Erste Erfahrungen mit Forth

CLAUS KÜHNEL

Mitteilung aus dem Zentralinstitut des Sportmedizinischen Dienstes Kreischa

In zunehmendem Maße sind in der internationalen Literatur Beiträge über die Programmiersprache Forth zu finden. Da die Meinungen über sie weit auseinandergehen, wurde beim Autor das Interesse an dieser Sprache geweckt. Erste Eindrücke und Erfahrungen sollen im folgenden Beitrag vermittelt werden.

Forth ist sehr verschieden von den gebräuchlichen höheren Programmiersprachen wie Pascal oder Basic. Je nach Standpunkt läßt sich Forth als höhere Programmiersprache, als Betriebssystem, als Interpreter bzw. interaktiver Compiler, als erweiterbare Datenstruktur und auch als Softwarekonzept auffassen [1]. Wie der Zusammenstellung der häufigsten Forth-Befehle entnommen werden kann, arbeitet Forth mit recht eigenwilligen Befehlskürzeln. Hinzu kommen die Umgekehrt Polnische Notation (UPN), wie sie auch von einer Reihe von Taschenrechnern benutzt wird, und die konsequente Ausrichtung auf einen Datenstack.

Die zunehmende Bedeutung der seit Anfang der siebziger Jahre bekannten Programmiersprache wird in der hohen Strukturiertheit und der nahezu beliebigen Erweiterbarkeit durch den Anwender gesehen. Wer Befehle kreieren will, die über den im Anhang angegebenen Umfang hinausgehen, hat ein weites Betätigungsfeld. Der entstehende Kode ist kompakt, wodurch Forth gute Laufzeiteigenschaften besitzt, die sie für Echtzeitanwendungen tauglich machen.

Folgende Eigenschaften kennzeichnen Forth [2]:

- maschinennahe, interaktive Hochsprache
- Umgekehrt Polnische Notation (UPN)
- Push-down-Pull-up-Datenstack
- streng strukturiert (weder GOTOs noch LABELs)
- gute Portabilität infolge guter Standardisierung
- gute Laufzeiteigenschaften (etwa zehnmal schneller als Basic)
- beliebige Erweiterbarkeit
- modulares Programmieren (Top-down planen, Bottom-up programmieren)
- schrittweises Testen der Moduln möglich
- Forth und Assembler beliebig mischbar, daher ideal für Echtzeitanwendungen
- 16-bit- und 32-bit-Integerarithmetik
- Kompaktheit (Forth, Assembler, Editor und Disk-Interface meist innerhalb von 8 Kbyte).

Beim Programmieren von Forth sind im Gegensatz zu anderen Programmiersprachen die Bewegungen auf dem Datenstack ständig zu verfolgen. Die erfolgreiche Ausführung eines bestimmten Befehles ist an einen entsprechend vorbereiteten Datenstack gebunden, da die meisten Befehle ihre Operanden auf dem Datenstack erwarten und ihre Ergebnisse auch wieder dort ablegen. Wer bereits Erfahrungen mit der Umgekehrt Polnischen Notation bei Taschenrechnern hat, ist gut auf diese Eigenschaften von Forth vorbereitet.

Die Programmierung in Forth erfolgt im Dialog über Tastatur und Bildschirm. Eingaben werden, wie von anderen Computern gewohnt, mit RETURN bzw. NEWLINE abgeschlossen. Wird die Eingabe angenommen, d. h., die Befehls- bzw. Datenfolge wird als korrekt betrachtet, dann antwortet das System mit OK.

Eine einfache Befehlsfolge am Beispiel der Addition zweier 16-bit-Zahlen soll diese Aussagen verdeutlichen:

```
12345 6789 + . (NL) 19134 OK
```

Die beiden Operanden (12345 und 6789) besetzen die beiden oberen Plätze des Datenstacks. Durch + erfolgt eine Addition und durch . (Dot) die Ausgabe auf dem Bildschirm. Nach Abschluß dieser Befehls- bzw. Datenfolge durch Betätigung der Taste NEWLINE erfolgt die Ausführung der genannten Operationen. Die Gültigkeit wird durch das bereits erwähnte OK dokumentiert.

Wichtig zu vermerken ist die Trennung der einzelnen Elemente in einer solchen Befehlsfolge durch SPACE (ASCII-Kode 32). In der im Anhang gegebenen Zusammenstellung der häufigsten Forth-Befehle sind die Belegungen des Datenstacks vor und nach Ausführung des betreffenden Befehls enthalten. Betrachtet man die durch + gekennzeichnete Addition zweier 16-bit-Zahlen, so erkennt man, daß die Operanden (mit n1 und n2 bezeichnet) auf dem TOS (Top-of-Stack, obere Zelle) und dem SECOND (Zelle unterhalb TOS) erwartet werden und daß das Ergebnis (mit SUM bezeichnet) auf dem TOS abgelegt wird. Die Operanden n1 und n2 werden bei dieser Operation vom Stack entfernt. Mit dem Befehl . wird anschließend der TOS zerstörend gelesen. Ein weiteres Beispiel (Rotation der ersten drei Stackzellen) soll die Bewegungen auf dem Stack verdeutlichen:

```
12 OK
34 OK
56 OK
ROT . (NL) 12 OK
. (NL) 56 OK
. (NL) 34 OK
```

Nach Eingabe der dritten Zahl befindet sich die Zahl 56 auf dem TOS. Die Rotation schafft den THIRD (dritte Stackzelle) zum TOS und verschiebt die anderen Inhalte. Auf dem TOS steht nun die Zahl 12, welche dann auch als erste ausgegeben wird. Durch das zerstörende Lesen befindet sich anschließend die Zahl 56 auf dem TOS, die durch den zweiten Befehl . ausgegeben wird usw.

Im Gegensatz zum ersten Beispiel wurden hier die Befehle einzeln durch das Betätigen von NEWLINE abgeschlossen. Beide Möglichkeiten sind äquivalent.

```
12 34 56 ROT . . . (NL) 12 56 34 OK
```

Die Bildschirmausgaben wurden in allen Beispielen durch Unterstreichen kenntlich gemacht.

Nach der Darstellung der prinzipiellen Arbeitsweise soll der Gültigkeitsbereich der Zahlen in Forth betrachtet werden. Wie bereits erwähnt wurde, arbeitet Forth mit einer 16-bit- bzw. 32-bit-Integerarithmetik. Für eine vorzeichenlose 16-bit-Zahl erhält man einen Gültigkeitsbereich von $0 \leq u \leq 2^{16} - 1$, wogegen bei Berücksichtigung des Vorzeichens durch Darstellung der Zahl als Zweierkomplementzahl der Gültigkeitsbereich von $-2^{15} \leq n \leq 2^{15} - 1$ reicht. Äquivalente Ergebnisse erhält man natürlich für die doppeltgenauen 32-bit-Zahlen. Bereichsüberschreitungen bei arithmetischen Operationen werden durch Forth nicht gemeldet. Der Programmierer muß hier sehr aufmerksam vorgehen. Einige Beispiele sollen das Gesagte verdeutlichen:

```
32766 . 32766 OK
32767 . 32767 OK
32768 . -32768 OK
32769 . -32767 OK
```

Der Befehl . (Dot) druckt eine vorzeichenbehaftete 16-bit-Zahl aus. Für Zahlen größer 32767 ergibt sich damit eine Bereichsüberschreitung. Da das MSB (Most Significant Bit) mit 1 belegt ist, wird eine negative Zahl ausgegeben. Anhand der binären Zahlendarstellung (und nur die kennt der Computer) kann man sich diesen Sachverhalt sehr schnell und einfach verdeutlichen.

