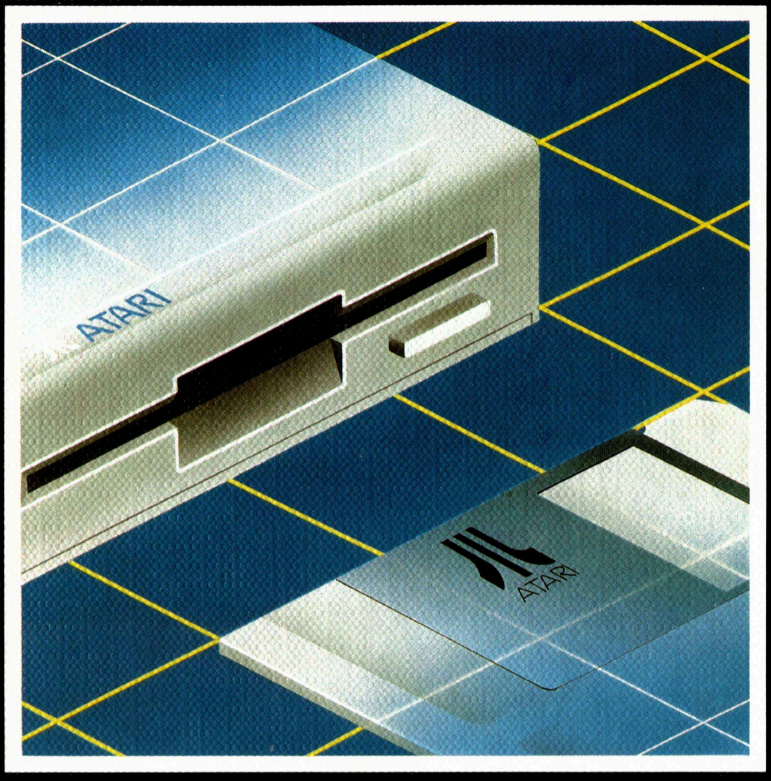
SYBEX

ATARI ST

Das Floppy Arbeitsbuch



Frank Aumann
Peter Maier
Ralf Stöpper

mit Programm-Diskette

ATARI ST
Das Floppy-Arbeitsbuch

ATARI ST
Das Floppy-Arbeitsbuch

Peter Maier
Ralf Stöpper
Frank Aumann

BERKELEY · PARIS · DÜSSELDORF

Satz: SYBEX-Verlag GmbH, Düsseldorf
Umschlaggestaltung: Daniel Boucherie / tgr
Gesamtherstellung: Druckerei Hub. Hoch, Düsseldorf

Die Reprovorlagen für dieses Buch wurden auf einem Apple Macintosh erstellt und mit einem Apple LaserWriter ausgegeben.

ISBN 3-88745-642-4
1. Auflage 1986
2. Auflage 1986

Inhaltsverzeichnis

Kapitel 1

Der Diskettenaufbau beim ATARI ST

Allgemeines über die ATARI-Laufwerke

Die Diskettenstationen SF 354 und SF 314 verarbeiten Disketten im 3,5-Zoll-Format. Während die SF 354 nur einen Schreib-/Lesekopf besitzt, kann die SF 314 auch doppelseitig bespielte Disketten bearbeiten.

Die folgende Tabelle zeigt die Formate der Disketten:

	SF 354	**SF 314**
Typ :	SSDD	DSDD
Spuren (Tracks):	80	80 (0..79)
Sektoren je Spur:	9	9 (1.. 9)
Seiten:	1	2
Kapazität:	500 kB	1 MB unformatiert
	360 kB	720 kB formatiert

Die verfügbare Kapazität von 360 kB (720 kB) wird eingeschränkt durch:

- diskетteninterne Informationen
- das Inhaltsverzeichnis
- und durch jeden weiteren Ordner (Unterverzeichnis, Folder)

Der Aufbau einer Diskettenspur – (Track)

Eine formatierte Diskette enthält im Normalfall 80 Spuren, von denen jede wiederum in 9 Sektoren zu 512 Bytes unterteilt ist. Um auf die Daten in einem bestimmten Sektor zugreifen zu können, dreht sich zum einen die Diskette, und zum andern kann der Schreib-/Lesekopf mechanisch zur Mitte bzw. zum Rand der Diskette bewegt werden. Dadurch ist es möglich, sämtlich Sektoren auf der Diskette anzufahren.

Zur Bearbeitung der Sektoren gibt es im Prinzip zwei Möglichkeiten: die *Hard-Sektorierung* und die *Soft-Sektorierung*. Weil der ATARI ST von der Soft-Sektorierung Gebrauch macht und die Hard-Sektorierung inzwischen auch keine größere Rolle mehr spielt, wird im folgenden nur auf die Soft-Sektorierung eingegegangen.

Um festzustellen, an welcher Stelle eine Spur beginnt, existiert auf der Diskette ein *Index-Loch*. Über eine Lichtschranke wird ein Impuls erzeugt, der dem FDC (Floppy-Disk-Controller) mitteilt, daß der Schreib-/Lesekopf am Anfang der Spur steht. Bei der 3,5-Zoll-Floppy entfällt das Index-Loch, dafür gibt es auf der Unterseite der Diskette in der Metallnarbe eine Einkerbung, die für die richtige Positionierung der Floppy in dem Laufwerk sorgt. Die Stellung des Schreib-/Lesekopfes wird nun vom Laufwerk bestimmt und an den FDC übergeben.

Würde man nun die Daten ohne ein spezielles Format auf die Diskette schreiben, so bekäme man beim Einlesen dieser Daten große Schwierigkeiten, weil sich die Scheibe nie mit konstanter Geschwindigkeit dreht. Zur Lösung dieses Problems wurden verschiedene Aufzeichnungsverfahren entwickelt:

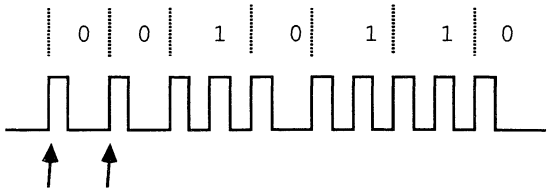
- das FM-Verfahren,
- das MFM-Verfahren,
- das M^2FM-Verfahren und
- das GCR-Verfahren.

Weil der ATARI ST das MFM-Verfahren verwendet, werden wir hier nur auf das MFM-Verfahren und seinen Vorgänger, das FM-Verfahren, eingehen.

Das einfachste Aufzeichnungsverfahren, die FM-Methode

Das FM-Verfahren (FM steht für Frequenzmodulation) ist auch unter dem Namen *Single-Density-Aufzeichnungsverfahren* bekannt. Zusätzlich zu den Datenbits werden auch Taktbits auf die Diskette übertragen. Im Abstand von 4 µs werden die Datenbits hintereinander auf die Diskette geschrieben.

Am Anfang einer solchen Bit Cell (Bitzelle) wird zunächst ein Taktimpuls geschrieben. Soll die Zelle dann das Datenbit 1 enthalten, folgt nach 2 µs wiederum ein Impuls. Bei jedem Impuls wird durch den Schreibkopf ein magnetischer Flußwechsel auf der Diskette erzeugt. Das Datenbyte $2C (Bitfolge 00101100) hätte bei dem FM-Verfahren das Aussehen wie auf der folgenden Seite gezeigt.



Das Byte $2C im FM-Format

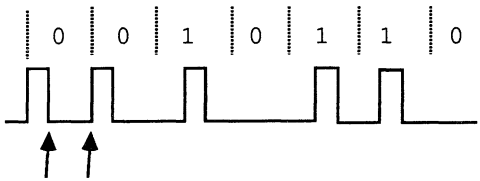
Der Nachteil des FM-Verfahrens liegt in seinem Platz- und Zeitverbrauch, da bei jedem Bit der Takt mit aufgezeichnet wird.

Das MFM-Verfahren

MFM steht für Modified Frequency Modulation und bedeutet, wie der Name schon sagt, eine Weiterentwicklung des FM-Verfahrens (Das MFM-Verfahren ist auch als *Double-Density-Aufzeichnung* geläufig). Dieses Verfahren ermöglicht die Verdoppelung der Aufzeichnungsrate, ohne daß die Anzahl der Flußwechsel erhöht wird. Die Bit Cell ist beim MFM-Verfahren nur halb so groß wie bei dem FM-Verfahren, nämlich 2 fs. Für die Aufzeichnung der Datenbits kann man von folgenden Regeln ausgehen:

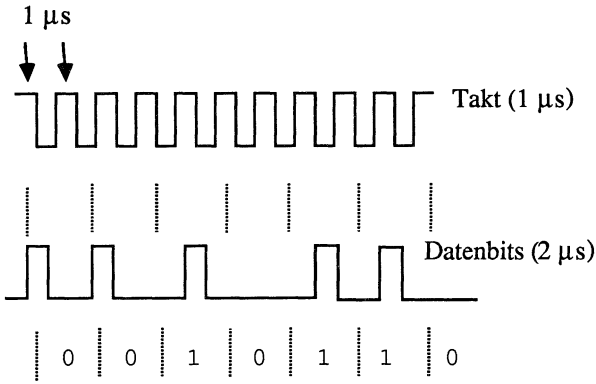
- Soll eine Zelle das Datenbit 1 enthalten, wird ein Impuls in der Mitte der Zelle aufgezeichnet.
- Bei einem Datenbit 0 wird ein Impuls am Anfang der Zelle aufgezeichnet, aber nur dann, wenn die vorherige Zelle ebenfalls ein Nullbit enthält.

Das Datenbyte $2C (Bitfolge 00101100) im MFM-Format:



Das Byte $2C im MFM-Format

Das Auslesen der Datenbits veranschaulicht die folgende Abbildung:



Auslesen der Datenbits im MFM-Format

Wenn man während eines Taktimpulses einen Datenimpuls erhält, wird er als logisch 1 interpretiert. Erhält man diesen Impuls, während kein Taktimpuls vorliegt, wird er als logisch 0 eingestuft. Nun sind wir in der Lage, die Datenbits von den Taktbits auf der Diskette zu unterscheiden. Aber wir wissen noch nicht, wann ein Datenbyte und wann ein Sektor anfängt. Deshalb gibt es Kennungen oder Marken für den Beginn von Daten- und Informationsfeldern.

GAP I	GAP II	Ad_1	GAP III	Da_1	GAP IV	GAP II	Ad_2	...

Aufbau einer Spur im MFM-Verfahren

Nach dem Index-Loch am Anfang einer Spur folgt nach ca. 92 Füllbytes (ATARI benutzt dafür das Byte $4E) die *Index-Address-Mark*, die jedoch inzwischen von den meisten Systemen nicht mehr benötigt wird (GAP I).

Für jeden Sektor folgt dann:

Eine Lücke (GAP II) am Anfang des *Adressenfeldes*, die aus zwölf $00-Bytes und drei $F5-Bytes besteht. Die $00-Bytes ermöglichen dem FDC, die Schreib-/Lese-Elektronik einzustellen, während die $F5-Bytes eine spezielle Bedeutung haben, auf die wir später eingehen werden.

Das Adressenfeld (Ad_x):

Dieses Feld beginnt mit der sogenannten ID-Address-Mark, die aus einem $FE-Byte besteht. Anschließend folgen die Informationen über die Spurnummer, die Sektornummer, die Diskettenseite, die Größe des Datensektors und eine Prüfsumme:

Marke	Spur	Seite	Sektor	Größe	Prüfsumme
$FE	0..$4F	0..1	1..9	$02	2 CRC-Bytes

Im allgemeinen sind bei der Spurnummer die Bytes $00 bis $4F erlaubt. Sie erlauben eine Numerierung zwischen 0 und 79, und zwar bei der Sektornummer die Bytes $01 bis $09 und bei der Seite $00 und $01, wobei $00 die Vorderseite und $01 die Rückseite der Diskette kennzeichnet. Die Größe eines Datensektors kann der folgenden Tabelle entnommen werden:

Byte	Größe
$00	128 Bytes
$01	256 Bytes
$02	512 Bytes
$03	1024 Bytes

Der letzte Eintrag in einem Feld ist die Prüfsumme. Dem FDC wird hierbei ein $F7-Byte übermittelt; dies hat ebenfalls eine bestimmte Bedeutung und soll später erklärt werden. Als nächstes folgt die Lücke vor dem *Datenfeld* (GAP III), die aus 22 Füllbytes ($4E), zwölf $00-Bytes und drei $F5-Bytes besteht.

Auf diese Lücke folgt das Datenfeld:

Marke	512 Bytes Daten	Prüfsumme
$FB	$E5 (neu formatiert)	2 CRC-Bytes

Wie das $FE-Byte das Adressenfeld, kennzeichnet das $FB-Byte ein Feld: das Datenfeld. Danach folgen so viele Datenbytes wie im Adressenfeld angegeben sind. Bei einer neuformatierten Diskette wären das 512 $E5-Bytes. Am Ende steht wieder die Prüfsumme.

Am Ende eines jeden Sektors folgt wiederum eine Lücke, die GAP IV. Diese besteht aus 40 Füllbytes. Nachdem alle Sektoren geschrieben wurden, wird

die Spur bis zum erneuten Auftreten des Index-Impulses, der das physikalische Ende der Spur kennzeichnet, mit $4E-Bytes beschrieben.

Spezielle Bitfolgen

Wie kann der FDC nun die Marken von normalen Datenbytes unterscheiden? Dazu wurden spezielle Bitfolgen gewählt, bei denen auch die Taktbits eine Rolle spielen. Durch das Weglassen bestimmter Taktbits wird ein Bitmuster erzeugt, das sonst nie auftreten kann. Der FDC kann durch seine Elektronik diese Muster erkennen und dementsprechend darauf reagieren.

Bei dem MFM-Verfahren ist diese Sache im Gegensatz zum FM-Verfahren nicht so einfach, weil dort bereits Taktimpulse durch das dichtere Aufzeichnungsverfahren fehlen. Daher werden nur zwei Bitfolgen verwendet: die $A1-Folge und die $C2-Folge.

Die $C2-Folge wird dreimal hintereinander aufgezeichnet. Danach folgt die Erkennungsmarke für das Index-Loch mit dem Code $FC, der mit allen Taktimpulsen aufgezeichnet wird. Die $C2-Folge dient also zum *Identifizieren der Index-Mark.*

Auch das $A1-Muster wird dreimal hintereinander aufgezeichnet. Anstelle der Index-Loch-Markierung folgt nun entweder ein $FE-Byte als ID-Address-Mark oder ein $FB-Byte, das ein Datenfeld identifiziert.

Die Prüfsumme (Cyclic Redundancy Check)

Die CRC-Bytes dienen, wie schon erwähnt, zur Fehlererkennung. CRC ist eine Abkürzung für *Cyclic Redundancy Check*, was man im Prinzip als Prüfsumme bezeichnen kann. Im eigentlichen Sinn ist dies aber keine Aufsummierung einzelner Bytes. Das Verfahren zur Erzeugung der CRC-Bytes ist wesentlich schwieriger. Hierbei werden nur bestimmte Bits, die aus dem Generator-Polynom

$$x^{16}+x^{12}+x^5+1$$

erzeugt werden, verwendet.

Das Formatieren einer Diskette

Nun hat man ein Feld voller Daten, Lücken und Markierungen. Wie bekommt man diese nun auf die Diskette? Diesen Vorgang nennt man die Formatierung einer Spur, weil die Spur auf ein spezielles Format gebracht wird. Die Formatierung ist im Prinzip nichts anderes als das Schreiben dieses Feldes auf die Diskette.

