Ingenieurbüro / Vertrieb
Klaus Kohl
Postfach 1173 - 86406 Mering
Zeppelinstr. 10 - 86415 Mering
Tel. 08233/30524
Fax 08233/9971

PC-Meßtechnik

Kennung: PC-MT02
Datum: 02.05.95
Stichworte: PC, Meßtechnik, Timer, FORTH

Klaus Kohl

Programmierung des Timer0-Interrupt

Wenn nicht durch externe Hardware nachgeholfen wird, hat man im PC nur durch die Verwendung des Hardware-Timers 0 eine Quelle für einen zeitsynchronen Interrupt. Der folgende Artikel beschreibt hier noch ohne direkten Bezug zu einem Meßproblem die Verwendung dieses auch vom Betriebssystem genutzten Interrupts in eigenen FORTH-Programmen.

Alle Beispiele wurden im KK-FORTH für PC erstellt. Da dieses FORTH an volksFORTH angelehnt ist und die Programmbeispiele sehr kurz sind, dürfte eine Anpassung an andere FORTH-Versionen einfach sein. Zur Erleichterung der Anpassung hier einige Hinweise:

- Das Zeichen & kennzeichnet eine dezimale Schreibweise
- Um feste Adressen zu erhalten, wurden Datenfelder mit `CREATE` und `ALLOT` angelegt
- `CS@` liefert Codesegment des FORTH-Programmes

1. Programmierung des Timer 0 im PC

Wie den meisten PC-Anwendern bekannt ist, verfügt der PC über einen Timerbaustein. Der dazu meist verwendete 8253 hat drei unabhängige 16Bit-Timer, die mit einem Grundtakt von 1,193180MHz angesteuert werden.

Timer	Verwendung
0	System-Timer mit 18,2Hz (1,19MHz/65536) über Interrupt
1	früher zusammen mit DMA-Kanal für DMA-Refresh verwendet
2	meist für Frequenzerzeugung des Pipsers verwendet

Da der Timer 1 nicht in allen PC´s zur Verfügung steht und der Timer 2 für die Tonausgabe reserviert bleiben sollte, blieb für eine interruptgesteuerte Meßdatenerfassung nur der Timer 0. Da dieser Timer auch vom Betriebssystem z.B. Verwaltung der Softwareuhr verwendet wird, müssen eigene Routinen nach entsprechenden Zeitintervalle auch wieder die Betriebssystem-Interruptroutine aufrufen.

Portadresse	Bedeutung
$40	Daten für/von Timer 0
$41	Daten für/von Timer 1
$42	Daten für/von Timer 2
$43	Steuerwort

Das folgende Programmbeispiel verändert das Verhalten des Timers 0, ohne daß gleichzeitig die Interrupts umgeleitet werden. Es wird deshalb die Softwareuhr falsch laufen und evtl. vorhandene Bildschirmschoner mit noch nie gesehenen Eigenschaften aufwarten.

```
\ Timer0 auslesen
  : ti0@    ( -- val )
    $40 pc@  $40 pc@  flip  or ;

\ Timer0 verändern
  : ti0!    ( div -- ) \ Timer0 auf neuen Teilerwert stellen
    %00110110 $43 pc!           \ Timer0; LSB/MSB; Mode3; binär
    dup $40 pc!  flip $40 pc!                \ LSB/MSB ausgeben
    %00000000 $43 pc! ;
```

Das Auslesen des Timers ist unkritisch und liefert den aktuellen 16Bit-Wert, der von 0 bis zum maximalen Wert-1 geht. Bei der Standardeinstellung dauert es genau 65536/1,193180MHz = 54,9254ms bis der Zähler von 0 bis 65535 läuft.

Bei der Veränderung des Timers muß auch daß Steuerregister angesprochen werden. Der FORTH-Befehl **`TI0!`** will nur die Obergrenze neu festlegen und stellt deshalb den auch vom Betriebssystem verwendete Mode 3 ein. Andere Betriebsarten sind in den entsprechenden Unterlagen (siehe Literaturverzeichnis) zu finden. Da der Timer vom DOS normalerweise nicht umprogrammiert wird, bleibt die neue Einstellung auch nach dem Verlassen des Programmes erhalten. Eine einfache Anwendung währe z.B. das Herunterbremsen des Computers. Bei einem Wert unter **$100** wird sogar ein 486DX2 so langsam, daß man bei **`DIR`** mitlesen kann. Zurückgestellt wird der Timer dann wieder durch einen Hardware-Reset oder durch **`0 ti0!`** .

2. Interruptprogrammierung in FORTH

Nun zur praktischen Anwendung des Timers in der Meßdatenerfassung. Da man nicht die gesamte Zeit auf den Ablauf des Timers warten will, ist die Einbindung eigener Routinen in die Interruptsteuerung des Betriebssystems notwendig. Hier noch einige Hintergrundinformationen dazu:

Wenn der Timer 0 abgelaufen ist, wird der Interrupt $08 ausgelöst. Falls Interrupts erlaubt sind, beendet der Prozessor seinen gerade laufenden Befehl und merkt sich Adresse des nächsten Befehls auf dem Prozessorstack (entspricht meist dem Datenstack in FORTH). Gleichzeitig blockiert der Interruptcontroller alle weiteren Interrupts, bis er wieder durch Einschreiben des Wertes $20 in Portadresse $20 freigegeben wird. Aus den Speicheradressen $0020..$0024 des Segmentes $0000 wird dann die Adresse der zugehörigen Interruptroutine geholt und gestartet. Der normale DOS-Timerinterrupt wird dann neben einigen internen Aktionen auch noch den Interrupt $1C aufrufen, bevor er den Interruptcontroller wieder freigibt und das unterbrochene Hauptprogramm fortsetzt. Der Interrupt $1C (Adresse dazu ab $0000:$0070 im Segment $0000) wird z.B. vom BASIC zur Einbindung eigener Routinen wie die Steuerung der Tondauer genutzt.

