

PETER'S ASSEMBLERECKE

Musik in Action!

In der Assemblerecke dieses Monats gibt es eine besondere Premiere: Das erste Action!-Programm, das wir in Computer-Kontakt abdrucken, wenn nicht sogar das erste in der gesamten deutschen Computer-Literatur. Action! ist schlicht gesagt eines der besten Software-Pakete, die für Atari-Computer je entwickelt wurden. Russ Wetmore bezeichnet es in einem Artikel (Analog 7/85, S. 23) als "Programmer's Dream" und ich kann Russ wirklich nur voll unterstützen.

Wer bisher noch keine eigenen Erfahrungen mit Action! sammeln konnte, sollte deshalb den Kasten "Was ist Action?" lesen. Dort finden Sie einige zum Verständnis wesentliche Punkte kurz zusammengefaßt. Es muß vorausgeschickt werden, daß man zum Eintippen des Listings unbedingt eine Action!-Cartridge benötigt, bestimmt eine bittere Pille für alle Leser der Assemblerecke, die keine besitzen. Vielleicht ist es aber auch für diesen Leserkreis interessant, einen Einblick in ein Action!-Programm zu bekommen.

Vierstimmige Musik

Das Programm kann ein Musikstück mit bis zu vier Noten gleichzeitig spielen, wobei jeder einzelnen Stimme eine eigene Hüllkurve zugeordnet werden kann. (Die Hüllkurve kann jeder Stimme näherungsweise den Klang eines Instrumentes geben.) Das Programm wurde nicht im Stil eines Musik-Composers mit einer narrensi-

cheren Eingabe der Noten angelegt, sondern ist vielmehr als ein offenes, einbaufertiges Musik-Modul für eigene Programme gedacht. Die Noten müssen daher direkt ins Programm eingetragen werden, was aber dank des hervorragenden Action!-Editors recht elegant machbar ist.

Historie des Programmes

Dem eifrigen Leser meines Buches, der Hexenküche, wird bestimmt eine Ähnlichkeit zu dem dort abgedruckten Musik-Programm auffallen. Richtig, das Action!-Listing ist ein Verwandter dieses Assemblerprogrammes, hat aber zwei wesentliche Unterschiede: Die Hüllkurve kann nun im ADS-System (Attack, Decay, Sustain) eingegeben werden, und außerdem läuft das Programm nicht im VBI. Durch die letztere Maßnahme kann man eine wesentlich bessere Auflösung der Hüllkurve erreichen, verliert aber leider die Möglichkeit, ein anderes Programm parallel laufen zu lassen.

Eingabe der Noten

Die Noten werden im Action!-Programm in den vier Byte Arrays N1 bis N4 untergebracht. Die Frequenzwerte und Notenlängen sind durch DEFINES (entspricht etwa dem "EQU" oder "=" Befehl eines Assemblers) bereits vorbesetzt, so daß eine recht komfortable Eingabe der Noten möglich ist.

Der Ausdruck C3:VI würde ein tiefes "C" der Länge einer Viertelnote (VI für Viertel) spielen. Das Spektrum der Tonhöhen reicht über drei Oktaven, beginnend bei einer tiefen Oktave, (C3, die Zahl nach der Notenbezeichnung gibt die Oktave an) über eine mittlere (C4) zu einer hohen Oktave (C5). Die Tonlängen sind dabei wie folgt festgelegt.

HA : halbe Note
VI : viertel Note
AC : achtel Note
SE : sechzehntel Note
HP : halbe Note punktiert
VP : viertel Note punktiert
AP : achtel Note punktiert

Daneben gibt es noch einen Pausenbefehl PA, der ebenfalls mit einer Dauer versehen werden muß: PA:HA würde die jeweilige Stimme eine halbe Note lang verstummen lassen. Der Ende-Befehl EN schließt eine Notenfolge ab. Das Musikprogramm endet aber erst, nachdem eine Ende-Anweisung auf allen vier Kanälen erfolgt ist.

