Schnelle Umwege in Action!

Im zweiten Action!-Center behandeln wir die Interrupts für flotte Programme.

Die Benutzung von Interrupts ist ein Kapitel, das normalerweise nur Assembler-Programmierern vorbehalten ist. Gerade ihre Beherrschung stellt jedoch ein sehr gutes Hilfsmittel für eine Programmiersprache dar, da durch einen periodisch wiederkehrenden Interrupt (beim Atari VBI genannt) sogar eine Art von Parallelverarbeitung möglich ist. Das bedeutet nicht mehr und nicht weniger, als daß zwei Programme (quasi) gleichzeitig bearbeitet werden. Sie können sich vorstellen, daß es mit diesen Fähigkeiten kein Problem ist, fließende Animation mit Musik oder Geräuschen zu verbinden.

Nun wäre es aber schade, wenn eine so vielversprechende Fähigkeit nur in Assembler nutzbar sein sollte, doch ist es ganz einfach eine Frage der Rechenzeit. Der bereits erwähnte VBI wird von der Hardware des Rechners 50mal pro Sekunde aufgerufen. Dann verzweigt der Programmablauf zu einem speziellen Programmteil im ROM des Computers und kann, wie wir später noch sehen werden, auch auf eigene Programme umgelenkt werden. Es ist klar, daß die

8 Bit

vom VBI aufgerufene Routine auch innerhalb von 20 Millisekunden (1/50 sec) abgeschlossen sein muß, denn sonst würde sie nochmals aufgerufen, bevor sie beendet ist. Ein heilloses Durcheinander wäre die sichere Folge.

Eine langsame Sprache wie das Atari-Basic könnte in dieser Zeit nur sehr wenige Befehle abarbeiten. (Ein Befehl wie A=B*C nimmt alleine schon ca. 12 Millisekunden in Anspruch!) Ein Assembler-Programm kann dagegen in 20 Millisekunden eine ganze Reihe von Befehlen bearbeiten. Während dieser Zeitspanne stehen etwa 23.000 bis 33.000 Taktzyklen zur Verfügung, die für einige tausend Maschinenbefehle ausreichen. Damit läßt sich schon etwas anfangen.

Wer die letzte Folge des Action!-Centers verfolgt hat, der weiß, daß diese Sprache einem Assembler-Programm in Sachen Geschwindigkeit nicht viel nachsteht. Warum sollte man also Interrupt-Routinen nicht in Action! codieren?

Im Prinzip ist das kein Problem, nur eine Schwierigkeit gibt es dennoch zu überwinden. Jede Sprache benötigt einige Hilfsregister zur Ablage von Zwischenergebnissen. Es darf keinesfalls geschehen, daß die Interrupt-Routine die Hilfsregister des parallel laufenden Programms (man spricht hier auch vom Vordergrundprogramm) überschreibt. In Assembler betrifft dies nur die internen Register des Prozessors wie Akku und Index-Register, die man zu Beginn auf den Stapel rettet und vor dem Ende der Interrupt-Routine wieder zurückholt

Action! als Compilersprache benötigt da schon ein paar Register mehr. Grundsätzlich geht man aber genauso vor: Zu Beginn des Interrupts rettet man alle internen Register auf den Stapel und holt diese vor Ende des Interrupts auf dem umgekehrten Weg wieder zurück. Diese Auf-

> **Strenge Trennung der Datenbereiche**

gaben überträgt man zwei Code-Blocks, die im Beispielprogramm (Listing 1) am Anfang und Ende der Prozedur Vbipgm ( ) definiert sind.

Der erste Block überträgt einen Teil der Zero-Page auf den Stack, hauptsächlich die Bereiche von $ 80 bis $ 8F, die Action! zur Berechnung von Formeln benützt, sowie von $ A0 bis $ AF, einen Bereich, der vorwiegend zur Parameterübergabe Verwendung findet. Daneben werden noch vereinzelte Speicherzellen und natürlich die Prozessorregister auf den Stack geschoben. Der Code-Block am Ende von Vbipgm ( ) kehrt diesen Vorgang um und stellt somit den ursprünglichen Zustand wieder her. Das Vordergrundprogramm merkt somit von der Unterbrechung nichts, was schließlich auch der Zweck der Übung war.

