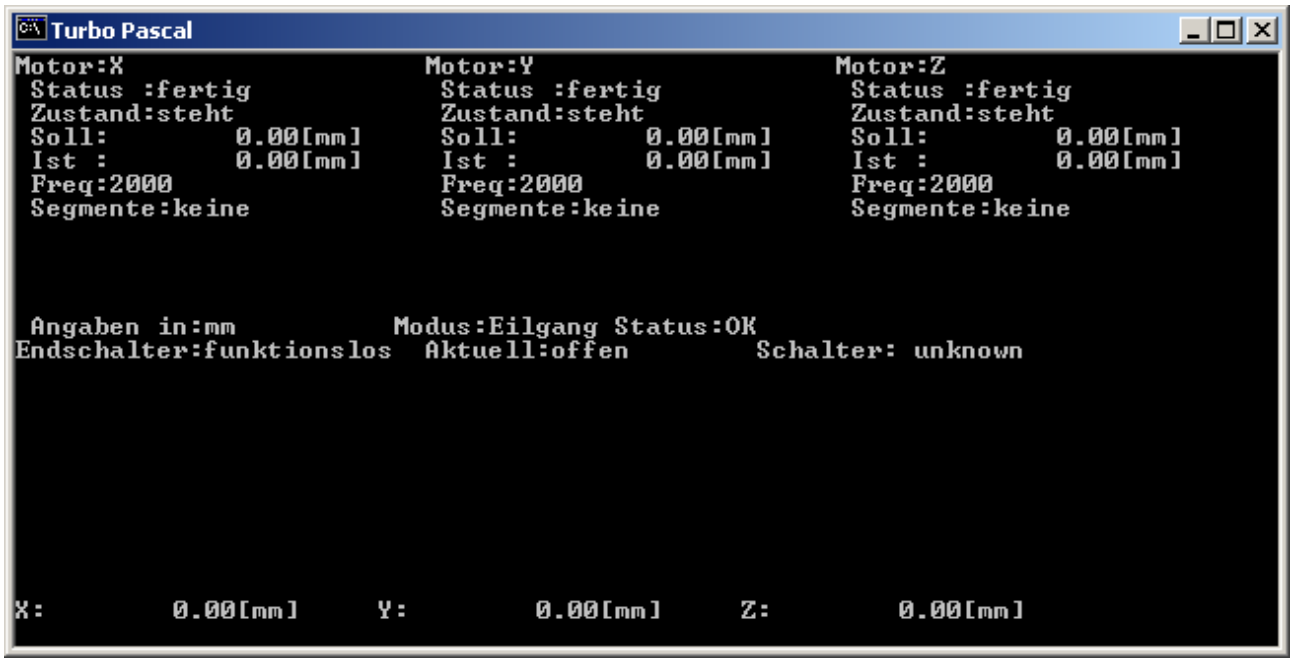
```
{ ------------------------------------------------- }
{ Modul initialisierung                             }
{ ------------------------------------------------- }
var p : Pointer;
begin
  p := @CheckKey;
  SetBreakKey ( TBreakKey(p)); { Tastaturabbruch in Fraese eintragen }
end.

{ ------------------------------------------------- }
{ Modul initialisierung                             }
{ ------------------------------------------------- }
var p : Pointer;
begin
  p := @ShowFraese;
  SetDisplay ( TDisplayProc(p)); { Display in Fraese eintragen }
  p := @ShowPos;
  SetShowPos ( TDisplayProc(p));
  ClrScr;
end.
```

Durch dieses Vorgehen, werden diese Funktionen leicht austauschbar. Es besteht, außer der definierten Schnittstellen für 'CheckKey', SchowFraese und 'ShowPos' keine Abhängigkeiten.