Beispiel

Ein etwas längeres, wenn auch nur zweistimmiges Beispiel ist bereits im Programm eingetragen. Wenn Sie eine andere Melodie eingeben möchten, so müssen Sie die Zeilen jeweils nach den Byte Arrays N1 und N2 bis zur Ende-Anweisung löschen. Damit haben Sie sich ein leeres Formular zur Noteneingabe geschaffen. Angenommen, Sie wollen jetzt drei Noten der mittleren C-Dur Tonleiter einstimmig als Viertelnoten spielen, so müssen Sie eingeben:

```
BYTE ARRAY N1 = [C4:VI  
D4:VI E4:VI EN:0]
```

Alle anderen, ruhigen Stimmen (N2 bis N4) bekommen nur die Ende-Anweisung eingetragen.

```
BYTE ARRAY N2 = [EN:0]
```

Die Null nach dem Ende-Befehl hat dabei keine praktische Bedeutung und dient nur der einfacheren Verarbeitung. Zum Spielen der Noten muß das Programm kompiliert und mit "R" gestartet werden.

Klang-Experimente

Mit sechs weiteren Byte Arrays können vier Hüllkurven vorgegeben werden, wobei jeweils das Element 0 zur Stimme 1, das Element 1 zur Stimme 2 usw. gehört. Im Byte Array Att stellen Sie die Härte des Anschlags ein. Je höher die Zahl, desto steiler der Anschlag. Dec(4) gibt die Steilheit des Abfalles (Decay) auf die Haltestärke (Sustain) an. Wiederum gilt hier: je höher die Zahl, desto steiler der Abfall. Sus(4) ist

ein Maß für die Haltelautstärke, die, wie alle Werte, im Bereich von 0 bis 255 eingetragen werden kann.

Wie bitte? 256 Lautstärke-stufen? Der Atari hat doch nur 16!

Richtig, aber durch die Unterteilung der 16 Stufen in jeweils 16 Bruchteile kann man bei Attack und Decay eine bessere zeitliche Auflösung erreichen. Konkret: Eine 1 im Attack bewirkt, daß sich die Lautstärke nur bei jedem 16-ten Schleifendurchlauf tatsächlich erhöht. Wenn Sie daher in Sus(0) den Wert 160 eintragen, so entspricht das der Lautstärke 10 eines Sound-Befehles.

Noch drei weitere Klang-Parameter können eingetragen werden: Max() gibt den höchsten Lautstärkepegel nach dem Attack an. Mit Dis() kann die Tonverzerrung wie im Sound-Befehl angegeben werden. Normal wird hier immer die 10 (für puren Ton) stehen, aber mit anderen Werten kann recht einfach ein Schlagzeug nachgeahmt werden. Off() gibt Ihnen die Möglichkeit, die Noten bewußt zu verstimmen. Ein interessanter Schwebungs-Effekt ergibt sich, wenn zwei Stimmen die gleichen Melodien spielen, und eine davon mit Off() = 1 um eine Frequenzeinheit verstimmt ist. Mit der Variablen TEMPO kann die Geschwindigkeit des Abspielens eingestellt werden, je größer die Zahl, desto langsamer wird gespielt.

Einige Hinweise:

Hauptmodul des Programmes ist die Prozedur Musik, die für das Spielen des ganzen Stückes verantwortlich ist. Es benutzt die Prozedur Alle-4(), in der nacheinander alle 4 Stimmen bearbeitet werden. Die Prozedur Hardware() ist der sogenannte Hardware-Treiber, der die in den Modulen Stimme() und Hülle() berechneten Werte der Lautstärke (in Laut(i)) und der Tonfrequenz (in Freq(I)) in eine für die Hardware angenehme Form bringt und diese schließlich in die passenden POKEY-Register einträgt.

Hier noch Hinweise zu einigen Formulierungen in Action!, die die Basic-Programmierern höchstwahrscheinlich nicht geläufig sind: Der Ausdruck $i==+1$ ist eine Kurzschreibweise

für $i=i+1$. LSH und RSH sind Operatoren. Für Newcomer: Links-Shift entspricht einer Multiplikation mit 2. Der Ausdruck $A = B \text{ LSH } 4$ bedeutet, daß B viermal links geschiftet wird. In Basic müßte dazu eine Multiplikation verwendet werden: $A = B * 16$. Weiterhin ist es in Action! möglich, einfache Variablen und sogar ganze Felder direkt auf Hardware-Register zu legen, wodurch Peek und Poke-Befehle überflüssig werden. Für $AUDCTL = 0$ würde man z. B. in Basic POKE $AUDCTL, 0$ schreiben müssen.