Der FDC des ATARI ST besitzt hierfür den *WRITE-TRACK-Befehl*, der eine ganze Spur beschreibt. Dieser Befehl interpretiert nun verschiedene Bytes innerhalb des Feldes als Befehle und nicht als Daten. So müssen z.B. die beiden CRC-Bytes am Ende eines Feldes erzeugt und die speziellen Bytes bei den Adreßmarken und der Index-Mark geschrieben werden. Die Bitfolgen haben im FM- und im MFM-Verfahren eine unterschiedliche Bedeutung. Für das MFM-Verfahren sind die folgenden Bytes reserviert:

Bitfolge	Bedeutung
$00-$F4	Schreibe das Byte normal im MFM-Format.
$F5	Schreibe $A1 im MFM-Format mit fehlendem Taktbit zwischen Bit 4 und 5.
$F6	Schreibe $C2 im MFM-Format mit fehlendem Taktbit zwischen Bit 3 und 4.
$F7	Schreibe 2 CRC-Bytes.
$F8-$FF	Schreibe das Byte normal im MFM-Format.

Die Bytes $F5, $F6 und $F7 sollten daher *niemals* als Füllwert beim Formatieren einer Spur bzw. einer Diskette verwendet werden.

Die Unterteilung der Diskette in logische Sektoren

Das Betriebssystem des ATARI ST kennt nicht nur die Unterteilung der Diskette in Spur, Sektor und Seite, sondern auch die in sogenannte *logische Sektoren.*

Bei dieser Unterteilung werden die Sektoren beginnend mit dem logischen Sektor 0 fortlaufend numeriert. Bei einer einseitigen Diskette wäre der logische Sektor 9 identisch mit dem physikalischen Sektor 1 auf der Spur 1. Auf einer doppelseitigen Diskette entspricht der logische Sektor 9 dem physikalischen Sektor 1 auf Spur 0 der Seite 1.

Die Umrechnung von Spur, Sektor und Seite in logische Sektoren kann man mit der folgenden Formel durchführen:

log = NSECTS * (Spur * NSIDES + Seite) + Sektor – 1

wobei

- NSECTS die Anzahl der physikalischen Sektoren je Spur ist,
- NSIDES die Anzahl der Seiten auf der Diskette (1 oder 2) ist und
- Seite entweder 0 oder 1 ist.

Die Umkehrung, also die Berechnung von Spur, Sektor und Seite aus dem logischen Sektor, ergibt sich aus den folgenden Formeln:

Seite = (log div NSECTS) mod NSIDES
Spur = log div (NSIDES * NSECTS)
Sektor = 1 + (log mod (NSIDES * NSECTS)) – (NSECTS * Seite)

Für eine einseitige Diskette mit 9 Sektoren je Spur ergibt sich also:

log = 9 * Spur + Sektor – 1

Seite = 0
Spur = log div 9
Sektor = (log mod 9) + 1

und für eine zweiseitige Diskette:

log = 9 * (Spur * 2 + Seite) + Sektor – 1

Seite = (log div 9) mod 2
Spur = (log div 18)
Sektor = (log mod 18) – 9 * Seite + 1

(*div* bezeichnet die Division *ohne Rest*, *mod* den Rest der Division.)

Die Systeminformationen der Diskette

Die ersten beiden Spuren nehmen eine Sonderstellung auf der Diskette ein. Hier werden nur Systeminformationen gespeichert und keine Daten des Anwenders.

Als Systeminformationen gelten der *Bootsektor*, die *File Allocation Table* (Datei-Belegungstabelle) und das Inhaltsverzeichnis, das *Directory*.

Diese befinden sich bei einer einseitigen Diskette auf:

	Sektor	Information
Spur 0:	1	Bootsektor
	2-6	File Allocation Table – Kopie
	7-9	File Allocation Table
Spur 1:	1-2	File Allocation Table
	3-9	Inhaltsverzeichnis

Im allgemeinen gilt:

logischer Sektor	Information
0	Bootsektor
1- 5	Kopie der FAT
6-10	File Allocation Table
11-17	Inhaltsverzeichnis
ab 18	Benutzerdaten

Der logische Sektor Null – Der Bootsektor

Der *Bootsektor* hat nicht nur die Aufgabe, sämtliche Informationen zu speichern, die beim Formatieren der Diskette notwendig waren, er ermöglicht auch bei einem *Reset* des Rechners ein Programm automatisch von der Diskette zu starten (zu *booten*). Die Werte sind hierbei im 8086-Format, also low-Byte/high-Byte, abgespeichert.

Der Bootsektor besteht insgesamt aus drei Teilen:

- einer Seriennummer
- den Disketteninformationen (BIOS-Parameter-Block, BPB)
- einem (optionalen) Ladeprogramm

Aufbau des Bootsektors

Bytes	*Bedeutung*	*Wert (*)*
$00 - $01	<BRA>	$60 $38
$02 - $07	Füller	"Loader"
$08 - $0A	Seriennummer	$xxxxxx
$0B - $0C	BPS	$00 $02 = 512
$0D	SPC	$02
$0E - $0F	RES	$01 $00 = 1
$10	NFATS	$02
$11 - $12	NDIRS	$70 $00 = 112
$13 - $14	NSECTS	$D0 $02 = 720
$15	MEDIA	$F8
$16 - $17	SPF	$05 $00 = 5
$18 - $19	SPT	$09 $00 = 9
$1A - $1B	NSIDES	$01 $00 = 1
$1C - $1D	NHID	$00 $00
$1E - $1FF	<Ladeprogramm>	$xx...

(*) Die Werte sind der TOS-Systemdiskette (einseitig) entnommen.

Die Abkürzungen bedeuten:

<BRA> Sprung (BRAnch) zum Lade-Programm. Dieser Sprung wird nur ausgeführt, wenn die Prüfsumme des Bootsektors dem Wert $1234 entspricht.

Füller	Diskettenbezeichnung.
Seriennummer	24-Bit-Nummer.
BPS	Bytes je Sektor, normalerweise 512.
SPC	Sektoren je Cluster. Der Rechner faßt je zwei Sektoren zu einem Cluster (Gruppe) zusammen.
RES	Reservierte Sektoren am Anfang einer Diskette, inklusive des Bootsektors; normalerweise 1.
NFATS	Anzahl der File Allocation Tables auf der Diskette.
NDIRS	Maximale Inhaltsverzeichniseinträge. Insgesamt112 auf einer einseitigen Diskette.
NSECTS	Gesamtanzahl der Sektoren auf der Diskette, 720 auf einer einseitigen.
MEDIA	Diskettenformat wird vom ATARI ST-BIOS nicht benutzt.
SPF	Sektoren je File Allocation Table.
SPT	Sektoren je Spur.
NSIDES	Anzahl der Diskettenseiten (1 oder 2).
NHID	Anzahl der versteckten Sektoren, wird vom ATARI ST-BIOS ignoriert.

Das Ladeprogramm

Das *Ladeprogramm* ist ein Assembler-Programm und steht im Bootsektor ab der Position $3A. Mit diesem *Loader* kann man entweder einen *Image-File* oder einen zusammenhängenden Block von der Diskette in den Rechnerspeicher laden und anschließend auch ausführen lassen. Ein Image-File ist ein genaues Abbild von dem auszuführenden Programm und enthält daher weder einen Programmkopf noch Relokations-Informationen.

Zu dem Ladeprogramm gibt es noch einige Informationen, die im Bootsektor ab der Position $1E abgespeichert sind.

Hier bedeuten die Abkürzungen:

EXECFLG	Ein Wort (Integer), das an die Adresse _cmdload in den Rechner kopiert wird. Ist _cmdload ungleich Null, dann wird beim Systemstart ein Programm mit dem Namen COMMAND.PRG geladen.
LDMODE	kennzeichnet den Lademodus. Ist LDMODE gleich Null, dann sucht das System nach einem File, das unter FNAME angegeben ist und lädt es. Ist LDMODE ungleich Null, werden NSECT Sektoren, angefangen mit dem logischen Sektor SSECT geladen.

SSECT, NSECT — siehe unter LDMODE.

LDADDR — an diese Adresse wird das File oder der Block geladen.

FATBUF — Adresse, an die die File Allocation Table und die Directory geladen werden.

FNAME — siehe unter LDMODE.

CHECKSUM — soll der Bootsektor ausführbar sein, so muß die Prüfsumme des Sektors $1234 entsprechen. Die letzten beiden Bytes werden zum Entstehen dieser Prüfsumme benötigt.

$1E - $1F	EXECFLG	$00 $00
$20 - $21	LDMODE	$00 $00
$22 - $23	SSECT	$00 $00
$24 - $25	NSECT	$00 $00
$26 - $29	LDADDR	$00 04 00 00
$2A - $2D	FATBUF	$00 00 80 00
$2E - $38	FNAME	"TOS IMG"
$39	DUMMY	$00
$3A - $1fD	Ladeprogramm	$33 $FA ...
$1fe - $1ff	CHECKSUM	

Als Beispiel eines Ladeprogramms haben wir im folgenden den Booter der einseitigen TOS-Systemdiskette als kommentiertes Assembler-Listing aufgeführt.

```
* ------------------------------------------------------------
*
*                        TOS-Booter
*
* ------------------------------------------------------------

*
*             Konstanten innerhalb des Programms
*

bios        .equ    13          ; TRAP-Nummer des BIOS
_cmdload    .equ    $482        ; für COMMAND.PRG
_bootdev    .equ    $446        ; Laufwerknummer
_membot     .equ    $432        ; Beginn des Benutzerspeichers
                                ; (TPA)

*
*         Struktur des BIOS-Parameter-Blocks
*

recsiz      .equ    0           ; Größe des Sektors in Bytes
clsiz       .equ    2           ; Größe der Gruppe in Sektoren
clsizb      .equ    4           ; Größe der Gruppe in Bytes
rdlen       .equ    6           ; Größe der Direc. in Sektoren
fsiz        .equ    8           ; Größe der FAT in Sektoren
fatrec      .equ    10          ; Erster log. Sektor der FAT
datrec      .equ    12          ; Log. Sektor der ersten
                                ; Datengruppe
numcl       .equ    14          ; Anzahl der Datengruppen
                                ; auf der Diskette

*
*
*   Das Listing beginnt mit dem ersten Byte des Bootsektors
*
*

BOOTER:

            bra     CODE        ; Sprung zum Ladeprogramm

*
*       Es folgt nun der BIOS-Parameter-Block, BPB
*

            .dc.b   'Loader'    ; Diskettenbezeichnung
            .dc.b   0,0,0       ; Seriennummer
            .dc.b   0,2         ; Bytes je Sektor
            .dc.b   2           ; Sektoren je Gruppe
            .dc.b   1,0         ; Reservierte Sektoren
            .dc.b   2           ; Anzahl der FATs
            .dc.b   $70,0       ; Directory-Einträge
            .dc.b   $D0,2       ; Gesamtzahl der Sektoren
            .dc.b   $F8         ; Kennzeichnung der Disk
            .dc.b   5,0         ; Sektoren je FAT
```

```
            .dc.b   9               ; Sektoren je Spur
            .dc.b   0               ; Nicht benutzt
            .dc.b   1               ; Seiten der Disk
            .dc.b   0,0             ; Versteckte Sektoren
EXEC_FLG:   .dc.b   0,0             ; Kein COMMAND.PRG
LDMODE:     .dc.b   0,0             ; Lademodus
SEC_START:  .dc.b   0,0             ; Nicht benutzt, da der
NR_SECT:    .dc.b   0,0             ; Lademodus Null ist
LOAD_ADR:   .dc.l   $40000          ; Ladeadresse
FAT_ADR:    .dc.l   $8000           ; FAT-Adresse
FILE:       .dc.b   'TOS IMG',0     ; Name des IMAGE-Files


CODE:

*
*  Zuerst wird EXEC_FLG in die Speicheradresse $482 kopiert
*

            move.w  EXEC_FLG(pc),_cmdload

*
* BPB über BIOS-Aufruf getbpb in den Speicher holen
*

            move.w  _bootdev,-(a7)  ; Geräteadresse auf Stack
            move.w  #7,-(a7)        ; Funktionsnummer von getbpb
            trap    #bios           ;
            addq.l  #4,a7           ; Stack aufräumen
            tst.l   d0              ; Adresse des BPB = 0?
            beq     ENDE            ; Ja, dann Fehler
            move.l  d0,a5           ; Adresse des BPB nach A5
            lea     FAT_ADR(pc),a0  ; Adresse der FAT nach A0
            tst.l   (a0)            ; = 0?
            bne     FATOK           ; Nein, dann weitermachen
            move.l  _membot,(a0)    ; Adresse FAT auf den Start
                                    ; des Benutzerspeichers
                                    ; legen

FATOK:      move.w  fsiz(a5),d0     ; SPF nach d0
            lsl     #8,d0           ; mit 512
            add.l   d0,d0           ; multiplizieren
            move.w  d0,a4           ;
            add.l   FAT_ADR(pc),a4  ; A4 zeigt auf das ENDE
                                    ; der FAT im RAM
            move.w  LDMODE(pc),d0   ; LDMODE gleich Null?
            beq     LD_FILE         ; Ja, TOS als Datei laden

*
* Der folgende Programmteil wird normalerweise nie ausgeführt
*

            move.w  SEC_START(pc),d6; Log. Startsektor des TOS
            move.w  NR_SECT(pc),d4  ; Anzahl der Sektoren
            move.l  LOAD_ADR(pc),a3 ; Startadresse im RAM
            bra     EXEC            ; Laden und Starten
```

```
*
* Das Betriebssystem wird als IMAGE-File eingeladen (TOS.IMG)
*
LD_FILE:    move.w  fatrec(a5),d6   ; Sektor der 2. FAT
            move.w  fsiz(a5),d4     ; Größe der FAT
            add.w   rdlen(a5),d4    ; DIR-Länge
            move.l  FAT_ADR(pc),a3  ;
            bsr     GETSEC          ; FAT und Directory ohne
                                    ; Starten laden
            bne     ENDE            ; Fehler?
            move.l  a4,a0           ; Ende der FAT im RAM
            move.w  rdlen(a5),d0    ;
            lsl.w   #8,d0           ; Directory-Länge in Bytes
            lsl.w   #1,d0           ;
            lea     0(a0,d0.w),a0   ; A0 zeigt auf das Ende der
                                    ; des Directory im RAM

            lea     FILE(pc),a1     ; Zeiger auf Filenamen nach
                                    ; A1
NEXT:       sub.w   #$20,a0         ; Nächster Name
            cmp     a4,a0           ; Ende der FAT erreicht?
            blt     ENDE            ; Ja, dann Fehler!
            moveq   #10,d0          ; Länge des Namens - 1

CHAR:       move.b  0(a0,d0.w),d1   ; Stimmt dieses Zeichen
            cmp.b   0(a1,d0.w),d1   ; überein?
            bne     NEXT            ; Nein, dann nächste Datei
                                    ; untersuchen
            dbf     d0,CHAR         ; Ja, das nächste Zeichen
                                    ; holen (oder fertig)

            moveq   #0,d7           ; Register D7 löschen
            move.b  $1b(a0),d7      ; 1. Sektorengruppe der
                                    ; Datei
            lsl.w   #8,d7           ; in das Register D7
            move.b  $1a(a0),d7      ; bringen
            move.l  FAT_ADR(pc),a6  ; Startadresse der FAT in A6
            move.l  LOAD_ADR(pc),a3 ; Startadresse des TOS in A3
            clr.l   d4              ; Anzahl der Sektoren auf
                                    ; Null setzen
TEST:       cmp.w   #$0ff0,d7       ; D7 >= $FF0?
                                    ; (letzte Gruppe)
            bge     EXEC            ; Ja, dann laden und starten
            move.w  d7,d3           ; Aktueller Sektor ist
            subq.w  #2,d3           ; (Gruppe - 2) * Größe der
            mulu    clsiz(a5),d3    ; Gruppe+erster Datensektor
            add.w   datrec(a5),d3   ; Aktuellen Sektor nach D3
            cmp     #$40,d4         ; Anzahl der Sektoren > $40?
            bge     GE_40           ; Ja, dann lade bis hierher
            tst.w   d4              ; D4 = 0 ?
            beq     FIRST           ; Ja, dann erste Gruppe
            cmp.w   d5,d3           ; Wurde der Sektor erwartet?
            beq     SECOK           ; Ja, dann teste weiter
```

```
GE_40:      bsr     GETSEC          ; Nein, lade bis hierher
            bne     ENDE            ; Fehler
            lsl.l   #8,d4           ; Anzahl der geladenen
                                    ; Sektoren mal
            lsl.l   #1,d4           ; 512 = Anzahl der geladenen
                                    ; Bytes
            add.l   d4,a3           ; + alte Startadr. = neue
                                    ; Startadr.

FIRST:      move.w  d3,d6           ; D6 = 1. Sektor fuer GETSEC
            move.w  d3,d5           ;
            clr.l   d4              ; Anzahl Sektoren auf Null

SECOK:      add.w   clsiz(a5),d4    ; Eine Gruppe mehr an
                                    ; Sektoren
            add.w   clsiz(a5),d5    ; Nächster erwarteter Sektor
            move.w  d7,d2           ; Aktuelle Gruppe
            lsr.w   #1,d2           ; Mit 1.5 multiplizieren
            add.w   d7,d2           ;
            move.b  1(a6,d2.w),d1   ; Als Zeiger in der FAT
            lsl.w   #8,d1           ;
            move.b  0(a6,d2.w),d1   ; Hole die nächste Gruppe
            btst    #0,d7           ; 12 Bits müssen isoliert
                                    ; werden:
            beq     ODD:            ; ungerade, dann Bits 0-11
            lsr     #4,d1           ; gerade, dann Bits 4-15

ODD:        and.w   #$0fff,d1       ;
            move.w  d1,d7           ; Nächste Gruppe
            bra     TEST            ; und testen

*
*  Programm laden und starten
*

EXEC:       tst.w   d4              ;
            beq     EX_1            ; Anzahl der Sektoren = 0
                                    ; dann sofort starten
            bsr     GETSEC          ; Sektoren holen
            bne     ENDE            ; Fehler

EX_1:       move.l  LOAD_ADR(pc),-(a7)
            rts

ENDE:       clr.l   d0
            rts

GETSEC:     move.w  _bootdev,-(a7) ; Laufwerk,
            move.w  d6,-(a7)       ; 1. log. Sektor,
            move.w  d4,-(a7)       ; Anzahl der Sektoren,
            move.l  a3,-(a7)       ; Startadresse und das
            clr.w   -(a7)          ; rw-Flag auf den Stack
            move.w  #4,-(a7)       ; rwabs-Aufruf
            trap    #bios          ;
            add.w   #14,a7         ; Stack wieder aufräumen
            tst.w   d0             ; Fehler-Flag testen
            rts                    ;
```