Bei der Programmierung der Interrupt-Routine muß man streng auf eine Trennung der Datenbereiche von Interrupt zu Vordergrund achten. Action! erleichtert dies durch die Verwendung von lokalen Variablen; dagegen sind globale mit Vorsicht zu genießen, können jedoch zur Kommunikation zwischen den

Interrupts in Action!

```
;**************************************
;VERTICAL-BLANK INTERRUPT IN ACTION!
;
;P.FINIZEL                        1987
;**************************************

DEFINE  JMP     ="$4C",
        XITVBV  ="$E462"

;-------------------------------------
;Hardware- & Schattenregister
;-------------------------------------
CARD Vvblkd=$0224, Vvblki=$0222

BYTE
Dmactl=$D400, Prior =$26F,
Nmien =$D40E, Colbk =$D01A,
Gractl=$D01D, Pmbase=$D407,
Casinh=$02F0, SDmactl=$022F

BYTE ARRAY
Color(5)=$02C4, PColr(4)=$02C0,
Hposp(4)=$D000, Sizep(4)=$D008,
Trig(4) =$D010, Grafp(4)=$D00D

;-------------------------------------
;Graphik-Daten
;-------------------------------------
BYTE ARRAY Shape(0)=
[ $80 $F0 $60 $50 $08 $04 $02 $01 ]

CARD ARRAY PMAdr(4)=[0 0 0 0]

BYTE ZeigerX, ZeigerY

;-------------------------------------
;PM-Graphik einschalten
;-------------------------------------
PROC Pmgraphics(BYTE Basis)
BYTE i
CARD Adr
BYTE Adrhi=Adr+1

Pmbase=Basis
Adr   =0
Adrhi =Basis+4
Gractl=3
SDMCTL=$3A
FOR i=0 TO 3
  DO
    Zero(Adr,256)
    Pmadr(i)=Adr
    Adr     ==+256
    Sizep(i)=0
OD
RETURN

;-------------------------------------
;Einbinden von Interrupt-Routinen
;-------------------------------------
PROC Setvbi(BYTE POINTER Vektor)
Nmien=0 Vvblkd=Vektor Nmien=$40
RETURN

;-------------------------------------
;VBI-Routine
;-------------------------------------
PROC Vbipgm=*()

;zuerst Register retten
[ $A2   7 $85 $C0 $48 $85 $48
  $48 $85 $A0 $48 $85 $B0 $48
  $CA $10 $F1 ]

Zero(PMAdr(0)+ZeigerY,8)
IF (Stick(0)&1)=0 THEN
      IF ZeigerY>32 THEN
            ZeigerY==-1
      FI
ELSEIF (Stick(0)&2)=0 THEN
      IF ZeigerY<224 THEN
            ZeigerY==+1
      FI
FI
IF (Stick(0)&4)=0 THEN
      IF ZeigerX>48 THEN
            ZeigerX==-1
      FI
ELSEIF (Stick(0)&8)=0 THEN
      IF ZeigerX<208 THEN
            ZeigerX==+1
      FI
FI
MoveBlock(PMAdr(0)+ZeigerY,Shape,8)
Hposp(0)=ZeigerX

;Register holen...
[ $A2 0 $68 $95 $B0 $68 $95 $A0 $68
  $95 $A8 $68 $95 $C0 $EB $E0 8 $D0
  $EF ]
;...und VBI beenden
[JMP XITVBV]

;-------------------------------------
;Das Hauptprogramm
;-------------------------------------
PROC HAUPT()

  Pmgraphics($80) ;PMG ab Adr. $8000
  Prior=1         ;Prioritaet
  PColr(0)=$CE    ;Farbe Player
  ZeigerX=120     ;Startpunkt
  ZeigerY=120
  SetVBI(VBIPge)  ;Interrupt
PUT($7D)          ;Bildschim loeschen
Casinh=1          ;Cursor aus
POSITION(10,8)    ;Kasten zeichnen
Put(17) Put(18) Put(18) Put(5)
POSITION(10,9)
Put(124) Put(79) Put(75) Put(124)
POSITION(10,10)
Put(26) Put(18) Put(18) Put(3)
DO
IF TRIG(0)=0 THEN
  IF (ZEIGERX)88) AND
     (ZEIGERX(104) THEN
     IF (ZEIGERY)94) AND
        (ZEIGERY(120) THEN
          EXIT
     FI
  FI
FI
OD
SETVBI(XITVBV)
RETURN
```

Programmteilen eingesetzt werden.