Ich hoffe, daß das Programm mit diesen Hinweisen auch für Leute ohne Erfahrung in Action! etwas transparenter geworden ist, aber es ist natürlich unmöglich, das (über 200 Seiten starke) Action!-Handbuch hier in ein paar Zeilen zusammenzu-

fassen. Vielleicht haben Sie trotz der Menge an neuen Informationen erkannt, daß Action!-Programme wesentlich einfacher als Assemblerprogramme zu schreiben und auch zu überblicken sind und obendrein ei-

nem Assemblerprogramm in Sachen Geschwindigkeit nicht viel nachstehen. Meiner Meinung nach wird die Zukunft des Programmierens in Sprachen wie Action! zu suchen sein.

Peter Finzel

Was ist Action?

Action! ist eine strukturierte Compiler-Sprache für 8-Bit Atari-Computer, die Pascal und besonders "C" recht nahe steht. Mehr noch, Action! ist eine gesamte Programmierungsumgebung, die einen wirklich traumhaften Editor, einen irrwitzig schnellen Compiler und einen Monitor vereinigt. Das alles findet auf einem 16 K Steckmodul Platz, das dank einer auf dem Modul befindlichen Banking-Logik nur 8 KByte des wertvollen RAMs belegt.

Die Sprache an sich ist nicht schwer zu erlernen, wenn gleich gegenüber Basic schon einige Unterschiede vorhanden sind. Am auffälligsten ist wohl das Fehlen eines GOTO-Befehles, der aber in der klaren Struktur eines Action!-Programmes keinen Platz hätte. Basic wurde auf Mikrocomputern recht populär, da es interaktives Arbeiten zuläßt, d. h. ein gerade eingetipptes Programm kann sofort mit RUN gestartet und anschließend wieder editiert werden, ohne daß Diskettenzugriffe nötig sind.

Gewöhnlich sieht die Sache mit Compilersprachen ganz anders aus: Hier muß zunächst ein Editor geladen und das damit eingetippte Programm (der Quelltext) auf Disk gespeichert werden. Nun wird der Compiler geladen, der aus dem Quelltext ein lauffähiges Programm erzeugt und dieses ebenfalls auf der Diskette hinterläßt. Sollte ein Fehler aufgetreten sein, so heißt das: Editor neu laden, Quelltext laden, editieren, abspeichern, Compiler laden...

Wer eine neue Sprache mit so einem Compiler lernen will, der muß schon eine ganze Portion Geduld und Hartnäckigkeit mitbringen. Nicht so bei Action! Hier sind Editor, Quelltext und Compiler gleichzeitig im Speicher untergebracht und daher ohne Wartezeit per Tastendruck erreichbar. Der unglaublich schnelle Action!-Compiler erledigt seine Aufgabe in wenigen Sekunden (bei kleinen Programmen sogar in Bruchteilen von Sekunden), so daß das compilierte Programm wie in Basic sofort mit RUN

gestartet werden kann. Action! hat also einen Compiler mit Interpreter-Komfort.

Obendrein produziert Action! reine 6502-Maschinensprache, so daß 100 bis 200-fache Geschwindigkeitssteigerungen gegenüber Basic keine Seltenheit sind. Es ist sogar möglich, so zeitkritische Programmteile wie Display-List Interrupts in Action! zu schreiben. Das schafft wirklich kein anderer Compiler.

Action!-Programme sind in einzelne Prozeduren (PROCs) und Funktionen (FUNCS) unterteilt, wobei ein Programmteil nur zuvor schon definierte PROCs und FUNCS aufrufen kann. Der zuletzt definierten Prozedur kommt daher die Sonderstellung des Hauptprogrammes zu, also desjenigen Modules, das mit RUN aufgerufen wird. Aus diesem Grund ist es auch sinnvoll, ein Action!-Programm von unten nach oben zu lesen.

An Variablentypen gibt es 1-Byte (BYTE) und 2-Byte (CARD) Variablen sowie Felder (ARRAYS) und Zeiger (POINTER) dieser Typen.

Letztere sind Variablen, die selbst die Adresse einer anderen Variablen, (oder sonstigen Speicheradresse) beinhalten. Wichtig ist auch die Unterteilung in lokale und globale Variablen. Erstere werden innerhalb einer Prozedur definiert und sind auch nur dort verfügbar. Globale Variablen behalten dagegen ihre Gültigkeit im ganzen Programm. (Vergleiche: In Basic sind alle Variablen global!)