Die File Allocation Table

Sämtliche Information über die Diskettenausnutzung stehen in der "Datei-Belegungstabelle", *File Allocation Table* (FAT). Beim ATARI ST werden zwei dieser Tabellen verwendet, die hinter dem reservierten Sektor (Bootsektor) hintereinander auf die Diskette geschrieben werden. Die erste FAT beginnt daher bei dem üblichen ATARI-Format ab dem logischen Sektor 1 und die zweite ab dem logischen Sektor 6. Das Betriebssystem des ATARI ST benutzt nur die zweite FAT, die erste steht als Kopie auf der Diskette.

Das TOS faßt jeweils zwei Sektoren zu einem Cluster (Gruppe) zusammen. Der erste Cluster, der für die Speicherung der Dateien zu Verfügung steht, ist der Cluster 2, der auf einer einseitigen Diskette bei Spur 2, Sektor 1 bzw. ab dem logischen Sektor 18 beginnt.

Jeder Eintrag in der FAT ist 12 Bits = 1.5 Bytes groß.

Bedeutung der FAT-Einträge

Die FAT-Einträge haben zwei Funktionen. Zum einen zeigen sie dem TOS an, welche Gruppen auf der Diskette noch frei sind, zum anderen verbinden sie diese zu einem File. Jeder Directory-Eintrag eines Files enthält die Nummer des ersten Clusters der Datei, die weiteren lassen sich über den folgenden Algorithmus berechnen:

1. CL ist die Nummer des gegebenen Clusters.

2. Der FAT-Eintrag F ergibt sich aus

```
F = CL + CL / 2;
```

3. Aus der 16-Bit-Zahl an der Stelle F in der FAT errechnet sich nun die nächste Cluster-Nummer NCL, wobei zu beachten ist, daß die Einträge im 8086-Format abgespeichert sind (erst das Low-Byte, dann das High-Byte)!

4. Falls CL eine ungerade Zahl ist, müssen die oberen 12 Bits in die unteren 12 Bits kopiert werden:

```
NCL := SHR(NCL,4);
```

5. Da ein Eintrag nur 12 Bits groß ist, werden nun die relevanten Bits ausmarkiert:

```
NCL = NCL & $0FFF;
```

Um alle Gruppennummern eines Files zu bekommen, führt man den Algorithmus so lange durch, bis die Nummer größer als $FF7 ist.

Wenn das TOS ein File auf der Diskette abspeichert, prüft es den Eintrag auf den Wert $000. Ist dieser Eintrag $000, dann ist dieser Cluster nicht belegt und kann zur Datenspeicherung benutzt werden. Liegt der Wert des Eintrages zwischen $FF1 und $FF7, dann ist auf der Diskette ein Schreibfehler aufgetreten, der die Benutzung des Clusters verbietet. In jedem anderen Fall ($002 - $FF0) ist diese Gruppe schon belegt und kann zur Datenspeicherung nicht benutzt werden.

Will das TOS nun eine Datei lesen, dann geht es diese Datei Cluster für Cluster durch und benutzt den Wert jeweils als Zeiger auf den nächsten Cluster. Ist der Eintrag $000 oder $001, dann ist die Diskette fehlerhaft, da das TOS diese Clusternummer nicht vergibt. Trifft das TOS trotzdem auf diesen Eintrag, so zeigt er auf den Cluster $000, bzw. auf den Cluster $001.

Liegt der Wert des Eintrages zwischen $002 und $FF7, dann ist diese Gruppe bereits belegt, und der Wert weist auf den der Datei folgenden Block. Allerdings ist dabei zu beachten, daß der Wert die Anzahl der Cluster nicht übersteigt, da der Lesekopf diesen Cluster sonst nicht finden kann. Ist er größer als $FF7, so ist diese Gruppe die letzte in der Datei.

$000	Freie Gruppe
$001	Nicht möglich
$002 - $FEF	Nächste Gruppennummer
$FF0 - $FF7	Fehlerhafte Gruppe
$FF8 - $FFF	Ende der Datei

Die ersten beiden Einträge in der FAT haben eine besondere Bedeutung, da sie nicht zu Clustern des Datenbereichs gehören. Diese Cluster liegen in der Directory und werden normalerweise vom TOS nicht benutzt. Den beiden Einträgen werden aus Sicherheitsgründen Werte zugewiesen, die EOF bedeuten. In Anlehnung an MS-DOS wird in das erste Byte eine Kennung für das Medium geschrieben, aber ansonsten nicht benutzt.

Einige Werte für das Medium:

$F8 – Single Sided/80 Tracks
$F9 – Double Sided/80 Tracks
$FC – Single Sided/40 Tracks
$FD – Double Sided/40 Tracks

Falls Sie diese Einträge ändern, sollten Sie beachten, daß der auf den Cluster 0 folgende Block jeweils der Startblock der Datei ist. Das TOS beachtet also den Eintrag für den nullten Cluster nicht. Außerdem werden diese beiden Cluster vom TOS nie zur Datenspeicherung benutzt, selbst wenn Sie die Einträge in der FAT auf $000 abgeändert haben.

Die Umrechnung von einer Gruppennummer zu dem dazugehörigen logischen Sektor kann über den folgenden Algorithmus erfolgen:

1. `log := CL - 2;`

2. Dieser Wert wird mit der Anzahl der Sektoren je Cluster multipliziert,
   `log := log * CLSIZ`

3. Dazu addiert sich der logische Sektor des ersten Clusters:

   `log := log + DATREC`

Die Werte von CLSIZ und DATREC erhält man über den GETBPB-Aufruf, der den BIOS-Parameterblock der Diskette liefert.

Das Inhaltsverzeichnis, die Directory

Die Directory ist das Inhaltsverzeichnis der Diskette. Es enthält Informationen über die Dateien auf der Diskette. Bei der ST-Serie beginnt es normalerweise ab dem logischen Sektor 11. Die Directory besteht aus einer Aneinanderreihung von Einträgen für jede Datei.

Jeder Eintrag ist 32 Bytes lang und folgendermaßen aufgebaut:

Bytes	Bedeutung
0 .. 7	Name der Datei
8 .. 10	Extension der Datei
11	Datei-Attribut
12 .. 21	nicht benutzt
22 .. 23	Uhrzeit der letzten Änderung
24 .. 25	Datum der letzten Änderung
26 .. 27	Zeiger auf den ersten Cluster der Datei
28 .. 31	Länge der Datei

Die Einträge des Directories bestehen aus folgenden Angaben, deren Format näher beschrieben wird:

Name und Extension

Hier steht der Name der Datei, wie er auch im Desktop erscheint, allerdings ohne den trennenden Punkt.

Datei-Attribut

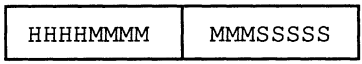
Das Attribut ist bitweise verschlüsselt:

Bit	Bedeutung, wenn gesetzt
0	nur lesen
1	versteckt
2	Systemdatei
3	Diskettenname
4	Unterverzeichnis
5	Archivstatus
6	- nicht benutzt
7	- nicht benutzt

Uhrzeit

Auch die Uhrzeit und das Datum sind bitweise verschlüsselt, stehen aber im 8086-Format auf der Diskette:

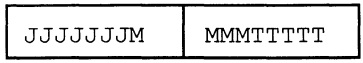
15 ..... 8 7 0



Bit 0 - 4 Sekunden
Bit 5-10 Minuten
Bit 11-15 Stunden

Datum

15 ..... 8 7 0



Bit 0- 4 Tag
Bit 5- 8 Monat
Bit 9-15 Jahr - 80

Länge

Auch die Länge der Datei ist im 8086-Format abgespeichert:

Für die Unterverzeichnisse (Folder) der Diskette ist kein fester Platz reserviert. Jeder dieser Subdirectories wird als eine Datei auf die Diskette geschrieben, das Inhaltsverzeichnis-Einträge enthält. Die ersten beiden Einträge in einem Folder sind die speziellen Unterverzeichnisse "." und "..". Das Verzeichnis "." zeigt auf das übergeordnete Subdirectory bzw. das Hauptdirectory, ".." auf das eigene.

Wird nun eine Datei oder ein Folder gelöscht, werden die Einträge nicht ausgenullt, sondern nur das erste Byte auf den Wert $E5 gesetzt.

Kapitel 2

Diskettenprogrammierung unter TOS

Die Systemroutinen des ATARI ST-DOS

In dem folgenden Kapitel werden die Systemroutinen des ATARI ST vorgestellt, die in der Assembler-Sprache durch einen TRAP-Befehl aufgerufen werden und in irgendeiner Weise mit der Floppy zu tun haben.

Das DOS des ATARI ST ist im Prinzip in drei Teile unterteilt:

- GEMDOS
- BIOS

und das erweiterte BIOS,

- XBIOS.

Der Aufruf einer Funktion dieser drei Bereiche erfolgt in Assembler wie folgt:

1. Parameter auf dem Stack ablegen.
2. Die Funktionsnummer auf den Stack legen.
3. Den TRAP-Befehl ausführen:
   - GEMDOS = TRAP #1
   - BIOS = TRAP #13
   - XBIOS = TRAP #14
4. Den Stack wieder in Ordnung bringen.
5. Eine eventuelle Fehlermeldung aus dem Datenregister DO auslesen.

Da die Programmierung dieser Funktionen in der Sprache C sehr einfach ist, sind im folgenden die Aufrufe der Funktionen in der üblichen C-Notation angegeben. Die Funktionen sind hierbei MAKRO-Definitionen, die über den

Preprozessor des Compilers in eine der folgenden Formen überführt wird:

- (long) GEMDOS (Nummer, Parameterliste);
- (long) BIOS (Nummer, Parameterliste);
- (long) XBIOS (Nummer, Parameterliste);

GEMDOS, BIOS und XBIOS sind hierbei in der "C"-Bibliothek enthaltene (z.B. LATTICE-C) oder externe Funktionen (z.B. bei DR-C in dem Modul OSBIND.O).

Das GEMDOS

GEMDOS-Befehle weisen teilweise Ähnlichkeiten mit MS-DOS-Befehlen auf, allerdings wurden nur einige MS-DOS-Funktionen übernommen. Wenn man ein $5^1/4$-Laufwerk an den ST anschließt und eine MS-DOS-Diskette einlegt, so wird man erstaunt feststellen, daß die Directory einwandfrei eingelesen wird. Leider sind aber alle restlichen Daten auf MS-DOS-Disketten ohne Hilfsmittel nicht zu lesen.

GEMDOS-Aufrufe haben die unangenehme Eigenschaft, daß sie die Register A0 und D0 verändern. Falls Sie in Assembler programmieren, sollten Sie den Inhalt dieser Register vor dem TRAP-Aufruf retten. Nach dem Funktionsaufruf wird im Register D0 ein Wert übergeben, der auch eine Fehlermeldung sein kann.

Programmiert man in Assembler oder Pascal, muß bei der Eingabe von Datei- oder Ordnernamen darauf geachtet werden, daß der Name durch ein Nullbyte abgeschlossen wird. In C ist das nicht nötig.

Man kann die Befehle des GEMDOS in vier Kategorien einteilen:

- Datei-Befehle
- Folder-Befehle
- Directory-Befehle
- Drive-Befehle

Datei-Befehle

Die Datei-Befehle setzen sich aus folgenden Befehlen zusammen, auf die nachfolgend näher eingegangen wird:

- FCREATE $3C
- FOPEN $3D
- FCLOSE $3E
- FREAD $3F
- FWRITE $40
- FDELETE $41
- FSEEK $42
- FATTRIB $43
- PEXEC $4B
- FRENAME $56
- FDATIME $57

Kommando: **Fcreate** *Funktionsnummer*: **$3C**

Format: Fcreate (name,attr)
char *name;
WORD attr;

Funktion: Vereinbarung eines Datei- oder Diskettennamens.

