

PETER'S ASSEMBLERECKE

Bit-Map Grafik

Das Thema dieser Assemblerecke geht auf einen Leserbrief von A. Schmitz aus Bonn zurück, der sich eine genauere Betrachtung der hochauflösenden Grafik gewünscht hatte. Hierzu lesen Sie diesmal, wie GRAPHICS, PLOT und DRAWTO-Befehle in Maschinensprache angesteuert werden.

Wie gewohnt, soll der Einsatz dieser Befehle gleich an einem konkreten Beispiel demonstriert werden. Dazu wollen wir ein BASIC-Programm direkt in ein Assemblerprogramm umwandeln, wenn Sie so wollen, erledigen wir also die Arbeit eines Compilers. Kürzlich bin ich in einer englischen Zeitschrift (Atari-User, Aug. 85, "Mandala") auf ein nettes Grafik-Demo gestoßen, das sich hervorragend für diesen Zweck eignet. Dieses Basic-Programm, das Sie aus Listing 1 entnehmen können, zeichnet einen von vier Hyperbeln begrenzten Körper, der interessanterweise nur durch gerade Linien erzeugt wird. Tippen Sie's doch gleich mal ein, es ist recht kurz und lohnt sich bestimmt. Damit auch alle Grafik-Befehle vorkommen, habe ich noch einen Fill-Befehl (mit dem mysteriösen XIO-Befehl) für den Hintergrund eingebaut.

Compiler per Hand

Dieses Programm soll nun in voller Schönheit in Assembler umgesetzt werden. Ein erster Schritt in diese Richtung ist die Programmierung eines Satzes von Unterprogrammen, die die Funktionen der Basic-Befehle GRAPHICS, POSITION, PLOT und DRAWTO simulieren. Und das ist gar nicht so schwer, denn das Betriebssystem des Atari-Computers erledigt fast alles für uns.

Die den Lesern der Assemblerecke bestens bekannte CIO (Central Input/Output) Funktion greift uns auch hier unter

die Arme. CIO-Funktionen, deren Anwendungsbereich vom Diskettenbetrieb bis zur Abfrage der Tastatur reicht, umfassen das Öffnen und Schließen eines Datenkanals, Ein- und Ausgabe von Daten sowie einige diverse Spezialfunktionen. Speziell auf den Display-Handler (also den Teil des Betriebssystems, der die Grafik-Modi umfaßt) angewendet, bedeutet das folgendes: Der GRAPHICS-Befehl wird durch Öffnen eines Kanals auf den Display-Handler bewirkt, gewöhnlich wird dazu der IOCB 6 verwendet. Die Grafikstufe wird dabei in die Auxiliary-Bytes des IOCBs eingetragen; hier sind noch einige Bit-Spielereien nötig, um Split-Screens und ähnliches zu berücksichtigen. Wie es im Detail gemacht wird, können Sie dem Assembler-Listing 2 entnehmen.

Grafik-Cursor

Grundlage der Grafik-Programmierung ist der POSITION-Befehl, mit dem der Cursor, der in den Bit-Map Grafikstufen unsichtbar ist, an eine gewünschte Stelle gebracht werden kann. Der Einfachheit halber werden im Demo-Programm in Listing 2 nur X-Positionen von 0 bis 255 zugelassen, so daß die Übergabe der Koordinaten im X-Register (Abstand vom linken Bildrand) und im Y-Register (Abstand vom oberen Bildrand) erfolgen kann. Somit kann ein GRAPHICS 8 Bildschirm, der über 320 Bildpunkte in horizontaler Richtung verfügt, nicht vollständig angesteuert werden. Für alle anderen Modi ist dieses Konzept jedoch voll ausreichend, und falls Sie trotzdem den Drang haben, unbedingt GR. 8 zu verwenden, so können Sie noch den Akku zur Übergabe des höherwertigen Bytes der X-Position verwenden, das dann in COLCRS+1 (\$56) eingetragen werden muß.