Verlangt eine Applikation das Verlassen des Bereiches der 16-bit-Zahlen, sind einfache Überführungen vom 16- in den 32-bit-Bereich möglich.

Bei den Vergleichsoperationen wird das Ergebnis durch ein Flag repräsentiert. Nun existiert aber in Forth kein Flagregister, sondern der TOS übernimmt diese Aufgabe. Ein Flag ist in Forth damit ein 16-bit-Muster, das lediglich die Zustände True (wahr) und False (falsch) markiert. Der Vorteil einer solchen Auslegung des Flagbegriffes ist, daß Rechenergebnisse selbst direkt als Flag gedeutet werden können.

Die speicherbezogenen Befehle ermöglichen einen direkten Zugriff auf die Ressourcen der Maschine. Speicherinhalte lassen sich auf diese Weise frei manipulieren.

Im Zusammenhang mit der Möglichkeit der Definition von Anwenderworten sollen diese Befehle und die schon angedeuteten Möglichkeiten der Stackmanipulation verdeutlicht werden. Vorteilhaft läßt sich das aus [3] übernommene Formular zur Programmierung verwenden (Bild 1). Zur Demonstration der Definition von Anwenderworten soll die Umrechnung von Altgrad in Neugrad verwendet werden. Bekanntlich umfaßt der Vollkreis in Altgrad 360°, in Neugrad hingegen 400°. Die Umrechnung läßt sich dementsprechend durch eine Multiplikation mit dem Faktor 40/36 vornehmen. Im Bild 1 sind die erforderlichen Programmierschritte eingetragen. Die Definition eines Anwenderwortes wird durch das Wort : (Colon) eingeleitet. Durch :

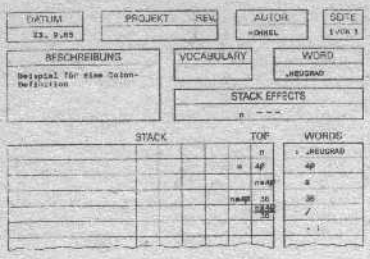


Bild 1: Programmierformular nach [3]

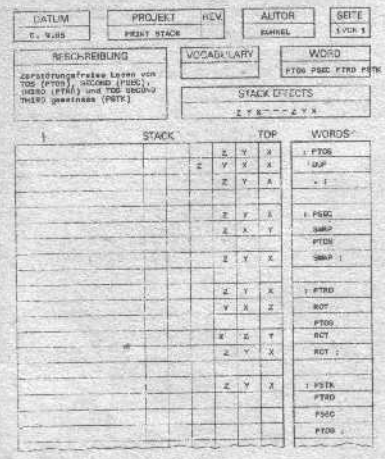


Bild 2: Lesen von Stackzellen

```
OK
100 LIST
SCR # 100
  0 ( STACK PRINTING )
  1
  2 ( PRINT TOS )
  3 : PTOS DUP . ;
  4
  5 ( PRINT SECOND )
  6 : PSEC SWAP PTOS SWAP ;
  7
  8 ( PRINT THIRD )
  9 : PTRD ROT PTOS ROT ROT ;
 10
 11 ( PRINT THIRD SECOND TOS )
 12 : PSTK PTRD PSEC PTOS ;
 13
 14
 15
OK
_

OK
4 3 2 1 OK
PTOS 1 OK
PSEC 2 OK
PTRD 3 OK
PSTK 3 2 1 OK
```

Bild 3: Listing von SCR # 100 und Beispiel

wird der Compiler aufgerufen, der einem frei wählbaren Namen (hier .NEUGRAD) eine Befehlsfolge zuordnet. Der Abschluß dieser als Colondefinition bezeichneten Definition eines Anwenderwortes wird durch das Wort ; (Semi-Colon) gebildet. Erkennt der Computer die vorgenommene Definition als gültig an, antwortet er wieder mit OK und verläßt den Compilermodus. Der Befehlssatz ist durch die vorgenommene Colondefinition um ein Wort erweitert.

```
: .NEUGRAD 40 * 36 / . ; OK
90 .NEUGRAD 100 OK
```

Mit der hier an einem sehr einfachen Beispiel vorgestellten Colondefinition wurde in [3] ein „Double Number Package" erstellt. Die Befehle entsprechen den in der Zusammenstellung enthaltenen Befehlen, nur daß 32-bit-Zahlen verarbeitet werden.

Für das Experimentieren mit Forth und die dazu erforderliche Beobachtung des Datenstacks wurden die im Bild 2 angegebenen Worte definiert. Betrachtet werden die drei oberen Stackzellen (TOS, SECOND, THIRD). Durch die Operationen darf der Stackinhalt nicht verändert werden. Im Feld STACK EFFECTS wird das durch die gewählte Beschreibung verdeutlicht. Damit der TOS zerstörungsfrei durch gelesen werden kann, wird er vor der Ausgabe durch DUP dupliziert. Die Bewegungen auf dem Stack können deutlich verfolgt werden. Beim Lesen des SECOND (PSEC) werden TOS und SECOND durch SWAP vertauscht. Es schließt sich das Lesen des TOS durch PTOS an. Um die ursprüngliche Ordnung auf dem Stack wieder herzustellen, ist ein weiteres SWAP erforderlich. Der THIRD läßt sich durch eine Rotation ROT der ersten drei Stackzellen nach dem TOS bringen (PTRD). Das Lesen kann nun wieder durch PTOS erfolgen. Zweimaliges Rotieren stellt den Ausgangszustand auf dem Stack her. Unter Verwendung der vorstehend definierten Worte PTOS, PSEC und PTRD wurde nun noch das Wort PSTK (Print Stack) definiert. Durch Aufruf von PSTK werden die drei oberen Stackzellen zerstörungsfrei gelesen und auf dem Bildschirm dargestellt.