Um die Diskette mit einem Namen zu versehen oder um eine Datei auf ihr anzulegen, muß man zunächst den Befehl Fcreate mit einem Namen (8 Zeichen) und einer Extension (3 Zeichen) und einem entsprechenden Attribut aufrufen.

Folgende Attributwerte sind bei *Fcreate* möglich:

$00 – diese Datei kann gelesen und beschrieben werden.
$01 – diese Datei kann nur gelesen werden.
$02 – ein sogenanntes "Hidden File" wird kreiert.
$04 – System-Datei.
$08 – Diskettenname.

Die Attributwerte $10 (Folder) und $20 können beim Kreieren einer Datei nicht angegeben werden. Für Folder gibt es einen Extrabefehl: *Dcreate*. Das Attribut $20 wird vom TOS selbst vergeben. Existiert die zu erstellende Datei schon, wird ihr die Länge 0 zugeordnet. Die Datei wird sozusagen gelöscht. Man erhält nach dem Aufruf ein File-Handle, über das später alle Zugriffe auf diese Datei laufen. Als erste freie Handle-Nummer erhält man eine 6. Es können maximal 40 Dateien gleichzeitig geöffnet werden.

Fehlercodes:

–34 Pfad nicht gefunden
–35 zu viele offene Dateien
–36 Zugriff nicht möglich

Erhält man als Fehlerwert die –36 zurück, so bedeutet dies, daß entweder das Inhaltsverzeichnis voll ist.

Kommando: **Fopen** *Funktionsnummer*: **$3D**

Format: Fopen (name,modus)

char *name;
WORD modus;

Funktion: Öffnen einer Datei

Dateien müssen zum Lesen oder Schreiben geöffnet werden. Dazu dient der Aufruf *Fopen*. Man gibt dabei den Namen der Datei, die man öffnen will, und die Zugriffsart (Modus) an.

Folgende Moduswerte sind bei *Fopen* möglich:

0 – nur Lesen
1 – nur Schreiben
2 – Lesen und Schreiben

Wird die Datei gefunden und war die gewünschte Zugriffsart möglich, so wird das Handle der Datei zurückgegeben, ansonsten wird eine Fehlernummer ausgegeben. Das erhaltene Handle wird später bei allen Lese-, Schreib- und Suchoperationen gebraucht.

Will man z.B. in eine Datei schreiben, deren Attribut auf "nur lesen" gestellt ist, so muß erst das Attribut dieser Datei geändert werden. Im Desktop wird das über das Menü "Zeige Info" bewerkstelligt. Man kann dazu auch die Funktion *Fattrib* dazu verwenden, worauf wir später zurückkommen. Der Aufruf *Fopen* setzt den Dateizeiger an den Anfang der Datei. Die Position des Zeigers kann dann mit *Fseek* verändert werden.

Fehlercodes:

–33 Datei nicht gefunden
–35 zu viele offene Dateien
–36 Zugriff nicht möglich

Kommando: **Fclose** *Funktionsnummer*: **$3E**

Format: Fclose (handle)

```
WORD handle;
```

Funktion: Schließen einer Datei.

Eine Datei, die man geöffnet hat, sollte man nach Beenden der Arbeit auch wieder schließen, sonst könnte es zu einem Datenverlust kommen. Man gibt dabei die Handle-Nummer an, die man beim Öffnen der Datei erhalten hat.

Fehlercodes:

–37 falsche Handle-Nummer

Kommando: **Fread** *Funktionsnummer*: **$3F**

Format: Fread (handle,anzahl,puffer)

```
WORD handle;
LONG anzahl;
char *puffer;
```

Funktion: Lesen einer Datei

Mit der Funktion *Fread* können Daten aus einer Datei gelesen werden. Dazu wird die Handle-Nummer, die man beim Öffnen der Datei erhalten hat, die Anzahl der zu lesenden Bytes und die Adresse eines Puffers, über den die Daten gelesen werden sollen, übergeben. Als Rückgabewert erhalten Sie entweder die Anzahl der fehlerfrei gelesenen Bytes oder eine Fehlernummer.

Wird über das logische Datei-Ende hinaus gelesen, wird der Vorgang abgebrochen, weil nicht mehr Bytes eingelesen werden können, als von der Datei in der FAT belegt sind.

Fehlercodes:

–37 falsche Handle-Nummer

Kommando: **Fwrite** *Funktionsnummer*: **$40**

Format: Fwrite (handle,anzahl,puffer)

```
WORD handle;
LONG anzahl;
char *puffer;
```

Funktion: Schreiben in eine Datei

Die Parameter zum Beschreiben einer Datei sind identisch mit denen beim Lesen einer Datei. Es ist dabei möglich, durch Ändern der Handle-Nummer die Ausgabe einer Datei auch auf einen Drucker oder auf den Bildschirm zu leiten.

Dabei sind folgende Handle-Nummern gestattet:

1 = Konsole-Ausgabe
2 = RS-232-Schnittstelle
3 = Drucker

Die Handle-Nummer 0 bedeutet Konsole, die Nummern 4 und 5 bewirken meistens Fehlermeldungen. Nun wird auch klar, warum man beim Kreieren einer Datei als erste freie Handle-Nummer eine 6 bekommt.

Fehlercodes:

–36 Zugriff nicht möglich
–37 falsche Handle-Nummer

Kommando: **Fdelete** *Funktionsnummer*: **$41**

Format: Fdelete (name)

```
char *name;
```

Funktion: Löschen einer Datei

Dieser Befehl ist mit Vorsicht zu verwenden, da damit Dateien fast unwiderruflich gelöscht werden. Diese Funktion wirkt ohne weitere Angaben nur im aktiven Directory. Befindet sich eine Datei, die man löschen möchte, in einem Folder, so gibt es zwei Möglichkeiten:

- Man übergibt den kompletten Pfadnamen.
- Man ernennt den Folder zum aktuellen Directory (Fkt.DSETPATH).

Fehlercodes:

–33 Datei nicht gefunden
–36 Zugriff nicht möglich

Kommando: **Fseek** *Funktionsnummer*: **$42**

Format: Fseek (anzahl,handle,modus)

LONG anzahl;
WORD handle;
WORD modus;

Funktion: Dateizeiger für direkten Zugriff setzen

Normalerweise wird eine Datei rein sequentiell gelesen, d.h. wenn man eine bestimmte Information sucht, muß man die Datei von Anfang an durchsuchen. Mit dem Befehl Fseek hat man nun die komfortable Möglichkeit, einen Zeiger innerhalb einer Datei an eine bestimmte Position zu setzen.

Es werden 3 Parameter verlangt:

- Anzahl der Bytes, um die der Zeiger bewegt werden soll
- Handle-Nummer der Datei (siehe Fopen)
- Modus

Der Modus kann folgende Werte annehmen:

0 – Suchen vom Anfang der Datei ausgehend

1 – Suchen von der aktuellen Position aus

2 – Suchen vom Dateiende aus rückwärts

Dabei dürfen bei Modus 0 nur positive Werte übergeben werden, bei Modus 1 positive und negative und bei Modus 2 nur negative Werte, da man ja vom Datei-Ende ausgeht und sich nur rückwärts bewegen kann.

Fehlercodes:

–32 falsche Funktion
–37 falsche Handle-Nummer

Kommando: **Fattrib** *Funktionsnummer*: **$43**

Format: Fattrib(pfad,modus,attr)

```
char *pfad;
WORD modus,attr;
```

Funktion: Attribut einer Datei erfragen und ändern

Durch diesen Befehl wird dem Anwender die Möglichkeit gegeben, das Attribut einer Datei zu ändern oder zu ermitteln. Man übergibt den Namen der Datei oder den kompletten Pfadnamen, falls sich die Datei in einem Folder befindet, den Zugriffsmodus (Attribut ermitteln oder ändern) und das neuzusetzende Attribut.

Beim Ermitteln des Attributs einer Datei wird der letzte Parameter nicht verwendet; er muß aber übergeben werden.

In der folgenden Aufstellung werden die möglichen Attributwerte angeführt:

$00 – lesen/schreiben (read/write)

$01 – nur lesen (read only)

$02 – verborgene Datei (hidden file)

$04 – System-Datei (system file)

$08 – Diskettenname (disk label)

$10 – Subdirectory (folder)

$20 – Datei wurde beschrieben und korrekt geschlossen

Die Attribute von Foldern und Disklabels können durch *Fattrib* nicht geändert werden. Man erhält nach dem Aufruf das aktuelle Attribut der Datei zurück, nach dem Ändern des Attributs also auch den neuen Wert oder eine Fehlermeldung.

Fehlercodes:

–33 Datei nicht gefunden
–34 Pfad nicht gefunden

Kommando: **Pexec** *Funktionsnummer*: **$4B**

Format: Pexec (modus,pfad,kommandozeile,umgebung)

```
WORD modus;
char *pfad;
char *kommandozeile;
char *umgebung;
```

Funktion: Laden und/oder Starten einer Datei

Dieser Befehl erlaubt das Laden oder Nachladen von Programmen. Das Programm kann dann gestartet werden.

Pexec verlangt vier Parameter:

Modus: 0 = laden und automatisch starten (load and go)

3 = nur laden (Rückgabe = Basepage) (just load)

4 = Basepage erstellen (create basepage)

5 = starten (just go)

Pfad: Name der Datei oder kompletter Pfadname

Kommandozeile: Die Kommandozeile wird in die Basepage kopiert; vergleichbar mit den Angaben, die man TTP-Programmen übergibt.

Umgebung: Parameter, die nicht über die Kommandozeile übergeben werden können.

Fehlercodes:

–33 Datei nicht gefunden
–39 zuwenig freier Speicher
–66 falsches Ladeformat

Kommando: **Frename** *Funktionsnummer*: **$56**

Format: Frename (null,alt,neu)

```
WORD null;
char *alt;
char *neu;
```

Funktion: Umbenennen einer Datei

Dieser Befehl dient dazu, Dateien umzubenennen. Diese Funktion ist sehr von Nutzen, da es oft notwendig ist, den Dateinamen zu ändern. Der erste Parameter ist eine Null, danach folgt der alte Dateiname und dann der neue Name.

Gibt man nun als neuen Namen zusätzlich den Pfad eines vorhandenen Ordners an (z.B. TEST.FOL\NEUNAME.C), so wird die umzubenennende Datei unter dem neuen Namen in den angegebenen Ordner kopiert.

Es ist nicht möglich, den Namen eines Folders oder eines Disklabels zu ändern. Fehlercodes:

–34 Pfad nicht gefunden
–36 Zugriff nicht möglich

Man erhält den Fehlerwert –36 zurück, wenn die umzubenennende Datei nur gelesen werden darf.

Kommando: **Fdatime** *Funktionsnummer*: **$57**

Format: Fdatime (handle,puffer,modus)

```
WORD handle;
char *puffer;
WORD modus;
```

Funktion: Datum setzen

Die letzte Funktion dieses Kapitels ist Fdatime. Damit ist man in der Lage, das Datum und den Zeitpunkt der Datei-Erstellung zu bestimmen oder zu ändern. Dazu muß die Datei aber mit Fopen oder Fcreate geöffnet worden sein. Die beim Öffnen erhaltene Handle-Nummer wird übergeben. Der zweite Parameter ist ein 4 Byte großer Puffer, der die Daten für die Datei enthält.

Zuletzt wird der Modus angegeben. Modus 0 bedeutet, daß man die Daten der Datei ermittelt; Modus 1 bedeutet, die Daten der Datei werden neu gesetzt. Der Datenpuffer enthält in den ersten beiden Bytes die Zeit; das Datum steht in den letzten zwei Bytes.

Es ergibt sich folgende Codierung:

Zeit: Bits 5 - 10 Minuten
Bits 11 - 15 Stunden

Dabei ist zu beachten, daß die interne Uhr des ATARI in 2-Sekunden-Schritten läuft, da in den ersten 5 Bits nur der Bereich von 0 - 31 dargestellt werden kann.

Datum: Bits 0 - 4 Tag
Bits 5 - 8 Monat
Bits 9 - 15 Jahr - 1980

Der Wertebereich der "Jahrbits" geht bis 119.

Programmbeispiel mit Dateibefehlen

Im folgenden C-Programm wird ein großer Teil der soeben erläuterten Befehle am Beispiel veranschaulicht.

```
/************************************************************/
/* FILE.C                                                   */
/*                                                          */
/* Ein C-Programm zur Veranschaulichung der DOS-Routinen    */
/*                                                          */
/* - Fcreate                                                */
/* - Fopen                                                  */
/* - Fclose                                                 */
/* - Fread                                                  */
/* - Fwrite                                                 */
/* - Fdelete                                                */
/* - Frename                                                */
/* - Fseek                                                  */
/* - Fattrib.                                               */
/************************************************************/


#include "portab.h"

/* Externe DOS-Funktionen                                   */

extern gemdos(), bios(), xbios();
```

```
/* Makrodef. fuer die DOS-Funktionen (LATTICE-C)                    */

#define Fcreate(a,b)                gemdos(0x3c,a,b)
#define Fopen(a,b)                  gemdos(0x3d,a,b)
#define Fclose(a)                   gemdos(0x3e,a)
#define Fread(a,b,c)                gemdos(0x3f,a,b,c)
#define Fwrite(a,b,c)               gemdos(0x40,a,b,c)
#define Fdelete(a)                  gemdos(0x41,a)
#define Fseek(a,b,c)                gemdos(0x42,a,b,c)
#define Frename(a,b,c)              gemdos(0x56,a,b,c)
#define Fattrib(a,b,c)              gemdos(0x43,a,b,c)

#define Bconout(a)                  bios(3,5,a)

#define Cnecin()                    (WORD) gemdos(0x8)

#define LATTICE 1


/* Variablendeklarationen                                           */

WORD  f_handle,mode,attribute,ret_code;
char  name[12],buffer[257];


/* Create_File: oeffnet eine neue Datei auf dem aktuellen           */
/*              Laufwerk.                                           */

create_file()
{
 printf("\n\n Bitte den Namen der Datei eingeben: ");
 scanf("%s",name);

 printf("\n Bitte das Attribut eingeben: ");
 scanf("%h",&attribute);

 f_handle = Fcreate(name,attribute);

 printf(" \n\nDie ID der Datei ist: %d",f_handle);

 ret_code = Fclose(f_handle);
 printf(" \n\nReturncode vom Schliessen ist: %d",ret_code);

 Cnecin();
}


/*Clear_File: loescht eine Datei auf dem aktuellen Laufwerk.*/

clear_file()

{
```

```
 printf(" \n\nBitte den Namen der Datei eingeben: ");
 scanf("%s",name);

 ret_code = Fdelete(name);

 printf("\n\nReturncode vom Loeschen ist: %d \n",ret_code);
 Cnecin();

}

/* Open_File: oeffnet eine vorhandene Datei und schreibt den*/
/*            Puffer hinein                                 */

open_file()
{

 WORD i;

      /* Daten fuer die Datei in den Puffer schreiben       */

 for (i=0; i < 256; i++)
      buffer[i] = i;
 buffer[256] = 0;

 printf(" \n\nBitte den Namen der Datei eingeben: ");
 scanf("%s",name);

 f_handle = Fopen(name,2);
 printf(" \n\nDie ID der Datei ist: %d",f_handle);

 if (f_handle > -1 )
    {

     ret_code = Fwrite(f_handle,255,buffer);
     printf(" \n\nEs wurden %d Bytes geschrieben. ",ret_code);
    }

  ret_code = Fclose(f_handle);
  printf(" \n\nReturncode vom Schliessen ist: %d",ret_code);

 Cnecin();

}

/* Read_File: liest die ersten 65 Bytes aus der Datei.      */

read_file()
{


 WORD i;
 LONG offset;

 printf(" \n\nBitte den Namen der Datei eingeben: ");
 scanf("%s",name);
```

```
 f_handle = Fopen(name,2);
 printf(" \n\nDie ID der Datei ist: %d",f_handle);
 if (f_handle > -1)

  {
   printf(" \n\nAb welchem Byte soll gelesen werden? ");
   scanf("%d",&offset);

   Fseek(offset,f_handle,0);

   ret_code = Fread(f_handle,40,buffer);
   printf(" \n\nEs wurden %d Bytes gelesen. \n\n",ret_code);

   for (i=0; i < 40; i++)
       Bconout(buffer[i]);
   ret_code = Fclose(f_handle);
   printf("\n\nReturncode vom Schliessen ist: %d",ret_code);
  }
 Cnecin();
}


rename()
{
 char alt[12],neu[12];
 WORD hd;

 printf(" \n\nBitte den alten Namen der Datei eingeben: ");
 scanf("%s",alt);

 printf(" \n\nBitte den neuen Namen der Datei eingeben: ");
 scanf("%s",neu);

 hd = Frename(0,alt,neu);
 printf(" \n\nDer Returncode ist: %d",hd);
 Cnecin();
}

change_attr()
{

 WORD mode,attr,hd;

 printf(" \n\nBitte den Namen der Datei eingeben: ");
 scanf("%s",name);

 printf(" \n\nAttribut ermitteln oder setzen (e,s)? ");
 mode = 0;

  if (Cnecin() == 's')
     {
     mode = 1;
     printf(" \n\nAttribut eingeben: ");
     scanf("%h",&attr);
    }
```

```
 hd=Fattrib(name,mode,attr);

 printf(" \n\nDas Attribut ist: %d",hd);
 Cnecin();

}

main()
{

/* Endlosschleife, bis von der Tastatur 0 gelesen wird      */

      while (TRUE)
      {
 /* Bildschirm loeschen und Menue anzeigen                  */

    printf("%cE\n\n",0x1b);
    printf(" - 1 - Datei erstellen\n\n");
    printf(" - 2 - Datei loeschen \n\n");
    printf(" - 3 - Datei oeffnen \n\n");
    printf(" - 4 - Datei lesen \n\n");
    printf(" - 5 - Datei umbenennen \n\n");
    printf(" - 6 - Attribut aendern \n\n\n");
    printf(" - 0 - Zurueck zum DESKTOP \n\n");

 /* Tastatur abfragen und in den Menuepunkt verzweigen      */

 switch (Cnecin())
   {
     case '0': exit(0);
     case '1': create_file();
           break;
     case '2': clear_file();
           break;
     case '3': open_file();
         break;
     case '4': read_file();
         break;
     case '5': rename();
         break;
     case '6': change_attr();
    }
 }
}
```