PLOT

Mit Hilfe des eben beschriebenen POSITION-Unterprogrammes können wir uns nun schon an die Ausgabe eines Punktes am Schirm wagen. Wiederum soll diese Aufgabe einem Unterprogramm übertragen werden, das Sie leicht in eigene Programme übernehmen können. Nachdem Sie bereits wissen, daß der GRAPHICS-Befehl auf ein OPEN-Kommando des Betriebssystems zurückgeführt wird, können Sie auch folgern, daß PLOT-Befehle einfach durch Ausgabe eines Wertes (der Farbe) auf den durch GRAPHICS geöffneten IOCB erfolgen. Genau richtig, nur zuvor müssen die gewünschten Koordinaten per X- und Y-Register durch die POSITION-Routine festgelegt werden. Die Farbe wird aus der Speicherzelle ATACHR entnommen, in der z. B. auch BASIC den Wert des letzten COLOR-Befehles aufbewahrt.

DRAWTO

Fast noch einfacher ist das Ziehen von Grafik-Linien. Wieder muß der Endpunkt der Linie durch einen POSITION-Aufruf festgelegt werden. Als Anfangspunkt wird die intern gespeicherte Position des letzten Grafik-Befehles verwendet. DRAW ist eine Spezialfunktion des Display-Handlers und wird mit dem CIO-Kommando CDRAW (\$11) angesprochen. Die Farbe der Linie wird wieder durch ATACHR bestimmt.

FILL

Ganz ähnlich zu DRAW ist das recht eigenwillige Fill-Kommando, das nicht, wie bei anderen Computern, eine beliebige umrandete Fläche einfärbt, sondern nur eine Fläche in horizontaler Richtung zwischen zwei Linien ausfüllt. Da die wenigsten Leute sich mit dieser Eigenart des Atari Betriebssystems auskennen, soll hier zuerst die Arbeitsweise erläutert werden: Zum Zeichnen eines beliebigen Viereckes beginnt man mit der rechten Begrenzung, angefangen von der rechten unteren Ecke, in die rechte obere Ecke geplottet, und schließlich der DRAWTO-Befehl zur linken unteren Ecke durch den (in BASIC mit XIO 18... aufgerufenen) FILL-Befehl ersetzt.

Fill selbst zeichnet eine Linie vom letzten gespeicherten bis zum neu angegebenen Punkt und füllt die jeweilige horizontale Linie (nach rechts) solange, bis ein Punkt mit einer Farbe ungleich Null angetroffen wird. Wie Draw ist Fill ein Spezialbefehl des Display-Handlers, der über CIO mit dem Befehlscode CFILL (\$12, daher auch XIO 18) aktiviert wird. Die zum Füllen benutzte Farbe wird der Speicherzelle FILLDAT (\$2FD) entnommen. Im Demoprogramm wird der Fill-Routine die Farbe jedoch im Akku übergeben.

Im Hauptprogramm des Assemblerlistings (Label START) sind als Referenz die umgesetzten BASIC-Befehle als Kommentar angegeben, so daß Sie die Umsetzung leicht verfolgen

können. Sie sehen, wenn man die passenden Unterprogramme hat, dann ist das gar nicht allzu schwer. Ich konnte mich jedoch nicht zurückhalten, noch einen Regenbogen für den Hintergrund einzubauen. Das ist wirklich einer der schönsten Effekte, den der Atari zu bieten hat.

Wenn Sie nun den Unterschied in der Geschwindigkeit betrachten, mit der die Grafik in BASIC und im Assemblerprogramm aufgebaut wird, werden Sie erkennen, daß dieser gar nicht so umwerfend ist, wie man das von einem Maschinenprogramm erwartet hätte. Leser der letzten Assemblerecke können sich dafür schon eine Erklärung zurechtlegen: Da das Assemblerprogramm dieselben OS-Routinen wie BASIC benutzt, geht's eben nicht viel schneller, da das OS recht allgemein programmiert ist. Wer Linien in High-Speed zeichnen will, der muß seine eigenen, z. B. speziell auf eine Grafikstufe ausgelegten Grafik-Routinen codieren. Aber das könnte das Thema einer weiteren Assemblerecke sein.

Das Assemblerprogramm ist mit ATMAS-II geschrieben, kann aber recht leicht für Atari-Assembler oder MAC/65 umgesetzt werden. Ich habe mir vorgenommen, bei künftigen Assemblerprogrammen zwischen dem Atari-Assembler und ATMAS-II abzuwechseln, damit jeder ein Programm für "seinen" Assembler vorfindet. Übrigens werden dank der durchaus positiven Leserresonanz auch zukünftig ACTION!-Listings auf diesen Seiten erscheinen. Unter anderem fragte ein Leser an, ob ACTION! wohl auch für den 520er erscheinen werde. Interessanterweise habe ich tatsächlich ein Gerücht gehört, daß OSS bereits an einer 520er-Version arbeitet. Wann die aber erscheinen wird, steht noch in den Sternen.