Bevor auf weitere Anwenderdefinitionen eingegangen wird, sollen Möglichkeiten für deren Abspeicherung betrachtet werden. In Forth wird der Massenspeicher in Blöcke zu je 128 byte (entspricht einem Floppy-Disk-Sektor) eingeteilt. Acht solcher Blöcke bilden einen Screen, der durch 16 Zeilen mit maximal 64 Zeichen auf dem Bildschirm dargestellt wird. Somit ist ein Screen die minimale handhabbare Informationsmenge eines Umfanges von 1 Kbyte. Große Programme lassen sich in eine Vielzahl von Screens aufteilen, die dann nach Bedarf geladen werden und somit relativ geringer Speicheraufwendungen bedürfen. Der Inhalt einmal erstellter Screens kann mit Hilfe des Editors beliebig geändert werden. Die Editierung ist etwas unterschiedlich gelöst und soll an dieser Stelle nicht weiter betrachtet werden. Die im Bild 2 definierten Worte wurden dem Screen:SCR#100 zugeordnet. Bild 3 zeigt dessen durch 100 LIST ausgegebenen Inhalt. In ( ) eingeschlossene Kommentare verdeutlichen Programminhalte. Ein im Anschluß angegebenes Beispiel verdeutlicht die Arbeit mit den definierten Worten. Mit Hilfe der (wenn auch knappen) Erläuterungen zur Arbeit mit einfachen Befehlen und der Zusammenstellung der häufigsten Forth-Befehle kann der Leser einen Eindruck zur Programmierung gewinnen. Wesentlicher Bestandteil aller Programme sind die strukturierten Worte, die den Schleifenaufbau und Entscheidungen ermöglichen. Im Bild 4 sind das Anwenderwort .ASCII und der betreffende Ausdruck für den ASCII-Zeichensatz dargestellt. Kernstück ist eine Schleife (DO LOOP), welche Start- und Stoppwert auf dem Stack erwartet. Durch das Wort I wird der Schleifenindex (der momentane Stand des Schleifenzählers) auf den Stack

```
OK
105 LIST
SCR # 105
  0 ( DRUCKEN DES ZEICHENSATZES )
  1
  2 : .ASCII
  3    CR CR
  4    DO
  5     I DUP
  6     5 .R      ( DRUCKE INDEX )
  7     2 SPACES ( DRUCKE 2 SPACES )
  8     EMIT      ( DRUCKE ASCII)
  9    LOOP
 10    CR ;
 11
 12
 13
 14
 15
OK
_

OK
91 32 .ASCII

   32      33 !    34 "    35 #
   36 $    37 %    38 &    39 '
   40 (    41 )    42 *    43 +
   44 ,    45 -    46 .    47 /
   48 0    49 1    50 2    51 3
   52 4    53 5    54 6    55 7
   56 8    57 9    58 :    59 ;
   60 <    61 =    62 >    63 ?
   64 @    65 A    66 B    67 C
   68 D    69 E    70 F    71 G
   72 H    73 I    74 J    75 K
   76 L    77 M    78 N    79 O
   80 P    81 Q    82 R    83 S
   84 T    85 U    86 V    87 W
   88 X    89 Y    90 Z
OK
_
```

Bild 4: Drucken des ASCII-Zeichensatzes mit DO... LOOP

```
: DIV DUP 5 SPACES
    IF / . ELSE
      ." DIVISOR=0 "
    ENDIF
    CR ;


OK
6 3 DIV     2
OK
6 0 DIV     DIVISOR=0
OK
```

Bild 5: Prüfung auf Division durch Null mit IF... ELSE...ENDIF

geholt. Durch 5 .R wird dieser Index rechtsbündig in ein fünf Stellen breites Feld gedruckt. Anschließend werden zwei Leerstellen (Space) ausgegeben. Diesen folgt die Ausgabe des durch den Schleifenindex I gekennzeichneten ASCII-Zeichens durch EMIT. Mit LOOP ist die Schleife beendet. Das folgende CR sorgt dafür, daß das zu erwartende OK auf der nächsten Zeile steht.

Als weiteres Wort dieser Kategorie soll noch IF...ELSE...ENDIF betrachtet werden. In Abhängigkeit von einem auf dem TOS befindlichen Flag werden die Teile dieses Befehls in der Form IF truepart ELSE falsepart ENDIF abgearbeitet. Das im Bild 5 gezeigte Divisionsbeispiel zeigt Programm und Ablauf. Für die definierte Divisionsanweisung DIV werden die Operatoren auf dem Stack erwartet. Da der TOS von IF als Flag gesehen wird, muß der Divisor durch DUP gerettet werden. Das von IF vorgefundene Flag wird zerstört, so daß nun wieder die Operatoren auf TOS

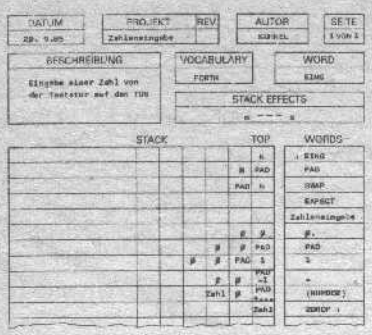


Bild 6: Zahleneingabe auf den TOS mit EING

```
OK
102 LIST
SCR # 102
  0 ( DRUCKEN DER ZAHLENBASIS )
  1
  2 : .BASE BASE @ DUP DECIMAL .
  3         BASE ! ;
  4
  5
  6 ( SETZEN BELIEBIGER BASIS )
  7 ( BASIS ---                 )
  8
  9 : SBASE BASE ! ;
 10
 11
 12 ( BINAERE BASIS )
 13
 14 : BINARY 2 BASE ! ;
 15
OK
-

OK
DECIMAL OK
.BASE 10 OK
HEX OK
.BASE 16 OK
3 SBASE OK
.BASE 3 OK
BINARY OK
.BASE 2 OK

DECIMAL OK
0 0 255 PSTK 0 0 255 OK
HEX PSTK 0 0 FF OK
8 SBASE PSTK 0 0 377 OK
3 SBASE PSTK 0 0 100110 OK
BINARY PSTK 0 0 11111111 OK
DECIMAL PSTK 0 0 255 OK
```

Bild 7: Listing SCR # 102 (Worte zur Zahlenbasis und Beispiele)

und SECOND zu finden sind. Wurde durch IF ein Flag = 0 vorgefunden, erfolgt die Abarbeitung des False Parts, das ist in diesem Fall die Ausgabe der Fehlermeldung DIVISOR = 0. Anderenfalls werden die beiden Operatoren entsprechend dem True Part dividiert, und das Ergebnis wird auf dem Bildschirm ausgegeben.

Neben den Operationen, die im Rechner direkt ablaufen, sind die Befehle zur Ein- und Ausgabe über das Computerterminal von besonderem Interesse. Bildschirmausgaben in der Form . und .R waren bereits in den Beispielen enthalten und bedürfen keiner weiteren Erläuterung. Das gleiche gilt für die Ausgabe von Leerzeichen und die Zeilenschaltung (CR). Eine einfache Tastaturbetätigung kann durch KEY (ähnlich dem INKEY in Basic) ausgewertet werden. Definiert man beispielsweise:

```
: .KEY KEY . ;
```

erhält man durch Eingabe von .KEY (NL) und Drücken der Taste A folgende Bildschirmausschrift:

```
.KEY 65 OK
```

Auf dem TOS steht demzufolge der der Tastenbetätigung entsprechende ASCII-Kode.