Verzweigungen im Programm werden nicht über Zeilennummern, sondern in strukturierten IF-THEN-ELSE-FI Blöcken abgewickelt. Die DO - OD Struktur führt alle Befehle zwischen DO und OD in Form einer Endlosschleife aus und kann durch FOR-TO, WHILE und UNTIL-Befehle beendet werden.

Diese kurzen Erläuterungen helfen Ihnen vielleicht, Action!-Listings etwas besser zu verstehen. Wer auch in Zukunft mehr über Action! lesen möchte, der schreibe mir bitte ein paar Zeilen. (Anschrift siehe Verlag).

Peter Finzel

Musik in Action!

```

;*****
;      MUSIK IN ACTION!
;
; PETER FINZEL                      1985
;*****

```

```

DEFINE B2  ="255",
C3  ="242", CIS3="229",
D3  ="216", DIS3="204",
E3  ="193",
F3  ="182", FIS3="172",
G3  ="162", GIS3="153",
A3  ="145", AIS3="137",
B3  ="129",
C4  ="122", CIS4="115",
D4  ="108", DIS4="102",
E4  ="97",
F4  ="91", FIS4="86",
G4  ="81", GIS4="77",
A4  ="72", AIS4="68",
B4  ="65",
C5  ="61", CIS5="58",
D5  ="54", DIS5="51",
E5  ="48",
F5  ="46", FIS5="43",
G5  ="41", GIS5="38",
A5  ="36", AIS5="34",
B5  ="32"

```

```

DEFINE HA="120", VI="60",
AC="30", SE="15",
HP="180", VP="90",
AP="45",
PA="0", EN="1"

```

```

DEFINE Pause="0", Attack="1",
Decay="2", Sustain="3"

```

```

;*****
;Ab hier stehen die Noten
;*****

```

```

; MENUETT NR.2

```

```

BYTE ARRAY N1=[
C4:VI G4:VI G4:AC F4:AC
E4:VI D4:AC E4:AC C4:VI
A4:AC G4:AC F4:AC E4:AC D4:AC C4:AC
B3:VI A3:AC B3:AC G3:VI
F4:VI F4:VI F4:VI
E4:VI E4:VI E4:VI
E4:VI G4:AC F4:AC E4:AC D4:AC
C4:HP
;
E4:AC C4:AC G4:AC C4:AC E4:AC C4:AC
D4:AC B3:AC G4:AC B3:AC D4:AC B3:AC
C4:AC D4:AC C4:AC B3:AC A3:AC G3:AC
FIS3:VI E3:AC FIS3:AC D3:VI
C4:VI C4:VI C4:VI
B3:VI B3:VI B3:VI
B3:VI D4:AC C4:AC B3:AC A3:AC
G3:HP
EN:0 ]

```

```

BYTE ARRAY N2=[
C3:VI C3:VI C3:VI
C3:HP
F3:VI D3:VI F3:VI
G3:VI D3:VI G3:VI
D4:VI D4:VI D4:VI
C4:VI C4:VI C4:VI
C3:VI E3:VI G3:VI
C4:VI G3:VI C3:VI
;
C4:VI G3:VI C4:VI

```

```

B3:VI G3:VI B3:VI
A3:VI E3:VI C3:VI
D3:VI PA:VI PA:VI
A3:VI D3:VI D3:VI
G3:VI D3:VI D3:VI
G3:HA D3:VI
G3:AC A3:AC G3:AC F3:AC E3:AC D3:AC
EN:0 ]

```

```

BYTE ARRAY N3=[
EN:0 ]

```

```

BYTE ARRAY N4=[
EN:0 ]

```

```

BYTE ARRAY
;KANAL -1- -2- -3- -4-
Att(4)=[ 070 030 000 000 ],
Dec(4)=[ 008 004 000 000 ],
Sus(4)=[ 048 032 000 000 ],
Max(4)=[ 200 140 000 000 ],
Dis(4)=[ 010 010 010 008 ],
Off(4)=[ 000 000 000 000 ]

```

```

BYTE TEMPO=[ 125 ]

```

```

;*****
;Interne Variable
;*****
BYTE SKCTL=$D20F,
AUDCTL=$D20B
CARD ARRAY NOTENZ(4)
BYTE ARRAY DAUER(4), LAUT(4),
FREQ(4), HKFlg(4)
BYTE ARRAY AUDF(8)=$D200,
AUDC(8)=$D201
BYTE ENDE
BYTE POINTER N_Ptr