Hier noch eine kleine Themenvorschau für die Assemblerecken des nächsten Jahres: Floating Point Arithmetik, Sector-Copy für 1050-Drives und Hardware-Timer.

Peter Finzel

Listing 1

```

100 REM *****
110 REM LISTING 1: "MANDALA"
120 REM
130 REM frei nach ATARI USER, Aug.85
140 REM *****
150 REM
160 MX=79
170 MY=47
180 SCHRITT=5
200 GRAPHICS 7+16
210 REM * FILL fuer Hintergrund
220 COLOR 3
230 PLOT 126,95
240 DRAWTO 126,0
250 DRAWTO 32,0
260 POSITION 32,95
270 POKE 765,3
280 XIO 18,#6,0,0,"S:"
300 REM * Hyperbel-Koerper
310 COLOR 1
320 FOR LAUF=MY TO 0 STEP -SCHRITT
330 PLOT MX-LAUF,MY
340 DRAWTO MX,LAUF
350 DRAWTO MX+LAUF,MY
360 DRAWTO MX,MY+MY-LAUF
370 DRAWTO MX-LAUF,MY
380 NEXT LAUF
390 GOTO 390

```

Listing 2

```

*****
* LISTING 2:
*
*      Bit-Map-Graphik
*      in Maschinensprache
*
*      Umsetzung des BASIC-Listings 1
*
* Assembler: ATMAS-II  PETER FINZEL
*****
* IOCB-Struktur:
*
ICCOM EQU $342
ICSTA EQU $343
ICBAL EQU $344
ICBAH EQU $345
ICBLI EQU $348
ICBLH EQU $349
ICAX1 EQU $34A
ICAX2 EQU $34B

CIOV EQU $E456  CIO-Vektor

* CIO-Befehle

COPEN EQU 3
CCLSE EQU 12
CPBIN EQU 11    hier: PLOT
CDRAW EQU 11    DRAWTO
CFILL EQU 12    XIO 18

* ATARI Graphik-Variable

ATACHR EQU $2FB  Farbe f. Draw
FILLDAT EQU $2FD  Farbe fuer Fill
ROWCRS EQU $54   Cursor-
COLCRS EQU $55   Position
RTCLK EQU $12    Frame-Zaehler
VCOUNT EQU $D40B Raster-Zeile
WSYNC EQU $D40A Synchronisierung
COLPFO EQU $D016 Farbreger

* Konstante des Demo-Programmes

```

```

MX EQU 79      Mittelpunkt X
MY EQU 47      Mittelpunkt Y
SCHRITT EQU 5  Schrittweite

ORG $A800 im res. Bereich

*****
* 'Mandala'
*
* Demo-Programm zeichnet einen
*      Hyperbel-Koerper
*
*****
* GRAPHICS 7+16
*
START LDA #7+16  GRAPHICS 7,
      JSR GRAPHICS  ganzen Screen
*
* COLOR 3:PLOT 126,95
*
      LDA #3
      STA ATACHR  COLOR 3
      LDX #126    Rechteck ausfuellen,
      LDY #95     in das die
      JSR PLOT    Figur gezeichnet wird
*
* DRAWTO 126,0: DRAWTO 32,0
*
      LDX #126
      LDY #0
      JSR DRAW
      LDX #32
      LDY #0
      JSR DRAW
*
* POKE 765,3: POSITION 32,95: XIO18, #6,0,0, "S:"
* (wird durch FILL etwas eleganter)
*
      LDA ATACHR  Farbe wie DRAW
      LDX #32
      LDY #95
      JSR FILL
*
* COLOR 1: FOR LAUF=MY TO ...
*
      LDA #1      COLOR 1
      STA ATACHR
      LDA #MY
      STA LAUF