Zur Eingabe einer Zahl auf den TOS existiert in Forth kein der INPUT-Anweisung von Basic entsprechender Befehl. Natürlich sind in Forth alle Mittel zur Schaffung eines solchen Befehles enthalten. Im Bild 6 ist die Definition des Befehles „EING" mit den entsprechenden Stackbelegungen dargestellt. Auf dem TOS wird eine Längenangabe der einzugebenden Zahl erwartet. Das Wort EXPECT interpretiert die Eingabe als String und benötigt für die Abspeicherung die Stringlänge. Kann die Länge nicht von vornherein angegeben werden, gibt man die größtmögliche Länge vor und schließt die Eingabe mit NL ab. Für die Abspeicherung werden dann nur die Stellen bis NL herangezogen.

PAD ist ein temporärer Speicher, der zur Zwischenspeicherung des eingegebenen Strings verwendet wird. Damit EXPECT die betreffenden Operatoren in der richtigen Reihenfolge bekommt, ist das Vertauschen von TOS und SECOND durch SWAP vor EXPECT erforderlich. Ist nun die Eingabe der Zahl durch NL oder durch Erreichen der vorgegebenen Stellenzahl (dann ist kein NL mehr erforderlich) abgeschlossen, befindet sich der Ziffernstring in den Adressen PAD bis PAD+n. Die Umwandlung des Strings in einen numerischen Wert besorgt das Wort (NUMBER). Durch 0. wird eine 32-bit-Zahl mit Null belegt. Des weiteren benötigt (NUMBER) den Speicherplatz unterhalb des Strings, d. h. PAD-1, was durch PAD 1 – gesichert ist. Nach Abschluß der Umwandlungsprozedur steht die eingegebene Zahl auf dem THIRD. SECOND und TOS sind nicht von weiterem Interesse, weshalb sie durch 2DROP beseitigt werden. Im Ergebnis dieser relativ umfangreichen Operation steht die eingegebene Zahl zur weiteren Verarbeitung auf dem TOS zur Verfügung. Mag diese Art der Eingabeorganisation anfänglich umständlich erscheinen, hat man jedoch alle Modifikationsmöglichkeiten in der Hand und ist nicht an die starren Eingabebedingungen anderer Programmiersprachen gebunden. Zur Führung eines Dialoges mit dem Computer ist die Verarbeitung von Strings (von alphanumerischen Zeichenketten) erforderlich. Ein voll ausgebauter Stringverarbeitungs-Befehlssatz existiert in Forth nicht. Mit den bestehenden Worten läßt sich jedoch ohne weiteres ein Stringvokabular aufbauen. In [1] wurde das in einem dem Basic entsprechenden Umfang getan.

Einfache Stringverarbeitung kann durch EXPECT, TYPE und das bereits verwendete ."(Dot-Quote) vorgenommen werden. Das folgende Beispiel soll das verdeutlichen. Definiert werden die beiden Worte DATUM und .DATUM.

```
: DATUM CR ." DATUM ? (TT.MM.JJ) "
  PAD 8 EXPECT CR ;
: .DATUM CR ." DAS EINGEGEBENE
  DATUM WAR DER " PAD 8 TYPE CR ;
```

Auf dem Bildschirm wird man bei Aufruf

```
OK
103 LIST
SCR # 103
  0 ( HEX-DUMP ZECH:FORTH 1983 )
  1 (STARTADR. STOPADR.---      )
  2
  3 : DUMP CR CR BASE @ HEX ROT
  4        ROT 1+ SWAP
  5        DO
  6         I 0 8 D.R 3 SPACES
  7         3A EMIT SPACE SPACE
  8         I 4 0
  9         DO
 10          DUP I + C@ 4 .R
 11         LOOP
 12         DROP CR
 13        4 +LOOP
 14        BASE ! ;
 15
OK
-

OK
16384 16448 DUMP

    4000  :    FF   C8   FC   7F
    4004  :     0   80    0    1
    4008  :     0    0    2    0
    400C  :    F0   73   4B   76
    4010  :    86   76   8B   9F
    4014  :    87   76   8E   40
    4018  :    7D   BF   8F   76
    401C  :    87   76   43   5D
    4020  :    40   AA    2    0
    4024  :     0   FF   FF    0
    4028  :    37   8F   40   FE
    402C  :    FF    0   69   D2
    4030  :    87    C   5D   43
    4034  :    4F   FE    0    0
    4038  :    BC    F    3   C0
    403C  :     0    0    0    0
    4040  :     0    0    0    0
OK
```

Bild 8: Hexdump nach [1]

der beiden Worte nacheinander folgendes Bild erhalten:

```
OK
DATUM (NL)
DATUM ? (TT.MM.JJ) 27.09.85
OK
.DATUM (NL)
DAS EINGEGEBENE DATUM WAR DER
27.09.85
OK
```

Die von ." und " eingeschlossenen Strings werden nur auf dem Bildschirm zur Darstellung gebracht. Durch EXPECT wird wie bereits bei der Zahleneingabe ein acht Zeichen langer String nach PAD gebracht. Natürlich kann auch ein anderer Speicherbereich hierzu verwendet werden. Bei der Ausgabe des Strings durch TYPE wird wieder auf PAD zugegriffen.

Zur Formatierung der Ausgabe ist eine eigenständige Gruppe von Worten in Forth vorhanden. Damit lassen sich doppelt genaue Zahlen in Strings wandeln. Will man z. B. eine Telefonnummer in der gewohnten Schreibweise xxx xx xx ausgeben, kann man von der folgenden Definition Gebrauch machen:

```
: .TEL <# # # 32 HOLD # # 32 HOLD
  #S#) TYPE SPACE ;
```

Das Wort .TEL erwartet eine doppelt genaue Zahl auf dem Stack und bewirkt die folgende Bildschirmausschrift:

```
OK
1234567. (NL) OK
.TEL 123 45 67 OK
```