Da wir vermeiden wollen, daß das normale Timing des Betriebssystems verändert wird, werden wir an Stelle des normalen Interrupts $08 eine eigene Routine einbinden. Diese hat folgende Aufgaben:

- Verwendete Register retten
- Eine eigene Routine ausführen

- Anzahl der Interrupts zählen und
  - - nur alle 52ms einen Interrupt zum Betriebssystem durchlassen
  - - sonst Interruptcontroller freigeben und zum Hauptprogramm zurückkehren

Um das Prinzip dieser Interruptsteuerung zu zeigen, werde ich hier aber keine aufwendige Abtastroutine einbinden, sondern einfach eine Variable hochzählen, die dann mit dem Hauptprogramm abgefragt und ausgegeben werden kann. Leider kann ich hier nicht mehr die Verwendung eines Assemblers vermeiden. Es handelt sich um einen leicht modifizierter 8086-Assembler mit folgenden Eigenheiten:

- Assemblerbefehle werden mit Komma beendet (z.B. CLI,)
- Befehlsaufbau: Quelle mit Addressierungsart Ziel mit Adressierungsart Befehl
- Adressierungsarten:
  - Register (AX, BX, ...) brauchen keine zusätzliche Angaben
  - Mit # nach dem Wert werden Konstanten angegeben
  - Mit #) werden direkte Adreßangaben gekennzeichnet
  - Segmentprefix werden unmittelbar zuvor mit Segment seg, angelegt
- Mit Kontrollstrukturen können bedingte Kurzsprünge angegeben werden
  - BEGIN, ... immer mit abschließenden Komma
  - Bedingungen (z.B. 0=) werden genau invertiert als Sprungbefehl compiliert

```
\ Timerwert n = 1193182Hz / Interruptfrequenz
 \ &1193 Constant #divt              \ Timerwert für  1000.15 Hz
    &298 Constant #divt              \ Timerwert für  4003.95 Hz
\ DOS-Interrupt alle  t = 65536 / n  Interrupts
 \   &55 Constant #div               \ Vorteiler für  1000.15 Hz
    &220 Constant #div               \ Vorteiler für  4003.95 Hz

\ Arbeitsvariablen
 2Variable oldint$08                \ Zeiger auf Standardroutine
 Create int8div  2 allot            \ Vorteiler mit fester Adresse

\ Interrupt ein-/ausschalten
  Code di       ( -- ) \ Interrupt abschalten
    cli,   next,   End-Code
  Code ei       ( -- ) \ Interrupt einschalten
    sti,   next,   End-Code

\ Eigenes Zählerprogramm
 Create counter  2 allot            \ Zählervariable
  PROC userint  ( -- ) \ Zählerinterrupt
     pushf,                         \ Flags retten
     cs seg,  counter #) inc,       \ Zähler erhöhen
     cs seg,  1 # int8div #) sub,   \ Teiler erniedrigen
     0= IF,                         \ Null erreicht ?
        cs seg,  #div # int8div #) mov,  \ alter Wert
        popf,                            \ Flag holen
        0 $20 l@   0 $22 l@  #  far jmp, \ DOS-Interrupt
     THEN,
     ax push,                       \ Register AF retten
      $20 # al mov,   $20 #) byte out,   \ Interruptcontroller frei
     ax pop,                        \ AF holen
     popf,                          \ Flags holen
     iret,                          \ Rückkehr aus Interrupt
   End-Proc

\ Interrupts auf eigene Routine und zurück zum DOS stellen
  : inton    ( -- ) \ Interrupt umleiten
```

```
    di   #divt ti0!                  \ Timer beschleunigen
         #div int8div !              \ Zähler zurückstellen
         0 $20 l2@  oldint$08  2!    \ Alte Vektoren merken
         cs@  (int++    0 $20  l2!   \ Neue Vektoren setzen
    ei ;
  : intoff   ( -- ) \ Interrupt zurückstellen
    di   $0000 ti0!                  \ Timer zurückstellen
         oldint$08 2@   0 $20  l2!   \ alte Interruptroutine
    ei ;

\ Hauptprogramm
  : main   ( -- ) \ Anzeigen der Zählerwertes im Interrupt
    0 counter !                      \ Zähler auf 0
    inton                            \ Interrupt ein
    BEGIN
       cr   counter @ u.   key?
    UNTIL
    intoff ;
```

Wenn das Programm mit **`MAIN`** gestartet wird, erfolgt eine laufende Ausgabe des aktuellen Wertes in **COUNTER** . Da dieser Zähler vom Hauptprogramm nicht verändert wird, ist dies eine Bestätigung des laufenden Interruptprogrammes. Man kann jetzt nach Rettung der verwendeten Register auch andere Routinen einbinden, die einen A/D-Wandler abfragt oder einen Digitalport verändert.

Hier noch eine Empfehlung vor dem Start dieser Versuche:

- Schreib-Caches für Laufwerke und Harddisks sollten abgeschaltet sein (z.B. Smartdrive)
- Windows möglichst nicht laufen lassen
- RESET-Schalter in Griffweite halten

3. Literatur zur Timerprogrammierung

c´t-Serie PC-Bausteine
Rund um den Timer
c´t 1988, Heft 4, Seite 196ff

Thom Hogan
Die PC-Referenz für Programmierer
Microsoft Press

Frank van Gilluwe
The undocumented PC
The Andrew Schulmann Programming Series
Addison-Wesley Publishing Company