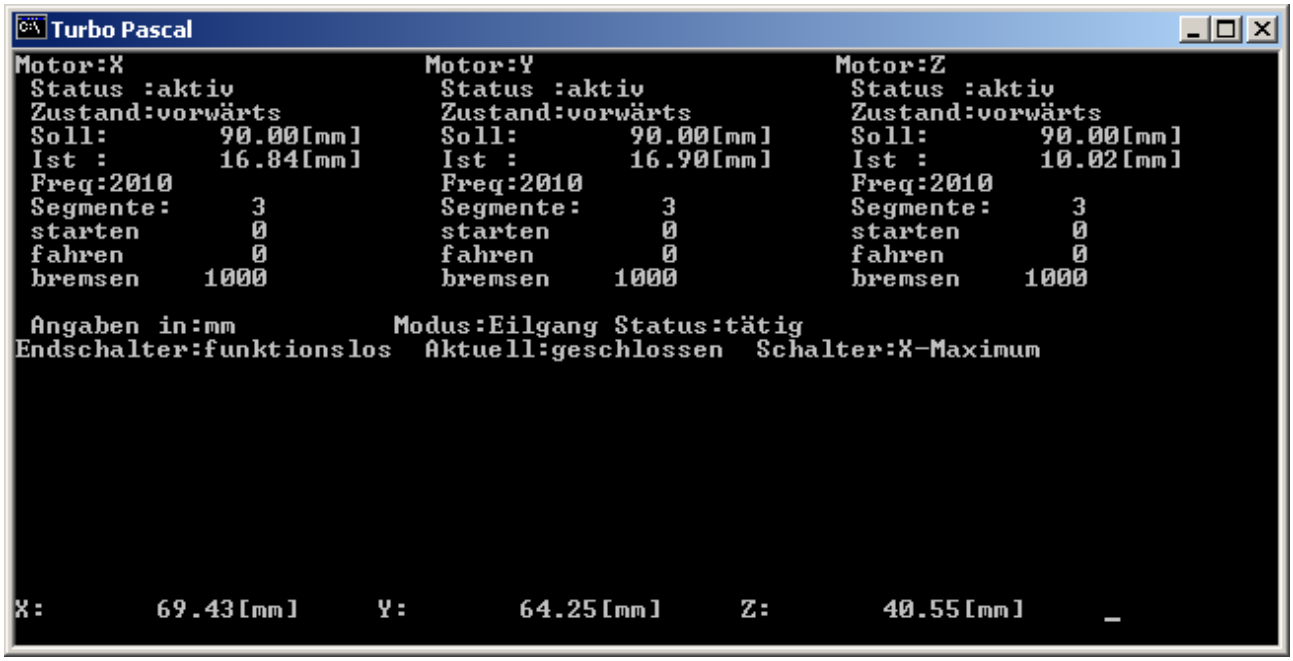
Noch ein Wort zu den Ein/Ausgabefunktionen: Die Verwendung der Funktionen aus den TP Standardunits sind für die Steuerung nicht ganz geeignet – sie sind zu langsam. Vor allem die Tastatureingabe stört die Ansteuerung der Motoren. Das Programm liest daher direkt aus dem Tastaturpuffer. Diese Funktionalität ist im Modul 'KEYBOARD.PAS' gekapselt. Die Realisierung erfolgte nach der Vorgabe der Unit KBDStuff aus Borland Pascal 7 (siehe /7/). Auch die Anzeige stört und sollte durch Funktionen die direkt in den Bildspeicher schreiben ersetzt werden – bis auf Weiteres habe ich mir damit geholfen, dass ich im Betrieb die Anzeige selten bis gar nicht aktualisiere. Damit die Bewegung der Motoren visualisiert werden kann, wurde eine Minimalanzeige ' ShowPos' angelegt, die relativ wenig stört.



Schirmbild: Datenanzeige und Anzeige der Kommandos



Schirmbild: Anzeige ohne Kommandos aber mit Positionsanzeige



Schirmbild: Positionierung mit Bahndaten und Positionsanzeige

Der Steuerrahmen:

Der Steuerrahmen im Modul FRAESE.PAS ist nur ein Demonstrationsrahmen. Er ist für die eigentliche Motorsteuerung und das Verständnis derselben nicht mehr notwendig, daher wird hier auch nur sehr kurz darauf eingegangen.

Dieses Modul soll die Funktionalitäten, die für eine CNC-Maschinenansteuerung notwendig werden exemplarisch zeigen:

- Abarbeiten von Kurven
- Kalibrieren der Maschine
- Werteumrechnungen

Außerdem soll eine Möglichkeit demonstriert werden, wie Anzeige und Eingabefunktionen in die Steuerung eingehängt werden können, ohne von diesen Funktionen abhängig zu werden – also die Möglichkeit, die Steuerung klar von der GUI (Graphical User Interface) zu trennen. Realisiert ist allerdings nur eine Textschnittstelle kein grafisches Interface – die aber leicht ausgetauscht werden kann.

Verewendete Literatur und Verweise:

/1/ Grundlagen der NC-Programmiertechnik Helmut Benkler Carl Hanser Verlag ISBN 3-446-17434-6

/2/ Mit Schrittmotoren steuern, regeln und antreiben Felix Schörlin Franzis-Verlag ISBN 3-7723-6723-2

/3/ http://www.webplain.de/turbopascal/downloads.php

/4/ http://www.conrad.com/

/5/ Chip Spezial TurboPascal Nr 16 1990 ISBN 3-8023-1072-1

/6/ http://g-spur.magix.net/website/

/7/ Borland Pascal 7 Programmierung Kiriakos Georgiadis IWT Verlag GmbH ISBN 3-88322-432-4

/8/ http://en.wikipedia.org/wiki/Intel_8253