Die Elemente der Zahl werden einzeln durch # in die entsprechenden Stringelemente gewandelt. Mit HOLD lassen sich in den String Zeichen (hier SPACE) einfügen. Damit ergeben sich für die Ausgabeformatierung wesentlich vielfältigere Möglichkeiten gegenüber dem PRINT USING verschiedener Basicdialekte. Verwiesen werden soll an dieser Stelle auf [4], wo eine Reihe weiterer Formatierungsbeispiele enthalten sind. Bei den bislang vorgeführten Beispielen wurde generell in einem dezimalen Zahlensystem gearbeitet. Forth bietet hier aber wesentlich mehr. In Forth sind direkt das dezimale und das hexadezimale Zahlensystem verankert. Beliebige Zahlensysteme sind ohne weiteres definierbar. Betrachtet man den im Bild 7 gezeigten Screen # 102 und die entsprechenden Beispiele, werden die Möglichkeiten sofort deutlich. In der Zelle BASE ist die aktuelle Zahlenbasis abgespeichert. Will man die aktuelle Zahlenbasis auf dem Bildschirm zur Anzeige bringen (.BASE), ist diese durch BASE@ auf den TOS zu bringen. Zur späteren Rückspeicherung wird der TOS dupliziert. DECIMAL . druckt die aktuelle Zahlenbasis dezimal aus. Durch die vorangegangene Duplizierung steht die ursprüngliche Zahlenbasis wieder auf dem TOS und kann durch BASE ! wieder abgelegt werden. Das Setzen einer beliebigen Zahlenbasis kann, wie in SBASE realisiert, durch BASE ! vorgenommen werden. Wird das Dualsystem häufig benutzt, macht sich eine den Worten DECIMAL oder HEX entsprechende Definition BINARY bezahlt. Die Anwendungsbeispiele nach Bild 7 demonstrieren die Nutzung der soeben definierten Worte. Mit DECIMAL wird die Basis 10 gesetzt. HEX ruft zwangsläufig die Basis 16 hervor. Mit 3 SBASE wurde die Drei als Zahlenbasis gewählt, und BINARY läßt Zwei als Zahlenbasis erscheinen. Zur Demonstration der Zahlendarstellung in den unterschiedlichen Zahlensystemen wurde anschließend der TOS mit der Zahl 255 dezimal beschrieben. Nach Aufruf von HEX steht auf dem TOS FF (der TOS wurde zerstörungsfrei durch das eingangs definierte Wort PSTK gelesen). Ruft man durch 8 SBASE das Oc-

```
 5    REM BENCHMARK TEST
 10   FOR I=1 TO 4000
 20    LET A=10
 30   NEXT I
 40   PRINT A
 50   STOP
 60   FOR I=1 TO 4000
 70    LET A=10/3
 80   NEXT I
 90   PRINT A
```

Bild 9: Basic-Bench-Mark-Test

talsystem auf, muß 377 auf dem TOS stehen. Wählt man hingegen Drei als Zahlenbasis, erhält man 100110 und im Binärsystem 11111111.

Unter Zuhilfenahme unterschiedlicher Zahlensysteme läßt sich eine Reihe von Aufgabenstellungen sehr günstig programmieren. Umfangreiche Unterprogramme zur Zahlenkonvertierung gehören hier der Vergangenheit an. Als Beispiel ist das aus [1] entnommene Programm zum Listen eines beliebigen Speicherbereiches zu betrachten (Bild 8). Das Programm selbst soll an dieser Stelle nicht weiter erläutert werden. Der interessierte Leser dürfte mit einem Stackdiagramm (entsprechend dem Programmierformular) und der Befehlsb