Das Einbinden einer Action!-Prozedur in den VBI ist mit der Routine SETVBI ( ) sehr leicht zu erledigen. Als einzigen Parameter muß man der Routine die Adresse des VBI-Programms als CARD übergeben. Dort wird zunächst der VBI mittels des Registers NMIEN abgeschaltet, dann der Vektor VVBLKD auf die gewünschte Adresse geändert und schließlich der Interrupt per NMIEN wieder zugelassen. Der Aufruf SETVBI (VBIPGM) hat somit zur Folge, daß die Prozedur VBIPGM 50mal pro Sekunde aufgerufen wird.

Interrupts sind Unterbrechungen des Programmablaufs für bestimmte Aufgaben. Macht man sich diese zunutze, so ist eine Art Parallelverarbeitung programmierbar.

Das Interrupt-Programm VBIPGM selbst beginnt, wie wir schon gesehen haben, mit dem Code-Block zum Retten der Register. Daran schließen sich die Befehle an, die im VBI ausgeführt werden sollen. Hier kann ganz normal in Action! programmiert werden; auch andere Prozeduren lassen sich selbstverständlich aufrufen. Vorsicht ist lediglich bei Befehlen geboten, die I/O-Funktionen aufrufen (z.B. OPEN, PRINT usw.) oder das Betriebssystem benötigen (PLOT, DRAW usw.). Den Abschluß bilden der Code-Block zur Wiederherstellung der Registerinhalte und ein Sprung zum Label XITVBV ($E462), der wiederum als Code-Block erfolgt.

Das Beispiel im Listing benutzt nun diese Möglichkeiten, um eine Art Mauszeiger mit dem Joystick bewegen zu können. Der Zeiger ist nichts weiter als ein Player, der sich durch den Einsatz des Interrupts vollkommen fließend und unabhängig vom Vordergrundprogramm bewegen läßt. Um das Ganze nicht allzu sehr aufzublähen, wird im Vordergrund nur eine sehr einfache Funktion erledigt, nämlich gewartet, bis der Zeiger auf das Kästchen OK deutet und der Knopf am Joystick gedrückt wird. Sie sehen, mit dieser Technik läßt sich eine mausähnliche Bedienung sehr leicht erzeugen.

Beachten Sie bitte, daß Bewegung des Zeigers und Abfrage der Position unabhängig voneinander erledigt werden. Als Verbindung zwischen den Programmteilen dienen nur die globalen Variablen Zeiger _X und Zeiger _Y.

Mit diesem Grundkonzept kann man aber auch sehr viel kompliziertere Programmsysteme aufbauen. Denkbar wäre z.B. ein Spiel, das alle Bewegungen der Players oder ein eventuelles Scrolling im VBI durchführt, während die Logik des Spiels (Ablauf, Punktezählung usw.) im Vordergrund erledigt wird. Warum nicht umgekehrt? Das liegt an der Natur des VBI, der bekanntlich in der vertikalen Austastlücke des vom Computer erzeugten Videobildes ausgelöst wird. Alle Grafikänderungen, die im VBI stattfinden, sind daher nicht mit Störungen (z.B. Flimmern) behaftet.

Interessant ist noch die Prozedur PMGraphics (), mit der die PM-Grafik eingeschaltet wird. Die Anfangsadressen des Videospeichers der einzelnen Players sind danach im CARD-Array PMAdr zu finden. Die Bewegung des Players geschieht in der Routine..., indem zuerst der Zeiger an der alten Position gelöscht, danach der Joystick abgefragt und schließlich die Form des Zeigers aus dem Array SHAPE an die neue Position kopiert wird. Dank der enormen Ge-

„Enorme Geschwindigkeit"

schwindigkeit eines Action!-Programms nützt die Routine den Zeitrahmen des VBI nur zu einem geringen Teil aus; tatsächlich sind bei geschickter Programmierung sehr viel aufwendigere Routinen möglich.

Man sollte natürlich auch bedenken, daß die Rechenzeit des VBI von der gesamt verfügbaren abgeht, d.h., bei einer sehr langen Routine kommt das Vordergrundprogramm fast zum Stillstand.

Auf ähnliche Weise lassen sich auch die anderen System-Interrupts (DLI oder I/O-Interrupts) auf Action!-Programme umlenken, doch mehr davon in einer der nächsten Folgen. Sie sehen, daß man mit dieser C-ähnlichen Sprache auch so komplizierte Sachverhalte wie Interrupts mühelos in den Griff bekommt, ohne auf die langwierige Assembler-Programmierung ausweichen zu müssen. Es ist daher kein Wunder, wenn die Betriebssysteme der neuen 16-Bit-Generation (z.B. ST oder Amiga) ausschließlich in C programmiert sind.