Folder-Befehle

Die Folder-Befehle setzen sich aus folgenden Funktionen zusammen:

- DCREATE $39
- DDELETE $3A
- DSETPATH $3B
- DGETPATH $47

Das Erstellen und Verwalten von Foldern oder Ordnern ist eine der komfortabelsten Funktionen des Betriebssystems. Damit hat man die Möglichkeit, das Disketten-Directory sehr übersichtlich zu organisieren. Leider kostet jeder Ordner auch Platz auf der Diskette, aber das nimmt man der besseren Übersicht in der Directory wegen gerne in Kauf.

Kommando: **Dcreate** *Funktionsnummer*: **$39**

Format: Dcreate (pfad)

char *pfad;

Funktion: Vereinbarung eines Folders

Mit diesem Befehl können Ordner (Folder) erstellt werden. Dazu wird der Name des Ordners, bestehend aus 8 Zeichen und 3 Zeichen Extension, übergeben. Es ist auch möglich, den Pfadnamen eines schon existierenden Ordners anzugeben. Dann wird der neue Ordner innerhalb des anderen erzeugt.

Fehlercodes:

–34 Pfad nicht gefunden
–36 Zugriff nicht möglich

Kommando: **Ddelete** *Funktionsnummer*: **$3A**

Format: Ddelete (pfad)

char *pfad;

Funktion: Löschen eines Folders

Diese Funktion erlaubt es, ein leeres Subdirectory zu löschen. Enthält der Ordner Dateien, so wird eine Fehlermeldung zurückgegeben. Um diesen Ord-

ner dennoch zu löschen, muß man erst alle Dateien in diesem Folder löschen. Dann erst kann man diesen Ordner löschen.

Die Übergabe des Namens erfolgt wie bei Dcreate.

Fehlercodes:

-34 Pfad nicht gefunden
-36 Zugriff nicht möglich
-65 interner Fehler

Kommando: **Dsetpath** *Funktionsnummer*: **$3B**

Format: Dsetpath (pfad)

char *pfad;

Funktion: Folder zum Directory ernennen

Damit kann ein Folder zum aktuellen Directory ernannt werden. Es ist somit nicht nötig, bei Zugriff auf eine Datei innerhalb eines Ordners, immer den kompletten Pfadnamen zu übergeben. Man ernennt den Ordner zum aktuellen Directory und kann dann direkt auf die Dateien innerhalb des Ordners zugreifen. Um wieder in das Haupt-Directory zu kommen, genügt es, den Backslash (\) einzugeben.

Fehlercodes:

-34 Pfad nicht gefunden

Kommando: **Dgetpath** *Funktionsnummer*: **$47**

Format: Dgetpath (pfadpuffer,drive)

char *pfadpuffer;
WORD drive;

Funktion: Directory ermitteln

Diese Funktion ermittelt das derzeitig aktuelle Directory. Dazu wird ein Zeiger auf einen 64-Byte-Puffer übergeben, in dem dann der komplette Pfadname

steht (wie immer durch ein Nullbyte abgeschlossen). Dazu muß die Nummer des gewünschten Laufwerks angegeben werden:

0 – aktuelles Laufwerk
1 – Laufwerk A
2 – Laufwerk B
.
.
.

Ist zum Zeitpunkt des Aufrufs das Haupt-Directory aktiv, so wird nur das Nullbyte zurückgegeben.

Fehlercodes:

–46 unbekanntes Laufwerk

Programmbeispiel mit Folder-Befehlen

Im folgenden C-Programm werden die soeben erläuterten Befehle am Beispiel veranschaulicht:

```
/*************************************************************/
/*                                                           */
/* FOLDER.C                                                  */
/*                                                           */
/* Ein C-Programm zur Veranschaulichung der DOS-Routinen     */
/*                                                           */
/* - Dcreate                                                 */
/* - Ddelete                                                 */
/* - Dsetpath                                                */
/* - Dgetpath                                                */
/*                                                           */
/*************************************************************/

#include "portab.h"


/* Externe DOS-Funktionen                                    */

extern gemdos(), bios(), xbios();

/* Makrodef. fuer die DOS-Funktionen (LATTICE-C)             */

#define Cnecin()                (WORD) gemdos(0x8)
#define Dcreate(a)              gemdos(0x39,a)
```

```
#define Ddelete(a)              gemdos(0x3a,a)
#define Dsetpath(a)             gemdos(0x3b,a)
#define Dgetpath(a,b)           gemdos(0x47,a,b)

#define LATTICE 1


/* Variablendeklarationen                                          */

WORD  f_handle,ret_code;
char  name[14],buffer[50];


create_folder()
{

 printf("\n\n Bitte den Folder-Namen eingeben: ");
 scanf("%s",name);

 f_handle = Dcreate(name);
 printf("\n\n Die Folder-ID ist: %d",f_handle);

 Cnecin();

}

delete_folder()
{

 printf("\n\n Bitte den Folder-Namen eingeben: ");
 scanf("%s",name);

 ret_code = Ddelete(name);
 printf("\n\n Der Return-Code vom Loeschen ist: %d",ret_code);

 Cnecin();


}


change_path()
{
 printf("\n\n Bitte den Folder-Namen eingeben: ");
 scanf("%s",name);

 ret_code = Dsetpath(name);
 printf("\n\n Der Returncode von SETPATH ist: %d",ret_code);

 Cnecin();

}

show_path()
{
```

```
 Dgetpath(buffer,0);
 printf("\n\n Der momentane Pfadname ist: %s",buffer);

 Cnecin();
}


main()
{


 /* Endlosschleife, bis von der Tastatur 0 gelesen wird      */

 while (TRUE)
   {


/* Bildschirm loeschen und Menue anzeigen                    */

   printf("%cE\n\n",0x1b);
   printf(" FOLDER.PRG\n\n");
   printf(" Ein Beispielprogramm aus dem FLOPPY-ARBEITS");
   printf("BUCH\n\n\n");
   printf(" - 1 - Folder erstellen \n\n");
   printf(" - 2 - Folder loeschen \n\n");
   printf(" - 3 - Pfadnamen setzen \n\n");
   printf(" - 4 - Pfadnamen holen \n\n\n");
   printf(" - 0 - Zurueck zum DESKTOP \n\n");


/* Tastatur abfragen und in den Menuepunkt verzweigen        */

      switch (Cnecin())
      {

   case '0': exit(0);

   case '1': create_folder();
         break;

   case '2': delete_folder();
         break;

   case '3': change_path();
         break;

   case '4': show_path();

   }

 }

}
```

Directory-Befehle

Die Directory-Befehle setzen sich aus folgenden Funktionen zusammen:

- FSETDTA $1A
- FGETDTA $2F
- DFREE $36
- FSFIRST $4E
- FSNEXT $4F

Kommando: **Fsetdta** *Funktionsnummer*: **$1A**

Format: Fsetdta (ptr)

LONG ptr;

Funktion: DTA setzen

Die Disk Transfer Address, kurz auch DTA genannt, ist ein 44-Byte-Puffer, der hauptsächlich bei Directory-Operationen Verwendung findet. Mit dem Befehl kann diese DTA gesetzt werden. Eine genaue Beschreibung des Pufferinhalts finden Sie unter der Funktionsbeschreibung des Befehls Fsfirst.

Kommando: **Fgetdta** *Funktionsnummer*: **$2F**

Format: Fgetdta()

Funktion: Adresse der DTA ermitteln

Als Ergebnis dieses Aufrufs erhält man die Adresse des DTA-Puffers zurück.

Kommando: **Dfree** *Funktionsnummer*: **$36**

Format: Dfree (puffer,drive)

LONG puffer;
WORD drive;

Funktion: Platz auf der Diskette prüfen

Mit Dfree stellen Sie fest, wieviel Platz auf der Diskette belegt ist und wieviel noch frei ist.

Dabei müssen ein 16 Byte großer Puffer und die Nummer des gewünschten Laufwerks angegeben werden.

0 – aktives Laufwerk
1 – Laufwerk A
2 – Laufwerk B
.
.
.

Der Puffer zeigt auf eine Struktur, die folgendermaßen aussieht:

```
LONG b_frei;
LONG b_total;
LONG b_sektorgröße;
LONG b_clustergröße;
```

b_frei: Anzahl der freien "Allocation Units" (Cluster). Eine Datei auf der Diskette, auch wenn sie nur ein paar Bytes groß sein sollte, benötigt mindestens ein Cluster (also zwei Sektoren).

b_total: Gesamtanzahl der Cluster auf einer Diskette. Bei einer einseitigen Diskette beträgt dieser Wert 351, bei einer doppelseitigen Diskette 711.

b_sektorgröße: Größe eines Sektors (normalerweise 512 Bytes).

b_clustergröße: Anzahl der zu einem Cluster gehörenden Sektoren (normalerweise zwei Sektoren).

Fehlercodes:

–46 unbekanntes Laufwerk

Programmbeispiel: Dfree

```
/*********************************************************/
/*                                                       */
/* SPACE.C Ein Beispielprogramm aus dem Floppy -         */
/* Arbeitsbuch                                           */
/*                                                       */
/* Dieses Programm gibt den freien Speicherplatz auf der */
/* Diskette aus.                                         */
/*                                                       */
/*********************************************************/
```

```
#include"osbind.h"
#include"stdio.h"

#define DIGITALC 1

main()

 {


long pbuf;
long free_space,used_space,max_space;
char c;


struct buffer

 {

   long b_free;
   long b_total;
   long b_secsize;
   long b_clsize;
 } buf;



pbuf = &buf;
label:

printf(" Bitte Diskette einlegen.....\n");
Cconin();


Dfree(pbuf,0);
free_space = buf.b_free * buf.b_clsize * buf.b_secsize ;
max_space = buf.b_total * buf.b_clsize * buf.b_secsize ;
used_space = max_space - free_space;


printf(" Maximaler Speicherplatz : %ld\n",max_space);
printf(" Freier Speicherplatz : %ld\n",free_space);
printf(" Belegter Speicherplatz : %ld\n",used_space);


printf(" Noch eine Diskette ? J/N \n");
c = getchar();
if (c== 'j')
   goto label;
printf(" Bitte Taste druecken.....\n");
Cconin();
 }
```

Kommando: **Fsfirst** *Funktionsnummer*: **$4E**

Format: Fsfirst (name,attr)

char *name;
WORD attr;

Funktion: Datei auf Diskette suchen

Dieser Befehl ermöglicht es, herauszufinden, ob eine bestimmte Datei in der Directory enthalten ist. Man kann auch nach mehreren Dateien suchen oder sich auch das ganze Directory ausgeben lassen. Man muß zuerst den DTA-Puffer *(Fsetdta)* einrichten, damit man die kompletten Informationen über eine Datei erhält.

Dieser Puffer baut sich folgendermaßen auf:

Byte	0 - 20	für GEMDOS reserviert
Byte	21	Datei-Attribut
Byte	22 - 23	Uhrzeit der Erstellung
Byte	24 - 25	Datum der Erstellung
Byte	26 - 29	Dateigröße in Bytes
Byte	30 - 43	Name und Extension der Datei

Bei Aufruf des Befehls übergibt man den Dateinamen und das Attribut der Datei, nach der gesucht werden soll. Bei der Angabe des Dateinamen ist es nicht nötig, immer den kompletten Namen anzugeben. Buchstaben können auch durch ein Fragezeichen (?) ersetzt werden, der Name oder die Extension durch einen Stern (*).

Man kann auch den gesamten Dateinamen durch *.* ersetzen. Es wird immer die erste Datei ausgegeben, die nach den Suchkriterien gefunden wurde. Als Attribute können die Werte eingesetzt werden, die schon bei *Fattrib* behandelt wurden. Man kann aber auch den Wert 255 als Attribut angeben. Dann wird das Attribut der Datei bei der Suche nicht mehr beachtet.

Beispiele:

Fsfirst(TEST.PRG,0)	sucht nach TEST.PRG (read/write).
Fsfirst(*.C,0)	sucht das erste read/write-Programm mit der Extension .C .
Fsfirst(??ST.PAS,16)	sucht den Ordner ??ST.PAS, wobei die ersten beiden Buchstaben beliebig sind.
Fsfirst(*.*,8)	sucht den Diskettennamen.
Fsfirst(*.*,255)	sucht die erste Datei.

Wird eine Datei gefunden, werden die Daten in den DTA-Puffer übertragen.