Tafel 1: Laufzeiten der Bench-Mark-Tests nach Bild 9 und Bild 10

	Forth	Basic
BMT1	1 s	24 s
BMT2	3,5 s	37 s

Tafel 2: Die häufigsten Forthbefehle

```
Operanden: n, n1...16-bit-Kompl.zahl     Stackbewegungen:
           d,d1...32-bit-Kompl.zahl      Ein-und Ausgaben immer von links nach
                                         rechts.
           u,u1...16-bit-Zahl o.Vz.      TOS steht stets rechts aussen.
           ud    32-bit-Zahl o.Vz.
           addr  Adresse
           b     8-bit-Byte              Second = Zahl unter dem TOS
           c     7-bit-ASCII             Third  = Zahl unter Second
           f     Boolsches Flag(16 bit)

Zahlensysteme:
DECIMAL          ( > )                   deklariert Dezimalsystem
HEX              ( > )                   deklariert Hexadezimalsystem
BASE             ( >addr)                Systemvariable,die Zahlenbasis enthaelt

Stackmanipulationen:
DUP              (n>nn)                  dupliziert den TOS
-DUP             (n0>n)                  dupliziert den TOS,wenn ungleich Null
DROP             (n> )                   beseitigt den TOS
SWAP             (n1 n2 > n2 n1)         vertauscht die beiden oberen Eintraege
OVER             (n1n2 > n1n2n1)         kopiert den Second zum neuen TOS
ROT              (n1n2n3 > n2n3n1        rotiert den Third zum TOS
>R               (n > )                  bringt den TOS zum Return-Stack
R>               ( > n)                  holt Wert von Return-Stack zum TOS
R                ( > n)                  kopiert den Return-Stack-Top zum TOS

Arithmetik:
+                (n1n2 > SUM)            Addition (16-bit-Summe)
D+               (d1d2 > SUM)            Addition (32-bit-Summe)
-                (n1n2 > DIFF)           Subtrakt.(16-bit-Differenz)
*                (n1n2 > PROD)           Multipl. (16-bit-Produkt)
/                (n1n2 > QUOT)           Division (16-bit-Quotient)
MOD              (n1n2 > REST)           Modulo-Division
/MOD             (n1n2 > REST QUOT)      Division mit Rest und Quotient
*/MOD            (n1n2n3 >REST QUOT)     Multiplikation mit anschl. Division bei
                                         32-bit-Zwischenergebnis
*/               (n1n2n3 > QUOT)         wie */MOD ohne REST
M/MOD            (udu2 > u3ud4)          Div. einer 32-Bit-Z.d.v. mit Uebergabe
                                         des 16-bit-Restes u. des 32-bit-Quotient
MIN              (n1n2 > MIN)            kleiner Zahl nach TOS
MAX              (n1n2 > MAX)            groessere Zahl nach TOS
ABS              (n > u)                 Absolutwert einer 16-bit-Zahl
DABS             (d > ud)                Absolutwert einer 32-bit-Zahl
MINUS            (n > -n)                Vorzeichenwechsel einer 16-bit-Zahl
DMINUS           (d > -d)                Vorzeichenwechsel einer 32-bit-Zahl
1+               (n > n+1)               incrementiert TOS mit Eins
2+               (n > n+2)               incrementiert TOS mit Zwei

Logische Befehle:
AND              (n1n2 > log.UND)        bitweise log. UND
OR               (n1n2 > log.ODER)       bitweise log. ODER
XOR              (n1n2 > log.EXOR)       bitweise log. Exklusiv-ODER

Vergleichsoperatoren:
<                (n1n2 > f)              Flag=1, wenn n1 kleiner n2
>                (n1n2 > f)              Flag=1, wenn n1 groesser n2
=                (n1n2 > f)              Flag=1, wenn n1 gleich n2
0<               (n > f)                 Flag=1, wenn TOS negativ
0=               (n > f)                 Flag=1, wenn TOS gleich Null

Speicherbezogene Befehle:
@                (addr > n)              ersetzt Adresse durch deren Inhalt
C@               (addr > b)              wie @, jedoch auf eine Byte
!                (n addr > )             speichere Second in die Adr. auf TOS
C!               (b addr > )             wie !, jedoch nur ein Byte
+!               (n addr > )             addiere Second zum Inhalt der Adr.auf
                                         TOS
CMOVE            (from to u > )          verschiebe u Bytes im Adressraum
FILL             (addr u b > )           fuelle u Bytes ab addr mit b
ERASE            (addr u > )             fuelle u Bytes ab addr mit Null
BLANKS           (addr u > )             fuelle u Bytes ab addr mit SPACE
TOGGLE           (addr b > )             EXOR Byte in addr mit Maske b
SP@              ( > addr)               uebergibt aktuelle SP-Position

Strukturierte Worte:
DO...LOOP        (n1 n2 > )              Schleife von n2 bis n1-1 mit Incr. 1
DO...+LOOP       (n1 n2 > )              wie DO...LOOP,jedoch beliebiges Incr.
I                ( > INDEX)              LOOP-Index > TOS
LEAVE            ( > )                   erzwingt Schleifenabbruch bei
                                         nächster Gelegenheit
IF...ENDIF       (f > )                  führt Befehle aus solange Flag=1
IF...ELSE...ENDIF (f > )                 dto., jedoch bei Flag=0 den ELSE-Teil

BEGIN...UNTIL    ( > f > )               Schleife mit Abbruch, falls Flag fuer
                                         UNTIL gleich Eins
BEGIN.WHILE.REPEAT ( > f > )             dto., jedoch Abbruchtest am Schleifen-
                                         anfang
BEGIN...AGAIN    ( )                     Endlos-Schleife

Terminal Eingabe/Ausgabe:
.                (n > )                  druckt Zahl auf dem TOS (zerstoerend)
.R               (n Feldweite > )        druckt Zahl rechts adjustiert in Feldw.
D.               (d > )                  druckt doppeltgenaue Zahl
D.R              (d Feldweite > )        druckt 32-bit-Zahl rechts adj.in Feldw.
CR               ( )                     Ausgabe CR/LF
SPACE            ( )                     Ausgabe SPACE
SPACES           (n > )                  Ausgabe von n SPACEes
TYPE             (addr u > )             druckt u Zeichen ab addr
COUNT            (addr > addr+1 u)       wandelt Byte-String in TYPE-Form um
?                (addr > )               druckt den Inhalt von addr
?TERMINAL        ( > f)                  uebergibt den Tastaturstatus (1=betaet.)
KEY              ( > c)                  wartet auf Tastaturbetaet. und legt
                                         ASCII-Char. auf TOS
EXPECT           (addr n > )             erwartet n Char. (oder bis CR) und
                                         bringt sie nach addr...
EMIT             (c > )                  gibt Char. c aus
WORD             (c > )                  liest ein Wort im Eingabe-Puffer

Eingabe/Ausgabe-Formatierung:
NUMBER           (addr > d)              wandelt den String in addr um in
                                         eine 32-bit-Zahl
<#               ( )                     eroeffnet Zahlenwandlung fuer Ausgabe
#                (d > d)                 wandelt naechste Stelle der Zahl und
                                         fuegt sie Ausgabestring eine Ziffer zu
#S               (d > 00)                wandelt alle sign. Stellen in String
SIGN             (n d > d)               fuegt bei VZ von n in den Zifernstring
#>               (d > addr u)            beendet Umwandlung in Zifernstring
HOLD             (c > )                  Einfuegg eines ASCII-Char. in den String

Massenspeicher (Disc):
LIST             (screen > )             Ausdrucken eins Screens von Disc
LOAD             (screen > )             Laden eins screens
BLOCK            (block > addr)          liest Disc-Block nach addr
B/BUF            ( > n)                  Systemkonstante (Blockgroesse in Bytes)
BLK              ( > addr)               Systemvariable (akt. Blocknummer)
SCR              ( > addr)               Systemvariable (akt. Screennumber)
UPDATE           ( )                     markiert letzten Buffer als "updated"
FLUSH            ( )                     schreibt alle "updated" Buffer auf Disc
EMPTY-BUFFERS    ( )                     markiert alle Buffer als "leer"

Definitionsworte:
: xyz            ( )                     Beginn einer Colon-Def. mit Namen xyz
;                ( )                     Abschluss einer Colon-Def.
VARIABLE xxx     (n > )                  erzeugt eine Variable xxx, die mit n
                                         initialisiert ist
CONSTANT yyy     (n > )                  erzeugt eine Konstante yyy mit Wert n
CODE zzz         ( )                     eroeffnet die Def. eines Primitive mit
                                         Namen zzz
;CODE            ( )                     Abschluss einer Colon-Def.
<BUILDS...DOES>  duerf( >addr)           wird zur Def. neuer Def.worte benutzt

Vokabulare:
CONTEXT          ( > addr)               uebergibt die Adr. eines Pointers zum
                                         CONTEXT-Vokabular
CURRENT          ( > addr)               uebergibt die Adr. eines Pointers zum
                                         CURRENT-Vokabular
FORTH            ( )                     Name des Haupt-vokabulars
EDITOR           ( )
ASSEMBLER        ( )                     weitere Vokabularnamen
DEFINITIONS      ( )                     macht das CURRENT- zum CONTEXT-Vokab.
VOCABULARY xyz   ( )                     definiert neues Vok. mit Namen xyz
VLIST            ( )                     druckt die Namen aller Worte im CONTEXT-
                                         Vokabular

System-Worte und Diverses:
(komentar)       ( )                     Kommentaret hinter ( immer ein SPACE !
FORGET abc       ( )                     vergisst alle Def. ab abc
ABORT            ( )                     erzwingt Abbruch einer Operation
'xxx             ( > addr)               findet addr des Wortes xxx in Dictionary
HERE             ( > addr)               uebergibt die adr des naechsten freien
                                         Platzes im Dic.
PAD              ( > addr)               uebergibt Startadresse eines Zwischensp.
IN               ( > addr)               Systemvariable,haelt Input-Puffer-Offset
                                         fuer WORD
ALLOT            (n > )                  hinterlaesst eine ungenutzte Luecke im
                                         Dic.
,                (n > )                  compiliert eine Zahl in das Dic.
```

```
OK
200 LIST
SCR # 200
  0 ( BENCHMARK TEST )
  1
  2 0 VARIABLE AA
  3 0 VARIABLE BB
  4
  5 : BMT1 4001 1
  6        DO
  7         10 AA !
  8        LOOP
  9        AA @ . ;
 10
 11 : BMT2 4001 1
 12        DO
 13         10 3 /MOD AA ! BB !
 14        LOOP
 15        AA @ . BB @ . ;
OK
-

BMT1 10 OK
BMT2 3 1 OK
OK
```

Bild 10: Forth-Bench-Mark-Test

schreibung in der Zusammenstellung die einzelnen Programmschritte verfolgen können. Das Programm hat sich als sehr nützlich erwiesen und sollte bei späterer Arbeit unter Forth abgespeichert zur Verfügung stehen.