Peter Finzel

Meister der Disketten

Benötigen Sie einen professionellen Programmierschutz für Ihre Software? Kein Problem mit dem neuen Programm "Diskmaster", das wir Ihnen hier vorstellen.

Ohne Zweifel übt das Thema Kopierschutz auf viele Leute eine enorme Faszination aus. Mit dem Programm "Diskmaster" ist es für jedermann möglich, einen Kopierschutz zu erzeugen, der selbst professionellen Ansprüchen genügt. Voraussetzung ist ein 1050-Laufwerk mit einer Happy-Erweiterung bzw. ein dazu kompatibles Produkt.

Erstaunlicherweise kann "Diskmaster" mit diesem Hardware-Zusatz sogar Kopierschutzformate erstellen, die sich nicht einmal mit der Happy-Erweiterung kopieren lassen, geschweige denn mit einem normalen Sektorkopierer.

8 Bit

Der zentrale Teil des Programms ist der sogenannte Format-Editor, mit dem ein Kopierschutzformat in einer Art eigener Programmiersprache entworfen werden kann. Dabei läßt sich jedes einzelne Byte des Formats bestimmen; mehr oder weniger kann der FDC (der Floppy-Controller der 1050-Station) direkt angesprochen werden. Wer sich schon einmal mit der Floppy-Anschaltung beim Atari beschäftigt hat, weiß, daß dies keine einfache Sache ist. Der FDC gehorcht nämlich nicht den Befehlen des Mikroprozessors im Computer, sondern verfügt über eine eigene CPU, die lediglich über einen seriellen Bus mit der zentralen CPU in Verbindung steht. Zudem ist das normale Betriebsprogramm der Floppy-CPU penibel dazu ausgelegt, solche nicht vorgesehenen Aktionen strengstens zu verhindern.

Durch den Format-Editor lassen sich Kopierschutzformate mit doppelten Sektoren oder absichtlichen Lesefehlern erzeugen oder einfach mehr Sektoren als gewöhnlich auf einer Spur unterbringen. Nach Wunsch kann dabei der Single- oder Double-Density-Modus verwendet werden. Dennoch ist es nicht ganz leicht, einen Kopierschutz zu erstellen, da man schließlich jedes Byte auf der Diskette berücksichtigen muß. Glücklicherweise enthält die Rückseite der Diskette noch eine Reihe fertiger Formate als Files, die man sofort ausprobieren und nachvollziehen kann. Außerdem sind auch einige Basic- und Assembler-Programme zu finden, welche die Abfrage der verschiedenen Kopierschutzformate als Beispiel zeigen.

Neben dem Editor besitzt "Diskmaster" noch einige weitere Funktionen, die zum Speichern und Laden von Formaten und sonstigen Daten dienen. Natürlich darf auch eine Option zum Aufbringen des Formats

„Happy-kompatible Erweiterung erforderlich"

nicht fehlen. Erwähnenswert ist noch die Funktion SPUR EINLESEN; sie überträgt eine ganze Spur mit allen darauf befindlichen Daten, also auch Sektor-Header und Prüfsummen-Bytes, in den Speicher des Computers.

Die sehr ausführliche Dokumentation umfaßt 15 DIN-A4-Seiten. Sie ist – was ich für besonders wichtig halte – nicht nur auf die Bedienung des Programms zugeschnitten, sondern erklärt auch einige wichtige Grundlagen der internen Funktion der Diskettenstation. Das ist gerade bei der Erzeugung eines Kopierschutzes von großer Bedeutung, da zu diesem Zweck alle Eigenheiten der Floppy genutzt werden. Allerdings ist es natürlich unmöglich, eine so komplexe Thematik dem Laien auf wenigen Seiten erschöpfend zu erklären; einige Vorkenntnisse sind hier schon erforderlich. Wer mit Begriffen wie Gapbyte und CRC-Generator nichts anzufangen weiß, wird bei der Lektüre schnell überfordert sein.

Trotzdem ist "Diskmaster" mit seinem günstigen Preis von 24.90 DM allen zu empfehlen, die mehr über ihr Diskettenlaufwerk und seine Funktionsweise wissen möchten. Beim Kauf ist unbedingt zu beachten, daß man zur Anwendung des Programms zumindest eine Happy-kompatible Erweiterung braucht. Funktionsähnliche Produkte wie Speedy 1050 oder Turbo 1050 sind hier nicht verwendbar. Im Zweifel fragen Sie am besten Ihren Händler.

Zu haben ist das Programm beim Software-Versand des **ATARI***magazins* (Bestellschein S. 29).

Peter Finzel