Fehlercodes:
–33 Datei nicht gefunden
–49 keine weiteren Dateien

Kommando: **Fsnext** *Funktionsnummer*: **$4F**

Format: Fsnext()

Funktion: Weitere Dateien suchen

Sucht man nun nicht nur eine ganz bestimmte Datei, sondern mehrere oder will man gar das Directory einlesen, benötigt man die Funktion *Fsnext*. Natürlich muß vorher ein *Fsfirst*-Aufruf erfolgt sein.

Parameter benötigt diese Funktion nicht. Zum Einlesen des Directories genügt es, nach einem einmaligen *Fsfirst*-Aufruf (*.*,255) solange den Aufruf *Fsnext* zu verwenden, bis die zurückgegebene Handle-Nummer ungleich 0 ist.

Dann wurde keine Datei mehr gefunden.

Fehlercodes:

–49 keine weiteren Dateien

Programmbeispiel: Catalog.C

```
/*********************************************************/
/*                                                       */
/*      Dieses Programm ist ein Beispiel zum Einlesen des */
/*   Directories. Folgende Routinen werden benutzt:      */
/*                                                       */
/*               Fsfirst, Fsnext und Setdta              */
/*                                                       */
/*********************************************************/

#include "osbind.h"
#define WORD  short
#define LONG  int
#define LATTICE 1
```

```
/* Aufbau des DTA-Puffers                    */

struct  buffer  {  char dummy[21];
                   char attr;
                   WORD zeit;
                   WORD datum;
                   LONG laenge;
                   char name[14];
                } dta;
/*   Diese Routine maskiert die Datumbits aus */
void
data(i)
WORD i;
{
        WORD j;
        j = i & 0x001f;
        printf("  %2u.",j);
        j = (i & 0x01e0) >> 5;
        printf("%2u.",j);
        j = ((i & 0xfe00) >> 9)+80;
        printf("%2u  ",j);
}
/*   Diese Routine maskiert die Zeitbits aus */
void
time(i)
WORD i;
{
        WORD j;
        j = (i & 0xf800) >> 11;
        printf("%02u:",j);
        j = (i & 0x07e0) >> 5;
        printf("%02u\n",j);
}
/*  Namen in der DTA löschen      */
void
clear_name()
{
        WORD i;
        for (i=0; i < 14; i++) dta.name[i] = '\0';
}
main()
{
        WORD    handle,a;
        char    *name = "A:\\*.*"; /* Directory von Drive A */
        clear_name();
        Fsetdta(dta);
        handle = Fsfirst(name,255); /* 255 = Alle Dateien */
        a = 0;
        if (handle == 0)
        do
        {

                printf("%-15.13s",dta.name);
                printf("%3d",dta.attr);
                printf("  %6u",dta.laenge);
                data(dta.datum);
```

```
                time(dta.zeit);
                clear_name();
                handle = Fsnext();     /* nächste Datei */
                if (a++ == 23) Cnecin();
        }
        while (handle == 0);
        Cnecin();
}
```

Drive-Befehle

Es gibt nur zwei Drive-Befehle:

Kommando: **Dsetdrv** *Funktionsnummer*: **$0E**

Format: Dsetdrv(drive)

WORD drive;

Funktion: Aktuelles Laufwerk bestimmen

Mit diesem Befehl kann ein Laufwerk zum aktuellen Laufwerk ernannt werden. Dem Aufruf wird ein Parameter übergeben, der die Laufwerksbezeichnung enthält, 0 für Laufwerk A, 1 für Laufwerk B. Nach dem Aufruf erhält man die Nummer des vor dem Aufruf aktiven Laufwerks.

Kommando: **Dgetdrv** *Funktionsnummer*: **$19**

Format: Dgetdrv()

Funktion: Aktuelles Laufwerk ermitteln

Die Funktion Dgetdrv ermittelt die derzeitig aktuelle Floppy. Man erhält die Nummer des aktiven Laufwerks.

Programmbeispiel:

```
/*********************************************************/
/*                        DRIVE.C                        */
/*         Dieses Programm dient zur Veranschaulichung der  */
/*           DRIVE-Routinen des ATARI-Betriebssystems TOS.  */
/*********************************************************/

#include "stdio.h"
#include "osbind.h"
```

```
/*       Makrodefinitionen fuer die GEMDOS-Aufrufe          */

#define ESC         0x1b

#define DIGITALC 1

/*                    Hauptprogramm                         */

main()
{

        WORD    new_drive, old_drive;
        char    number;

/*   Bildschirm loeschen und Kopfzeile ausgeben             */

printf("%cE",ESC);
printf("\n              -   DRIVE.PRG  -\n\n");
printf("Ein Beispielprogramm aus dem FLOPPY-ARBEITSBUCH\n\n");

/*   Aktuelle Laufwerknummer holen und ausgeben             */

old_drive = Dgetdrv();
printf("\n  Aktuelles Laufwerk:     %c",old_drive + 0x41);
Cnecin();

/*  Neues Laufwerk eingeben und setzen                      */

        printf("\n\n  Laufwerk eingeben (A..D): ");
        number = Cnecin();
        printf("  %c\n",number);

/* Wenn number ein Kleinbuchstabe ist, 0x20 subtrahieren    */

        if (number > 0x60)
          number -= 0x20;

/* Wenn number < 'A' oder > 'D' ist, dann number            */
/* auf 'A' setzen                                          */

        if ((number < 0x41) || (number > 0x44))
          number = 0x41;

/*   Neues Laufwerk (0..3) setzen                           */

        Dsetdrv(number - 0x41);
```

```
/*   Aktuelle Laufwerknummer holen und ausgeben                    */

   new_drive = Dgetdrv();
   printf("\n\n Aktuelles Laufwerk:     %c",new_drive + 0x41);
   Cnecin();


/*  Laufwerknummer in den Anfangszustand zuruecksetzen             */

        Dsetdrv(old_drive);
}
```

Die Fehlermeldungen des GEMDOS

Als Ergebnis des Aufrufs einer GEMDOS-Funktion erhält man einen Wert zurück, der Aufschluß darüber gibt, ob während derAusführung der Funktion ein Fehler aufgetreten ist.

Ist dieser Wert 0, so ist kein Fehler aufgetreten. Die Funktion wurde korrekt ausgeführt. Erhält man einen negativen Wert zurück, so ist ein Fehler aufgetreten.

Diese haben folgende Bedeutung:

–32 ungültige Funktionsnummer

–33 Datei nicht gefunden

–34 Pfadname nicht gefunden

–35 zu viele offene Dateien

–36 Zugriff nicht möglich

–37 ungültige Handle-Nummer

–39 nicht genügend Speicher

–46 ungültige Laufwerkbezeichnung

–49 keine weiteren Dateien

–65 interner Fehler

–66 falsches Ladeformat

Einschränkungen des GEMDOS

Bei einigen Funktionen des GEMDOS sind folgende Einschränkungen zu beachten:

Frename: Ordner und der Diskettenname können nicht umbenannt werden.

Fattrib: Das Attribut von Ordnern und dem Diskettennamen kann ebenfalls nicht geändert werden.

Dcreate: Es können maximal 10 Ordner ineinander verschachtelt werden. Auf einer Diskette können maximal 32 Ordner erstellt werden, ohne daß bei weiterer Erstellung von Ordnern total unsinnige Fehlermeldungen erscheinen oder mit Diskettenfehlern zu rechnen ist.

GEMDOS-Systemaufrufe

Es folgt nun eine Zusammenfassung der GEMDOS-Aufrufe. Dabei werden die hexadezimalen Werte, die Funktion, Eingabeparameter und Rückgabewerte dargestellt:

Hex	Funktion	Eingabe	Rückgabe
E	Dsetdrv	drive	
19	Dgetdrv		Drive
1A	Fsetdta	ptr	
2F	Fgetdta		DTA
36	Dfree	* puffer drive	freie Bytes Anz. Cluster Bytes/Sektor Sektor/Cluster
39	Dcreate	* pfad	Fehler
3A	Ddelete	* pfad	Fehler
3B	Dsetpath	* pfad	Fehler
3C	Fcreate	* pfad attribut	Fehler
3D	Fopen	* pfad modus	Fehler
3E	Fclose	handle	Fehler

Hex	Funktion	Eingabe	Rückgabe
3F	Fread	handle anzahl * puffer	Fehler
40	Fwrite	handle anzahl * puffer	Fehler
41	Fdelete	* pfad	Fehler
42	Fseek	anzahl handle modus	Fehler
43	Fattrib	* pfad modus attribut	Fehler
47	Dgetpath	* pfadpuffer drive	Pfadname
4B	Pexec	modus * pfad kommandozeile umgebung	Adr. Basepage Fehler
4E	Fsfirst	* pfad attribut	Datei Fehler
4F	Fsnext		Datei Fehler
56	Frename	* pfad (alt) * pfad (neu)	Fehler
57	Fdatime	handle * puffer modus	Datum Zeit Fehler

Dabei bedeuten:

*	–	Zeiger auf
DTA	–	Disk-Transfer-Adresse
handle	–	Datei-Handle
pfad	–	Name oder Pfadname
ptr	–	Pointer

Die BIOS-Routinen

Die Schnittstelle zwischen GEMDOS und der Hardware des ATARI ist das BIOS (Basic Input/Output System).

Das BIOS kümmert sich um grundlegende Ein- und Ausgabe-Funktionen, z.B. die Tastatureingabe, Bildschirmausgabe, RS-232-Schnittstelle, Druckerausgabe sowie Ein- und Ausgabe von und auf Disketten.

Die BIOS-Funktionen benutzen bei einem Aufruf die Register A0-A2 und D0-D2. Aufgerufen werden sie durch einen TRAP #13.

Folgende Funktionen des BIOS werden für Diskettenoperationen benutzt:

- Rwabs 4
- *getbpb 7
- Mediach 9
- Drvmap 10

Funktion: **Rwabs** *Funktionsnummer*: **4**

Format: LONG Rwabs (rwflag,puffer,anzahl,start,device)

WORD rwflag;
LONG puffer;
WORD anzahl,start,device;

Funktion: Sektoren lesen oder schreiben

Mit diesem Befehl können Sektoren von der Diskette gelesen und geschrieben werden.

Die Parameter haben folgende Bedeutung:

rwflag: 0 Sektoren lesen

1 Sektoren schreiben
2 Sektoren lesen, ignoriere Diskettenwechsel
3 Sektoren schreiben, ignoriere Diskettenwechsel

puffer: ist die Adresse eines Puffers, über den die Daten gelesen werden sollen oder aus dem die Daten auf Diskette geschrieben werden sollen. Dabei ist zu beachten, daß der Puffer an einer geraden

Adresse (in Assembler) beginnen sollte, da sonst die Übertragung der Daten sehr langsam vonstatten geht.

anzahl: Anzahl der Sektoren, die gelesen bzw. geschrieben werden sollen.

start: gibt an, ab welchem logischen Sektor dabei begonnen wird.

device: gibt an, welches Laufwerk benutzt wird.

0 – Laufwerk A
1 – Laufwerk B
...

Wie bei GEMDOS- werden auch bei BIOS-Aufrufen Fehlercodes zurückgegeben. Ist dieser Wert 0, so ist der Aufruf korrekt abgearbeitet worden; ist er negativ, so ist ein Fehler aufgetreten.

Eine Liste der Fehlermeldungen finden Sie im Anhang.

Kommando: ***getbpb** *Funktionsnummer*: 7

Format: LONG getbpb(device)

WORD device;

Funktion: Pointer auf BPB

Dieser Befehl liefert einen Pointer auf den BIOS-Parameter-Block (BPB) des Laufwerks device.

Dabei steht 0 für Laufwerk A und 1 für Laufwerk B. Man erhält die Adresse des BPB zurück oder 0, falls ein Fehler aufgetreten ist.

Der BIOS-Parameter-Block ist folgendermaßen aufgebaut:

int recsiz	Sektorgröße in Bytes
int clsiz	Cluster-Größe in Sektoren
int clsizb	Cluster-Größe in Bytes
int rdlen	Directory-Länge in Sektoren
int fsiz	FAT-Größe in Sektoren
int fatrec	Sektornummer der FAT-Kopie
int datrec	Sektornummer des ersten Daten-Clusters
int numcl	Anzahl der Daten-Cluster auf Diskette
int bflags	diverse Flags

Die Daten des BPB für 80-Track-SS- und 80-Track-DS-Laufwerke sind:

Parameter	80 Track SS	80 Track DS
recsiz	512	512
clsiz	2	2
clsizb	1024	1024
rdlen	7	7
fsiz	5	5
fatrec	6	6
datrec	18	18
numcl	351	711

Kommando: **Mediach** *Funktionsnummer*: **9**

Format: LONG Mediach (device)

Word device;

Funktion: Diskettenwechsel ermitteln

Dieser Befehl ermittelt, ob zwischenzeitlich eine Diskette gewechselt wurde. Dazu übergibt man die Laufwerknummer (0 für Laufwerk A, 1 für Laufwerk B) und erhält dann einen von drei möglichen Werten als Ergebnis zurück:

- 0 Diskette wurde definitiv nicht gewechselt
- 1 Diskette kann gewechselt worden sein
- 2 Diskette wurde definitiv gewechselt

Kommando: **Drvmap** *Funktionsnummer*: *10*

Format: LONG drvmap()

Funktion: Bitvektor liefern

Dieser Befehl liefert einen Bitvektor, der die angeschlossenen Laufwerke enthält. Die Bitnummer n ist dabei gesetzt, wenn das Laufwerk n angeschlossen ist (wie immer ist 0 = Laufwerk A usw.).

BIOS-Systemaufrufe

Dies ist eine Zusammenfassung der BIOS-Aufrufe. Es werden die dezimalen Werte, die Funktion, Eingabeparameter und Rückgabewerte dargestellt:

Dez	Funktion	Eingabe	Rückgabe
4	Rwabs	rwflag puffer anzahl start device	Daten Fehler
7	getbpb	device	Adr. BPB
9	Mediach	device	gewechselt?
10	Drvmap		Laufwerke

Dabei bedeuten:

Adr – Adresse
BPB – BIOS-Parameter-Block

Programmbeispiel mit BIOS-Routinen

Im folgenden C-Programm werden die soeben erläuterten BIOS-Befehle am Beispiel veranschaulicht:

```
/************************************************************/
/*                                                          */
/* BIOS.C - Ein Beispielprogramm aus dem Floppy-Arbeitsbuch */
/*                                                          */
/*In diesem Programm werden folgende BIOS-Routinen benutzt: */
/*                                                          */
/*                    - Rwabs,                              */
/*                    - Drvmap                              */
/*                    - getbpb                              */
/*                                                          */
/************************************************************/


extern long bios();
extern long gemdos();

#define Rwabs(a,b,c,d,e) bios(4,a,b,c,d,e)
#define getbpb(a)        ((long *) bios(7,a))
#define Mediach(a)       bios(9,a)
```

```
#define Drvmap()          bios(10)
#define WORD              short

/* Dieses MAKRO wartet auf die Eingabe der Tastatur          */

#define Cnecin()         gemdos(8)
#define LATTICE 1

struct BPB
{
        WORD     recsize;
        WORD     clsize;
        WORD     clsizeb;
        WORD     rdlen;
        WORD     fsize;
        WORD     fatrec;
        WORD     datrec;
        WORD     numcl;
        WORD     bflags;

}  *bp_block;


params()
{
    bp_block = getbpb(0);
    printf("Sektorgroesse in Bytes : %d\n",bp_block->recsize);
    printf("Sektoren je Cluster    : %d\n",bp_block->clsize );
    printf("Bytes je Cluster       : %d\n",bp_block->clsizeb);
    printf("Sektoren der Directory : %d\n",bp_block->rdlen  );
    printf("Sektoren je FAT        : %d\n",bp_block->fsize  );
    printf("Startsektor der FAT    : %d\n",bp_block->fatrec );

    printf("Startsektor der Daten  : %d\n",bp_block->datrec );
    printf("Anzahl der Cluster     : %d\n\n",bp_block->numcl);

}

drive()
{
        WORD     handle,k,i;

        handle = Drvmap();
        k = 0x80;
        printf("\nAngeschlossene Laufwerke: ");

        for(i=7; i>-1; i--)
        {
          if (handle & k)
                printf(" %c",i+65);
          k = k / 2;
        }

        printf("\n\n");

}
```

```
boot()
{

     /*  Einlesen des Bootsektors über die RWABS-Funktion */

     char sektor[512];
     WORD i,j;

     Rwabs(0,sektor,1,0,0);

     for (i=0; i<512; i++)
     {
         if (i % 16 == 0) printf("\n");
         j = sektor[i] & 0x00ff;
         printf(" %02x",j);
     }

}

main()
{
        drive();
        Cnecin();
        params();
        Cnecin();
        boot();
        Cnecin();
}
```

Die XBIOS-Routinen

Außer den BIOS-Funktionen gibt es beim ATARI ST noch die erweiterten BIOS-Funktionen (XBIOS – extended BIOS). Diese werden über TRAP #14 aufgerufen.