Den Abschluß der Betrachtungen soll ein Beispiel bilden, welches die Geschwindigkeitsrelationen zwischen Basic und Forth verdeutlichen soll. Für die Beantwortung derartiger Fragestellungen werden sog. Bench-Mark-Tests hinzugezogen. Da die anwenderspezifischen Forderungen an einen Computer sehr unterschiedlich sind, wird es einen allgemeinen Test zur Charakterisierung von bestimmten Eigenschaften nicht geben. Selbst wenn man unterschiedliche Dialekte einer Programmiersprache vergleichen will, kann man sehr schnell auf diverse Schwierigkeiten stoßen. In [5] wurden Geschwindigkeit und Genauigkeit verschiedener Computer mit Hilfe eines kleinen Basicvierzeilers getestet. Unter Berücksichtigung der genannten Einschränkungen lassen sich einige allgemeine Ableitungen treffen. Aus diesem Grund wurde dieses kleine Programm auch zur Charakterisierung der Laufzeiteigenschaften von Basic und Forth benutzt. Herz des verwendeten Kleincomputers war eine CPU Z 80 A mit einer Taktfrequenz von 3 MHz. Bild 9 zeigt die beiden Basictests. In den Zeilen 10 bis 40 erfolgt eine einfache Wertzuweisung, der sich nach 4 000 Schleifendurchläufen das Ausdrucken des Wertes der Variablen anschließt. In den Zeilen 60 bis 90 ist zusätzlich eine Gleitkommadivision enthalten. Einen etwa gleichwertigen Test in Forth zeigt Bild 10. In den Zeilen 2 und 3 werden die Variablen AA und BB definiert. Beide Variablen werden mit Null initialisiert. Im Test BMT1 erfolgt wie beim äquivalenten Basictest eine Wertzuweisung für die Variable AA. Dieser Wert wird wiederum nach 4 000 Schleifendurchläufen auf den TOS geholt und ausgedruckt. Im Test BMT2 nehmen die Variablen AA und BB Quotient und Rest der Division /MOD an. Die Verarbeitungszeiten sind in Tafel 1 einander gegenübergestellt.

Unter Berücksichtigung der vorhandenen Unterschiede in den betreffenden Tests kann dennoch das wesentlich bessere Laufzeitverhalten der Programmiersprache Forth ersehen werden. Die Verknüpfung mit Assembler (die hier nicht betrachtet werden konnte) wird weitere Verbesserungen nach sich ziehen.

In der Tafel 2 sind die häufigsten Forthbefehle zusammengefaßt.

Literatur

[1] Zech, R.: Die Programmiersprache FORTH. München: Franzis-Verlag 1984

[2] Briner, R. G.: Ein- und Ausgabe in FORTH. Elektroniker, Aarau 23 (1984) 3, S. 56–60

[3] Briner, R. G.: Double Number Words in FORTH. Elektroniker, Aarau 22 (1983) 18, S. 49–55

[4] Briner, R. G.: Ein- und Ausgabe in FORTH. Elektroniker, Aarau 23 (1984) 6, S. 49–56

[5] Feichtinger, H.: Basic-Benchmarks: Genauigkeit kostet Zeit. mc, München 3 (1983) 6, S. 28 und 29

Analogwerteingabe in Kleincomputer

Dipl.-Ing. KLAUS KRÜGER

Der Beitrag beschreibt die Eingabe von Analogsignalen über einen 8-bit-A-D-Wandler. An einem Beispiel, der Abbildung von periodisch oder einmalig verlaufenden Vorgängen im Bereich von 0 bis 2 kHz auf einem Bildschirm, soll dieser Vorgang näher untersucht werden.

Der Einsatz von Kleincomputern ist recht vielschichtig und kann bei Vorhandensein einer Systembusschnittstelle als programmierbare Steuereinheit für periphere Geräte verwendet werden.

Durch Kopplung von höherer Programmiersprache mit Maschinensprache können in kurzer Zeit recht komfortable Programme erstellt werden. Auf Grund der Leistungsmerkmale eines Kleincomputers ergeben sich über den Heimbereich hinaus auch interessante Einsatzmöglichkeiten zur Nutzung im Betrieb, u. a. zur Rationalisierung der automatischen Prüf- und Meßtechnik für Kleinserien. Zu steuernde Quellen könnten dabei programmierbare Generatoren, programmierbare Spannungsquellen, programmierbare Fensterdiskriminatoren zur Gut-Schlecht-Bewertung und Gleichstrom- und Wechselstrom-Meßwerterfassungsanlagen sein.

Auf der Basis eines Kleincomputers mit U-880-Prozessor und einer geeigneten Hardwareerweiterung soll ein Speicheroszillograf für langsam verlaufende Zeitabläufe beschrieben werden.

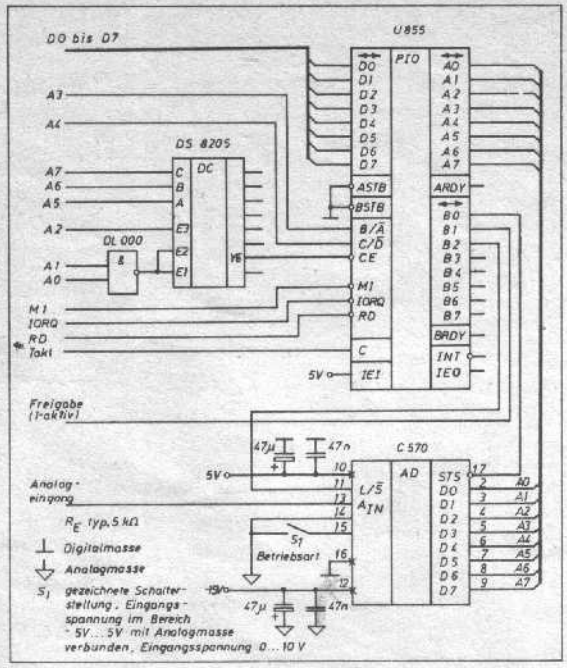
Schaltungsaufbau

Der Schaltungsaufbau ist Bild 1 zu entnehmen. Kernstück der Hardwareerweiterung ist ein 8-bit-A-D-Wandler vom Typ C 570. Er ist ein Anfalltyp des 10-bit-A-D-Wandlers C 571, der bei gleicher Funktion jedoch nur eine Auflösung von 8 bit garantiert [1]. Er wird mit Versorgungsspannungen von 5 V und —15 V betrieben, seine Ausgänge sind TTL-kompatibel. Gestartet wird der A-D-Umsetzprozeß mit einem H-L-Impuls an Pin 11. Einen Überblick über den zeitlichen Verlauf der Umsetzung gibt Bild 2