```

```

*
* PLOT MX-LAUF,MY
*
FORNXT LDA #MX      Beginn der
      SEC           Schleife
      SBC LAUF
      TAX
      LDY #MY
      JSR PLOT
*
* DRAWTO MX,LAUF
*
      LDX #MX
      LDY LAUF
      JSR DRAW
*
* DRAWTO MX+LAUF,MY
*
      LDY #MY
      LDA #MX
      CLC
      ADC LAUF
      TAX
      JSR DRAW
*
* DRAWTO MX,MY+MY-LAUF
*
      LDA #MY      ;statt MY+MY
      ASL
      SEC
      SBC LAUF
      TAX
      LDX #MX
      JSR DRAW
*
* DRAWTO MX-LAUF,MY
*
      LDA #MX
      SEC
      SBC LAUF
      TAX
      LDY #MY
      JSR DRAW
*
* ... TO 0 STEP SCHRITT: NEXT LAUF
* (Beendung derFOR-NEXT Schleife)
*
      LDA LAUF      Laufvariable
      SEC
      SBC #SCHRITT  Schritt

```

```

      STA LAUF      abziehen
      BCS FORNXT   noch nicht fertig->

*
* Zur Abrundung der Graphik noch einen
* Regenbogen im Hintergrund
* (der im BASIC-Pgm nicht enthalten ist)
*
REGBOG LDA VCOUNT   Raster-Zeile
      CLC             plus
      ADC RTCLK+2     Frame-Zaehler
      STA WSYNC       in
      STA COLPF0+4    Farbregister
      JMP REGBOG

LAUF   DFB 0         Laufvariable

*****
* GRAPHICS-Unterprogramm
*
* Aufruf: JSR GRAPHICS
*
* PARAMETER:
* <A> 0 bis 15 (XL/XE)
*      0 bis 11 (400/800)
*****

GRAPHICS PHA         Graphik-Stufe merken
      LDX #$60       IOCB Nr. 6
      LDA #CCLSE     Screen-IOCB zuerst
      STA ICCOM,X    schliessen
      JSR CIOV
      PLA            Graphik-Stufe
      STA ICAX2,X     zurueckholen
      AND #$F0       und passende
      EOR #$10       Bit-Kombination
      ORA #$0C       fuer Handler
      STA ICAX1,X     herstellen
      LDA #COPEN     jetzt den Befehl
      STA ICCOM,X    zum Oeffnen des Screens

      LDA #SDEVICE   Zeiger auf Device-
      STA ICBAL,X    bezeichnung
      LDA #SDEVICE/256
      STA ICBAH,X
      JMP CIOV
      RTS

SDEVICE ASC "S:"     Display-Handler

*****
* Positionierung des Cursors
*
* Aufruf: JSR POSITION
*
* PARAMETER:
* <X>,<Y> je nach Graphikstufe
*****

POSITION STX COLCRS   Parameter in
      STY ROWCRS     Database des
      LDA #0         Display-Handlers
      STA COLCRS+1   eintragen (ACHTUNG:
      RTS            X nur bis 255!!)

*****
* Graphik-Punkte setzen
*
* Aufruf: JSR PLOT
*
* PARAMETER:
* <X>,<Y> je nach Graphikstufe
*****

PLOT     JSR POSITION
      LDX #$60       Screen-IOCB
      LDA #CPBIN     Plot geht wie
      STA ICCOM,X    PUT CHAR.
      LDA #0         nur ein
      STA ICBLL,X    Datum

      STA ICBLLH,X   Aktuelle Farbe
      LDA ATACHR
      JSR CIOV
      RTS

*****
* Graphik-Linien ziehen
*
* Aufruf: JSR DRAW
*
* PARAMETER:
* <X>,<Y> je nach Graphikstufe
*****

DRAW     JSR POSITION
      LDX #$60       Screen-IOCB
      LDA #CDRAW     Draw-Befehl
      STA ICCOM,X    in IOCB
      JSR CIOV
      RTS

*****
* FILL zum Ausfuellen von Flaechen
*
* Aufruf: JSR FILL
*
* PARAMETER:
* <A> : Farbe
* <X>,<Y> je nach Graphikstufe
*****

FILL     STA FILLDAT   Farbe fuer Fill
      JSR POSITION
      LDX #$60       Screen-IOCB
      LDA #CFILL     Fill-Befehl
      STA ICCOM,X
      JSR CIOV
      RTS

```