Folgende Funktionen des XBIOS befassen sich mit der Floppy:

- Floprd 8
- Flopwr 9
- Flopfmt 10
- Protobt 18
- Flopver 19

Kommando: **Floprd** *Funktionsnummer*: **8**

Format: WORD Floprd (puffer,füller,dev,startsek,track,seite,anzahl)

LONG puffer,füller;
WORD dev,startsek,track,seite,anzahl;

Funktion: Sektoren lesen

Mit diesem Befehl kann man einen oder mehrere Sektoren von der Diskette lesen. Die Parameter haben folgende Bedeutung:

puffer: In diesen Puffer werden die Daten von Diskette gelesen. Der Puffer muß an einer Wortgrenze beginnen und für die zu lesenden Daten groß genug sein (512 Bytes mal Anzahl der Sektoren).

füller: Dummy-Parameter (immer 0).

dev: Laufwerk (0 = Laufwerk A, 1 = Laufwerk B).

startsek: Nummer des ersten zu lesenden Sektors (normalerweise zwischen 1 und 9).

track: Nummer des zu lesenden Tracks.

seite: Gibt an, auf welcher Diskettenseite gelesen werden soll (0=Seite 1, 1 = Seite 2).

anzahl: Anzahl der Sektoren, die gelesen werden sollen.

Als Rückgabewert erhält man entweder 0, wenn kein Fehler aufgetreten ist, oder wie bei GEMDOS eine negative Zahl als Fehlermeldung.

Diese Fehlermeldungen sind am Ende dieses Kapitels aufgelistet.

Kommando: **Flopwr** *Funktionsnummer*: 9

Format: WORD Flopwr (puffer,füller,dev,startsek,track,seite,anzahl)

LONG puffer,füller;
WORD dev,startsek,track,seite,anzahl;

Funktion: Sektoren schreiben

Mit dieser Funktion kann man einen oder mehrere Sektoren auf die Diskette schreiben. Die Parameter haben die gleiche Bedeutung wie bei Floprd.

Kommando: **Flopfmt** Funktionsnummer: **10**

Format: WORD
Flopfmt (puffer,füller,dev,spt,track,seite,intlv,magic,virgin)

LONG puffer,füller;
WORD dev,spt,track,seite,intlv,virgin;
LONG magic;

Funktion: Diskette formatieren

Mit diesem Befehl hat man die Möglichkeit, einzelne Tracks oder auch die ganze Diskette zu formatieren.

Dabei werden folgende Übergabe-Parameter verlangt:

puffer: Puffer, der die kompletten Track-Daten enthält. Bei 9 Sektoren pro Track muß dieser Puffer mindestens 8 KByte groß sein.

füller: Dummy-Parameter (immer 0).

dev: Nummer des Laufwerks (0 = Laufwerk A, 1 = Laufwerk B).

spt: "Sectors per Track"; dies ist die Anzahl der zu formatierenden Sektoren pro Track.

track: Nummer des Tracks.

seite: Diskettenseite (0 oder 1).

intlv: "Interleave"; bestimmt, in welcher Reihenfolge die Sektoren auf die Diskette geschrieben werden.

Beispiel: Bei einem Interleave von 1 lautet die Sektor-Reihenfolge: 1-2-3-4-5-6-7-8-9.

Bei einen Interleave von 2 lautet die Reihenfolge:
1-3-5-7-9-2-4-6-8.

magic: Als sogenannte "magic number" muß die Konstante $87654321 benutzt werden, da sonst die Formatierung abgebrochen wird.

virgin: Dies ist der Wert, der beim Formatiervorgang als Datenbytes auf die Diskette geschrieben wird.

ATARI hat als Standardwert $E5E5 empfohlen. Man sollte es vermeiden, den Wert $F als Hi-Nibble-Wert zu wählen, da die Bytefolgen $F* (* – Joker) vom Floppy-Disk-Controller als Befehle interpretiert werden.

Als Ergebnis des Aufrufs erhält man einen Fehlercode zurück. Beträgt dieser Wert 0, so war die Formatierung erfolgreich. Falls eine negative Zahl zurückgegeben wurde, ist ein Fehler aufgetreten.

Der Wert –16 (Bad Sectors) bedeutet, daß einige Sektoren nicht korrekt formatiert wurden und die Daten nicht richtig zurückgelesen werden konnten. (Die Formatierroutine im TOS liest nämlich nach jedem Sektor, der formatiert wurde, diesen Sektor gleich wieder ein (verify) und überprüft die gelesenen Daten mit denen im Formatierpuffer.)

Falls ein Fehler –16 aufgetreten ist, steht im Puffer eine Liste der "schlechten" Sektoren. Sie können daraufhin noch einmal formatiert werden (natürlich muß der ganze Track formatiert werden, da immer bei Sektor eins angefangen wird) oder in der FAT als "bad" markiert werden.

Kommando: **Protobt** *Funktionsnummer*: **18**

Format: VOID Protobt (puffer,seriennr,disktyp,execflag)

LONG puffer,seriennr;
WORD disktyp,execflag;

Funktion: Image des Bootsektors erzeugen

Mit diesem Befehl wird das "Image" (Bild) eines Bootsektors erzeugt. Der erzeugte Bootsektor steht dann in einem Puffer, den man mit dem Befehl Flopwr auf Track 0, Sektor 1 auf Seite 1 schreibt.

Die Parameter haben dabei folgende Bedeutung:
execflag: Dieses Flag bestimmt, ob der Bootsektor ausführbar ist.

0 nicht ausführbar
1 ausführbar
–1 Bootsektor bleibt, wie er war

disktyp: Gibt den Typ der Diskette an (SS/DS 40T/80T)
0 40 Tracks, single sided (180 kB)
1 40 Tracks, double sided (360 kB)

2 80 Tracks, single sided (360 kB)
3 80 Tracks, double sided (720 kB)
–1 Disktyp wird nicht verändert

seriennr: Ist eine 24-Bit-Seriennummer, die in den Bootsektor geschrieben wird. Ist die Seriennummer größer als 24 Bits ($01000000), so wird sie durch den Zufallsgenerator erzeugt. Ein Wert von –1 bedeutet, wie auch schon bei den zwei ersten Parametern, daß die Seriennummer nicht verändert wird.

puffer: Adresse eines 512-Byte-Puffers, in dem die Daten des Bootsektors stehen.

Der Bootsektor einer Systemdiskette hat folgenden Aufbau:

Adresse		40 Tracks SS	40 Tracks DS	80 Tracks SS	80 Track DS
0 – 1		Branch auf Bootprogramm			
2 – 7		"loader"			
8 – 10		Seriennummer			
11 – 12	BPS	512	512	512	512
13	SPC	1	2	2	2
14 - 15	RES	1	1	1	1
16	NFATS	2	2	2	2
17 – 18	NDIRS	64	112	112	112
19 – 20	NSECTS	360	720	720	1440
21	MEDIA	252	253	248	249
22 – 23	SPF	2	2	5	5
24 – 25	SPT	9	9	9	9
26 – 27	NSIDES	1	2	1	2
28 – 29	NHID	0	0	0	0
510 – 511	CHECKSUM				

Die Abkürzungen haben folgende Bedeutung:

BPS: Bytes pro Sektor. Normalerweise immer 512.
SPC: Sektoren pro Cluster.
RES: Anzahl der reservierten Sektoren am Beginn der Diskette einschließlich des Bootsektors.
NFATS: Anzahl der File Allocation Tables (FATs) auf der Diskette.
NDIRS: Maximale Anzahl der Directory-Einträge.
SEC: Gesamtzahl der Sektoren auf der Diskette.
MEDIA: Media Descriptor Byte; wird vom ST-BIOS nicht benutzt.
SPF: Anzahl der Sektoren in jedem FAT.

SPT: Anzahl der Sektoren pro Track.
NSIDES: Anzahl der Seiten auf der Diskette.
NHID: Anzahl der "versteckten" Sektoren; wird vom ST-BIOS ignoriert.

Kommando: **Flopver** *Funktionsnummer*: **19**

Format: WORD Flopver (puffer,füller,dev,startsec,track,seite,anzahl)

LONG puffer,füller;
WORD dev,startsec,track,seite,anzahl;

Funktion: Sektoren prüfen

Dieser Befehl dient zum Überprüfen eines oder mehrerer Sektoren auf der Diskette. Dabei werden die Sektoren von der Diskette gelesen und mit den Daten in einem Puffer verglichen. Stimmen die gelesen Daten mit denen aus dem Puffer überein, so wird keine Fehlermeldung ausgegeben.

Bei einem Fehler wird eine negative Zahl zurückgegeben, und im Puffer steht eine Liste der fehlerhaften Sektoren.

Die Parameter der Funktion sind die gleichen wie bei der Funktion Floprd.

XBIOS-Systemaufrufe

Dies ist eine Zusammenfassung der XBIOS-Aufrufe. Es werden die dezimalen Werte, die Funktion, Eingabeparameter und Rückgabewerte dargestellt:

Dez	Funktion	Eingabe	Rückgabe
8	Floprd	puffer füller dev startsek track seite anzahl	Daten Fehler

Dez	Funktion	Eingabe	Rückgabe
9	Flopwr	puffer füller dev startsek track seite anzahl	Fehler
10	Flopfmt	puffer füller dev spt track seite intlv magic virgin	Fehler
18	Protobt	puffer seriennr disktyp execflag	Fehler
19	Flopver	puffer füller dev startsek track seite anzahl	Fehler

Programmbeispiel mit XBIOS-Routinen

Im folgenden C-Programm werden die soeben erläuterten XBIOS-Befehle am Beispiel veranschaulicht.

```
/*********************************************************/
/* FORMAT.C-Ein Beispielprogramm aus dem Floppy-Arbeitsbuch */
/*                                                       */
/* Dieses Programm formatiert eine einseitige Diskette mit  */
/* 83 Spuren  und 10 Sektoren je Spur. Anschliessend werden */
/* die ersten 30 Bytes des Bootsektors ausgegeben.          */
/*********************************************************/
```

```
/************************************************************/
/*                                                          */
/*In diesem Programm werden folgende XBIOS-Routinen benutzt:*/
/*                                                          */
/*                    - Floprd                              */
/*                    - Flopwr                              */
/*                    - Flopfmt                             */
/*                    - Protobt                             */
/*                    - Flopver                             */
/*                                                          */
/************************************************************/


extern long xbios();
extern long gemdos();

#define Floprd(a,b,c,d,e,f,g) xbios(8,a,b,c,d,e,f,g)
#define Flopwr(a,b,c,d,e,f,g) xbios(9,a,b,c,d,e,f,g)
#define Flopfmt(a,b,c,d,e,f,g,h,i) xbios(10,a,b,c,d,e,f,g,h,i)
#define Protobt(a,b,c,d) xbios(18,a,b,c,d)
#define Flopver(a,b,c,d,e,f,g) xbios(19,a,b,c,d,e,f,g)

/* Dieses MAKRO wartet auf die Eingabe der Tastatur         */

#define Cnecin() gemdos(8)

#define DIGITALC 1

/*    Global definierter Puffer fuer einen Sektor           */

        char    boot[513];

/*    In dieser Routine wird die Diskette formatiert        */

format()
{

  char    buffer[11000]; /*Genuegend Platz fuer 10 Sektoren */
  int     track, handle;


/*  Zuerst werden die Systemspuren mit NULL formatiert      */

        Flopfmt(buffer,0l,0,10,0,0,1,0x876543211,0);
        Flopfmt(buffer,0l,0,10,1,0,1,0x876543211,0);

/* ... anschliessend die restlichen Spuren mit HEX E5E5     */

        for(track = 2; track < 83; track++)
         Flopfmt(buffer,0l,0,10,track,0,1,0x876543211,0xe5e5);

/*  Der Bootsektor wird nun mit 80 Spuren und 9 Sektoren    */
/*  je Spur erstellt */

        Protobt(boot,-1l,2,-1);
```

```
/*  ... und auf unser Format abgeaendert                        */

        boot[0x13] = 0x34;
        boot[0x14] = 0x03;
        boot[0x18] = 0x0A;

/*   zum Schluss wird er auf die Diskette geschrieben           */

        Flopwr(boot,0l,0,1,0,0,1);

/*   ... und noch einmal ueberprueft.                           */

        handle = Flopver(boot,0l,0,1,0,0,1);
        printf("  Fehlercode: %d",handle);

}


/*  Ueber die Floprd-Routine wird hier der Bootsektor           */
/*  eingelesen und die ersten 30 Bytes ausgegeben werden.       */

read_boot()
{

        int     i;

        printf("\n");
        Floprd(boot,0l,0,1,0,0,1);
        for (i=0; i < 0x1e; i++)
                printf(" %02x",boot[i]);
        printf("\n\n");
}

/* Eine MAIN-Routine darf in keinem C-Programm fehlen ...      */

main()
{
        printf("Insert Disk");
        Cnecin();
        format();
        printf("\nOk\n");
        read_boot();
        Cnecin();
}
```

Die Fehlermeldungen des BIOS und XBIOS

0	kein Fehler
–1	allgemeiner Fehler
–2	Drive not ready
–3	unbekannter Befehl

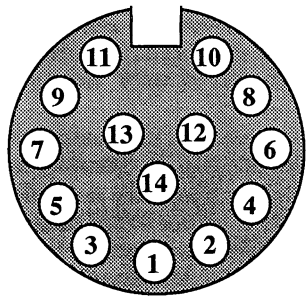
–4 CRC-Error (Checksum Error)
–5 Bad Request (falsches Kommando, Parameterübergabe könnte fehlerhaft sein)
–6 Seek Error (Track nicht gefunden)
–7 Unknown Media (fehlerhafter oder zerstörter Bootsektor)
–8 Sektor nicht gefunden
–9 No Paper
–10 Schreibfehler
–11 Lesefehler
–12 allgemeiner Fehler
–13 Write Protect (Diskette ist schreibgeschützt)
–14 Media Change (Diskette wurde gewechselt)
–15 Unknown Device (nicht bekannt)
–16 Bad Sectors (schlecht formatiert)
–17 Insert Disk (eigentlich eine Aufforderung und kein Fehler)

Kapitel 3

Die Floppy-Disk-Schnittstelle

Für die Floppy-Laufwerke des ST hat sich ATARI einen etwas ungewöhnlichen Anschluß ausgesucht. Ähnlich dem Monitor-Anschluß besteht dieser aus einem runden *"Plug-In"-Stecker* mit 14 Pins.