Programm für den Kleincomputer

```
BASICPROGRAMM
10 REM 3E 4F D3 . . . . .
20 POKE 16389,117
22 FAST
25 LET Q=USR 16514
30 LET B=30001
35 FOR N=0 TO 63
40 LET C=INT ((PEEK B)/5,93)
45 PLOT N,C
47 LET B=B+1
50 NEXT N
55 SLOW
60 STOP

MASCHINENPROGRAMM
4082 3E4F     LD  A,4FH
4084 D3D7     OUT(D7),A
4086 3E03     LD A,03H
4088 D3D7     OUT(D7),A
408A 3ECF     LD  A,CFH
408C D3DF     OUT(DF),A
408E 3E03     LD  A,03H
4090 D3DF     OUT(DF),A
4092 3E03     LD  A,03H
4094 D3DF     OUT(DF),A
4096 213075   LD HL,7530H
4099 0600     LD  B,00H
409B 3E04     LD  A,04H
409D D3CF     OUT(CF),A
409F DBCF     IN A,(CF)
40A1 FE07     CP  07H
40A3 20FA     JR NZ,FAH=>409FH
40A5 3E00     LD  A,00H
40A7 D3CF     OUT(CF),A
40A9 23       INC HL
40AA 04       INC B
40AB 78       LD A,B
40AC FE41     CP  41H
40AE 280F     JR Z,OFH=>40BFH
40B0 DBCF     IN A,(CF)
40B2 CB47     BIT 0,A
40B4 20FA     JR NZ,FAH=>40B0H
40B6 DBC7     IN A,(C7)
40B8 77       LD  (HL),A
40B9 3E04     LD  A,04H
40BB D3CF     OUT(CF),A
40BD 18E6     JR E6H=>40A5H
40BF C9       RET
```



◀ **Bild 1: Stromlaufplan der Hardwareerweiterung**

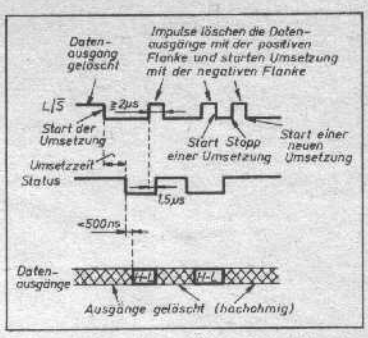


Bild 2: Überblick über zeitlichen Verlauf der Umsetzzeiten [1]

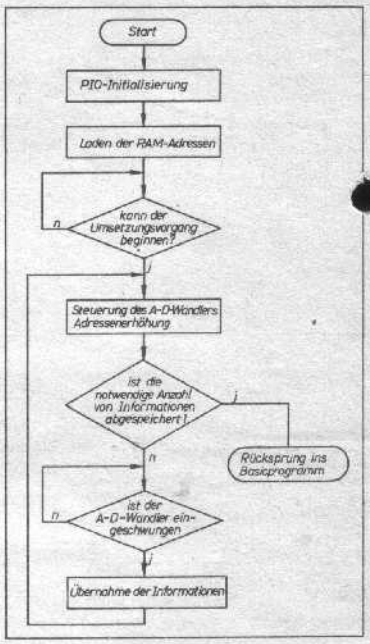


Bild 3: Programmablaufplan des Maschinenprogramms

wieder. Die typischen Umsetzzeiten liegen zwischen 15 µs und 40 µs. Im Mustergerät lag die Umsetzzeit bei 17 µs. Die Eingangsspannung beträgt je nach gewählter Betriebsart —5...5 V bzw. 0...10 V. Deshalb wird bei den meisten Anwendungsfällen ein Verstärker davor geschaltet werden müssen, [1] enthält dazu einen Schaltungsvorschlag.

Die Schnittstelle zwischen Kleincomputer und C 570 bildet die PIO. Teilweise verfügen die Kleincomputer bereits über eine integrierte, dem Nutzer zugängliche Schnittstelle. In diesem Fall kann die PIO-Erweiterung entfallen, wenn ihre Programmierung in das Maschinenprogramm mit einbezogen werden kann.

Die Schaltung enthält keine schaltungstechnischen Besonderheiten. Die Adressen ergeben sich bei dem gewählten Ausgang der IS DS 8205 zu:

C7 – Port A, Datenwort;
D7 – Port A, Steuerwort
CF – Port B, Datenwort;
DF – Port B, Steuerwort.

Software

Das Programm in der Tafel besteht aus einem kleinen Basicteil, in dem das Maschinenprogramm in einer REM-Zeile eingebunden wurde. Das hat den Vorteil, daß das Maschinenprogramm beim Abspeichern des Basicprogramms mit aufgezeichnet wird.

Mit Zeile 25 wird das Maschinenprogramm aufgerufen.

Der Ablauf ist dem Programmablaufplan im Bild 3 zu entnehmen. Um eine Information abzuspeichern, wurden 48 µs benötigt. Das Mustergerät bot eine Grafikmöglichkeit von 64 × 44 Bildpunkten. Um die gesamte Bildbreite zu nutzen, wurden 64 Eingabeinformationen im Speicher abgelegt (Programmlaufzeit etwa 3 ms).

Die Basiczeilen 25 bis 50 wurden zur Ausgabe der Informationen mit Hilfe der Grafikmöglichkeit des Kleincomputers benötigt. In Zeile 40 wurde der maximal mögliche Wert von 255 (FF) durch 5,93 dividiert, um die 44 Bildpunkte in der Vertikalen nicht zu überschreiten.

Mit dem Mustergerät konnten Wechselgrößen mit einer Frequenz bis 2 kHz noch recht gut abgebildet werden.

Langsam verlaufende Vorgänge konnten durch Verlängern der Programmlaufzeit erfaßt werden. Zu diesem Zweck wurden ins Maschinenprogramm Warteschleifen eingearbeitet bzw. periodische Sprünge ins Basicprogramm durchgeführt, um dort über den Befehl Pause den Abtastzyklus zu verlängern.

Möglichkeiten zur Verbesserung

Die Gebrauchswerte des beschriebenen Geräteteils lassen sich verbessern, wenn höhere Frequenzen bei besserer Auflösung erfaßt werden können.

Mit Hilfe einer Interruptroutine zur Steuerung des Pins 11 am C 570 und Programmumgestaltung zum Zwecke der Aufzeichnung nach dem Samplingverfahren kann dieser Forderung nachgekommen werden.

Die Grafikmöglichkeit des Mustergerätes war nicht zufriedenstellend. Ein moderner Kleincomputer, wie beispielsweise der KC 85/2, bietet mit seinem PSET-Befehl eine Grafikmöglichkeit von 320 × 255 Bildpunkten. Über ein in Basic geschriebenes Programm läßt sich auch der Zeitmaßstab mit abbilden, zusätzlich können Maxima, Minima und Effektivwerte ausgegeben werden.

Zusammenfassung

An einem Beispiel wurde gezeigt, daß mit geringem Aufwand steuerbare Meßmittel entstehen können, die sowohl im Beruf als auch im Heimbereich Anwendung finden.

Literatur

[1] Dietrich, J.; Kahl, B.: 10- und 8-bit-A-D-Wandlerschaltkreise C 571 D und C 570 D. Referate des 11. Mikroelektronik-Bauelemente-Symposiums 1985 in Frankfurt (Oder): Band 2, S. 341

[2] Schütz, M.: Datendrehscheibe. Funkschau, München 56 (1984) 3, S. 62–65; 4, S. 75; 5, S. 70–72