```

```

;*****
;Vorbereitungsprogramm
;
;Initialisierung der Notenzeiger
;*****

```

```

PROC Vorbereitung()
BYTE i
NOTENZ(0)=N1 NOTENZ(1)=N2
NOTENZ(2)=N3 NOTENZ(3)=N4

```

```

FOR i=0 TO 3
DO
Dauer(i)=0
Freq(i)=0
LAUT(i)=0
DD

```

```

SKCTL=3
AUDCTL=0
RETURN

```

```

;*****
;Eintragen in Hardware-Register
;
;Die Werte Laut(i), Freq(i) und
;Dis(i) werden verknuepft und in
;die Audio-Register eingetragen
;*****

```

```

PROC Hardware(BYTE i)
BYTE j,d
j=i LSH 1
d=Dis(i) LSH 4
AUDC(j)=(Laut(i) RSH 4) % d
AUDF(j)=Freq(i)
RETURN

```

```

;*****
;Befehle zur Tonaenderung
;
;Pausen und der Befehl zum Beenden
;einer Stimme werden hier bearbeitet
;*****

```

```

PROC Befehl(BYTE i,f,d)

```

```

IF f=PA THEN ;Pause
Dauer(i)=d
Laut(i)=0
Freq(i)=0
Notenz(i)=+2
HKFlg(i)=Pause
ELSEIF f=EN THEN ;Ende
Laut(i)=0
Freq(i)=0
HKFlg(i)=Pause
ENDE=-1

```

```

FI
RETURN

```

```

;*****
;Spielen einer Stimme
;

```

```

;Besorgt Frequenz und Timing einer
;Note eines Kanales

```

```

;*****

```

```

PROC STIMME(BYTE i)
BYTE f,d

```

```

IF Dauer(i)>1 THEN
Dauer(i)--1 ;Note dauert an
ELSE
f =N_ptr^ ;neue Frequenz
N_ptr==+1 ;und Dauer
d =N_ptr^

```

```

IF f<=1 THEN
Befehl(i,f,d)

```

```

ELSE
Freq(i) =f
Dauer(i) =d
Laut(i) =0
HKFlg(i) =Attack
Notenz(i)=+2
Freq(i) ==+Off(i)

```

```

FI

```

```

FI
RETURN

```

```

;*****
;Huellkurve bearbeiten
;
;Berechnet die Huellkurve nach dem
;ADS-Verfahren(Attack/Decay/Sustain)
;*****

```

```

PROC Huell(BYTE i)
INT l

```

```

l=Laut(i)
IF HKFlg(i)=Attack THEN
l==+Att(i)
IF l>Max(i) THEN
l=Max(i)
HKFlg(i)=Decay

```

```

FI
ELSEIF HKFlg(i)=Decay THEN
l==Dec(i)
IF l<Sus(i) THEN
l=Sus(i)
HKFlg(i)=Sustain

```

```

FI
ELSEIF HKFlg(i)=Sustain THEN

```

```

;dann passiert nichts!

```

```

ELSEIF HKFlg(i)=Pause THEN
    l=0
FI
Laut(i)=1
RETURN

;*****
;Alle vier Stimmen spielen
;
;Der Notenzeiger wird auf das zu be-
;arbeitende Array gerichtet.
;Stimme() gibt die Frequenz des
;Tongenerators i vor, waehrend
;Huelle(i) die momentane Lautstaerke
;berechnet. Hardware (i) schreibt
;die berechneten Werte in die
;Audio-Register
;*****

PROC Alle_4()
BYTE i

FOR i=0 TO 3
    DO
        N_Ptr=NotenZ(i)
        Stimme(i)
        Huelle(i)
        Hardware(i)
    OD
RETURN

;*****
;Ganzes Musikstueck spielen
;
;Dies ist das Hauptmodul und wird
;nach RUN angesprungen. Das Musik-
;stueck wird vollstaendig durch-
;gespielt und wiederholt. Abbruch
;mit <RESET>
;*****

PROC Musik()
CARD i

DO
    Vorbereitung()
    DO
        ENDE=4
        ALLE_4()
        FOR i=0 TO TEMPO DO OD
        UNTIL (ENDE=0)
    OD
OD
RETURN

```