Die Schnittstelle ist über den DMA-Chip (DMA – Direct Memory Access) mit dem eingebauten Floppy-Disk-Controller (FDC) WD1772 von Western Digital verbunden. Es werden hierbei maximal zwei Laufwerke unterstützt, was in den meisten Fällen auch ausreichend ist. Zudem hat man noch die Möglichkeit, ein Festplattenlaufwerk (Harddisk) anzuschließen. Die Anschlußbuchse der Floppy-Laufwerke hat folgende Belegung:



Die Pin-Belegung des Steckers

1	$\overline{RD}$	Read Data	Das ist der Lese-Eingang für die Laufwerke. Hierüber werden Daten- und Taktimpulse an den FDC geleitet.
2	S0	Side-0 Select	Über diese Leitung wird die Seite Null des Laufwerks angesprochen.

3	GND	Ground	Dies ist der Masseanschluß.
4	$\overline{IP}$	Index Pulse	Über diese Leitung wird der FDC informiert, wenn das Index-Loch der Diskette erkannt wurde.
5	$\overline{D0}$	Drive-0 Select	Über diese beiden low-aktiven Signale
6	$\overline{D1}$	Drive-1 Select	werden die Laufwerke A und B adressiert.
7	GND	Ground	Masseanschluß.
8	$\overline{MO}$	Motor On	Der FDC sendet hierüber das Signal zum Starten des Laufwerkmotors.
9	$\overline{DIRC}$	Direction	Ein High-Pegel an diesem Ausgang veranlaßt die Laufwerkelektronik, den Schreib- und Lesekopf zur Mitte der Diskette zu bewegen; ein Low-Pegel bewegt ihn zum Rand hin.
10	$\overline{STEP}$	Step	Der FDC übermittelt über diesen Ausgang die Impulse für die schrittweise Weiterbewegung des Schreib-/Lesekopfes.
11	$\overline{WD}$	Write Data	Hierüber werden die Takt/Datenbytes auf Diskette geschrieben.
12	$\overline{WG}$	Write Gate	Das ist ein Sicherheitseingang für den Schreibvorgang. Vor jedem Schreibzugriff auf Diskette muß das Signal auf low gesetzt werden.
13	$\overline{TR00}$	Track00	Dieser low-aktive Eingang zeigt dem FDC an, daß der Schreib-/Lesekopf über der Spur Null positioniert ist.
14	$\overline{WPRT}$	Write Protect	Dieses Signal gibt den Zustand des Schreibschutzes auf der Diskette an. Es wird vor jedem Schreibzugriff abgefragt; falls es low ist, wird der Schreibvorgang abgebrochen.

Zum Schluß sind noch die Signalpegel der Anschlüsse aufgeführt:

Anschluß	Pegel
1	TTL - low-aktiv, intern über 1kOhm auf +5V gelegt
2	TTL - high-aktiv, ist nach einem RESET auf high
4	TTL - low-aktiv, intern über 1kOhm auf +5V gelegt
5/6	TTL - low-aktiv, ist nach einem RESET auf high
8-12	TTL - low-aktiv, diese Signale werden invertiert.
13/14	TTL - low-aktiv, intern über 1 kOhm auf +5V gelegt.

Kapitel 4

Direktprogrammierung des FDC

Dieses Kapitel behandelt die direkte Programmierung des FDC ohne Umweg über die DOS-Routinen des ATARI ST. Die hier vorgestellten Beispielprogramme sind in Assembler geschrieben und wurden mit dem AS68 von DIGITAL RESEARCH in den Maschinencode übersetzt. Einige dieser Quelltexte sind auf der beigefügten Diskette zu finden.

Der Aufbau des Floppy-Disk-Controllers

Der ATARI ST ist mit dem Floppy-Disk-Controller WD1770/1772 von WESTERN DIGITAL ausgestattet. Dieser FDC ist kompatibel mit der WD179x-Serie, besitzt jedoch einen digitalen Datenseparator und eine Schreib-Vorkompensiationslogik. Er hat einen Eingang, über den er zwei Aufzeichnungsformate unterscheidet: das FM bzw. das MFM-Verfahren, wobei die Datenübertragungsrate bei dem MFM-Format 250 kBits/sec, bei dem FM-Format 125 kBits/sec beträgt.

Normalerweise benutzt ATARI den WD1772, der Step-Raten von 2, 3, 5 und 6 msec erlaubt, während der WD1770 die Step-Raten der 179x-Serie 6, 12, 20 und 30 msec verarbeitet. Die Größe der Datensektoren ist bei beiden Versionen gleich. Es werden Sektoren mit 128, 256, 512 und 1024 Bytes geschrieben und gelesen.

Der FDC befindet sich in einem 28-Pin-Gehäuse, dessen Anschlüsse nun kurz beschrieben werden:

Pin	Symbol		Funktion
1	$\overline{CS}$	Chip Select	Ein logischer low-Pegel ermöglicht den Zugriff auf die Register des FDC.
2	R$\overline{W}$	Read/$\overline{Write}$	Dieser Anschluß bestimmt die Richtung des Zugriffs. Ein low-Pegel bewirkt einen

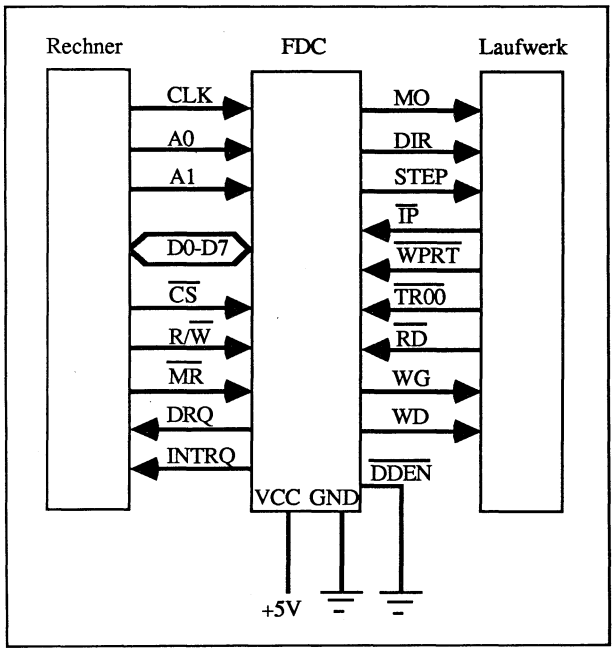
Pin	Symbol		Funktion
			Schreib-, ein high-Pegel einen Lesevorgang.
3, 4	A0 A1	Address 0,1	In Verbindung mit $\overline{RW}$ werden hierüber die FDC-Register ausgewählt:

A1	A0	$\overline{RW} = 1$	$\overline{RW} = 0$
0	0	Status-Reg.	Command-Reg.
0	1	Track-Register	
1	0	Sector-Register	
1	1	Data-Register	

5-12	D0	Data 0-7 -D7	Bidirektionaler 8-Bit-Bus, für die Übertragung von Daten, Kommandos und den Statusinformationen.
13	$\overline{MR}$	Master Reset	Ein logischer low-Pegel bewirkt ein Zurücksetzen des FDC in den Grundzustand und ein Löschen des Statusregisters.
14	GND	Ground	Masseanschluß.
15	VCC	Power Suply	+5V.
16	STEP	Step	Dieser Ausgang liefert einen Impuls für jede Kopfbewegung.
17	DIRC	Direction	Dieser Ausgang ist high, falls die Kopfbewegungen in die Mitte der Diskette, low, wenn die Bewegungen zum Rand der Diskette hin erfolgen.
18	CLK	Clock	An diesem Eingang liegt eine Frequenz von 8 MHz und liefert den Takt für den FDC.
19	$\overline{RD}$	Read Data	Über diesen Eingang werden Takt- und Datenbits gelesen, die anschließend von dem Datenseparator getrennt werden.
20	MO	Motor On	Ein high-Pegel an diesem Ausgang erlaubt die Motoransteuerung bei den Lese-, Schreib- und Step-Operationen.

Pin	Symbol		Funktion
21	WG	Write Gate	Wenn dieses Signal high ist, sind keine Schreibzugriffe auf Diskette gestattet.
22	WD	Write Data	Über diesen Ausgang werden Takt- und Datenbits auf die Disk geschrieben.
23	$\overline{\text{TR00}}$	Track 00	Ist der Schreib-/Lesekopf über der Spur 0 positioniert, ist der Anschluß logisch low.
24	IP	Index Pulse	Dieser Eingang ist low, wenn das Laufwerk das Index-Loch erkannt hat.

Das folgende Schaubild verdeutlicht die Verbindung zwischen dem FDC, dem Laufwerk und dem Rechner:



25	$\overline{\text{WPRT}}$	Write Protect	Ist die Diskette schreibgeschützt, wird dieser Eingang logisch low.

26	$\overline{\text{DDEN}}$	Double Density	Ein low-Pegel an diesem Eingang wählt das MFM-Format, ein high-Pegel das FM-Format aus.
27	DRQ	Data Request	Dieser Ausgang signalisiert, daß das DATA-Register voll (bei einem Lesezugriff) bzw. leer (bei einem Schreibzugriff) ist.
28	INTRQ	Interrupt Req.	Wurde ein Kommando beendet, ist dieser Ausgang auf high-Pegel gesetzt.

Wie die Pins 3 und 4 des FDC bereits vermuten lassen, enthält der Floppy-Disk-Controller fünf Register, die man beschreiben oder auslesen kann. Diese Register sind das DATA-Register, TRACK-Register, SECTOR-Register, COMMAND-Register und das STATUS-Register.

Zusätzlich zu diesen fünf Registern existiert noch das DATA-SHIFT-Register, das beim Lesen die seriellen Daten in Byteform konvertiert und beim Schreiben die Bytes bitweise überträgt. Mit Ausnahme des Command-Registers darf während der Ausführung eines Kommandos auf kein Register des FDC zugegriffen werden, da sonst die Floppy-Operation beendet wird. Sämtliche Register sind 8 Bit breit.

Das DATA-Register, DR

Dieses Register wird bei den Schreib- und Leseoperationen als Zwischenspeicher benutzt. Bei einem Lesekommando schreibt das DATA-SHIFT-Register die gelesenen Daten in das DR, bei einem Schreibkommando holt es sich von hier die Daten.

Wenn ein SEEK-Kommando ausgeführt wird, enthält das DR die zu suchende Spur.

Das TRACK-Register, TR

In diesem Register befindet sich die Position des Schreib-/Lesekopfes. Wenn sich der Kopf um eine Spur zur Mitte der Diskette bewegt, wird das TRACK-Register inkrementiert, bewegt er sich um eine Spur zum Rand der Diskette, wird es dekrementiert.

Falls zwei Diskettenlaufwerke angesteuert werden, muß vor jeder Operation die richtige Position des Schreib-/Lesekopfes in dieses Register geladen werden.

Das SECTOR-Register, SR

In diesem Register steht der gewünschte Sektor. Der Inhalt dieses Registers wird bei den Schreib-/Leseoperationen mit der gelesenen Sektornummer des ID-Feldes verglichen.

Das COMMAND-Register, CR

Dieses Register kann nur beschrieben werden. Es enthält dann das FDC-Kommando.

Das STATUS-Register, STR

Dieses Register kann nur gelesen werden und enthält eine Statusmeldung des FDC. Diese Meldung ist bitweise verschlüsselt:

Bit	Name	Bedeutung
7	MOTOR ON	Dieses Bit zeigt den Zustand des MOTOR-ON-Pins an.
6	WRITE PROTECT	Dieses Bit wird nur bei einer Schreiboperation verwendet. Wenn es gesetzt ist, kann der Schreibzugriff nicht stattfinden.
5	RECORD TYPE/ SPIN_UP	Bei den Typ-I-Kommandos zeigt dieses Bit das Ende der SPIN-UP-Wartezeit an; bei Kommandos des Typs II oder III kennzeichnet das Bit die gelesenen Daten: 0 – Normal DATA MARK 1 – DELETED DATA MARK Die Bedeutung der DELETED DATA MARK wird bei der Beschreibung der FDC-Kommandos genauer erklärt.
4	RECORD NOT FOUND	Dieses Bit ist gesetzt, wenn der FDC die gewünschte Spur, Seite oder den Sektor nicht gefunden hat.

3	CRC ERROR	Falls zu diesem Bit das RECORD NOT FOUND-Bit zusätzlich gesetzt ist, wurde ein Fehler in einem Adressenfeld erkannt, ansonsten ein Fehler im Datenfeld.
2	LOST DATA/ TRACK_00	Bei Kommandos des Typs I zeigt dieses Bit den Zustand des TR00 Pins an. Ansonsten wurde der Inhalt des DATA-Registers von dem DATA-SHIFT-Register überschrieben, bevor es ausgelesen werden konnte.
1	DATA REQUEST/ INDEX	Dieses Bit kennzeichnet bei einem Typ I-Kommando den Zustand des IP-Pins, ansonsten den Zustand des DRQ-Pins.
0	BUSY	Falls zur Zeit ein Kommando bearbeitet wird, ist dieses Bit gesetzt.

Die Programmierung über Direct Memory Access

Die Direktprogrammierung des FDC kann mit Direct Memory Access (DMA) erfolgen oder auch über den direkten Zugriff auf die Systembausteine des ATARI ST. Da sich diese Art der FDC-Programmierung äußerst kompliziert gestaltet, so muß z.B. der Programmierer die Buszuteilung selbst verwalten, wird im folgenden nur die Direktprogrammierung unter DMA behandelt.

Der für den DMA reservierte Speicherbereich beinhaltet alle Register, die für die FDC-Programmierung benötigt werden. Bevor die Datenübertragung von oder zu dem FDC beginnen kann, muß der DMA-Puffer gelöscht und die DMA-Adresse gesetzt werden.

Der DMA-Puffer ist 32 Bytes breit. Alle Daten, die von oder zu dem FDC übertragen werden, werden zuerst in diesen Puffer geschrieben. Stehen schließlich mehr als 15 Bytes in dem DMA-Puffer, beginnt die Datenübertragung. Dabei werden jeweils 16 Bytes gesendet; stehen also 17 Bytes im Puffer werden davon nur 16 übertragen!

Das 17. Byte verbleibt im DMA-Puffer bis dieser wieder halb voll ist. Dieser Puffer wird von dem DMA-Chip selbst verwaltet. Der Programmierer hat auf diesen Puffer keinen Zugriff. Um diesen Puffer zu löschen, wird zuerst der Wert $190 und danach der Wert $90 an die Adresse $FF8606 geschrieben (togglen).

Die Adresse $FF8606 wird auch DMA-Modus-Register, DMA_Mode, genannt. Dieses Register ist für die Steuerung des DMA zuständig. Es wählt über das 4. Bit die Register aus, die über die Adresse $FF8604, FDC_REG ange-

sprochen werden. Ist dieses Bit gesetzt, wird das Sector-Count-Register adressiert; ist es gelöscht, werden die Register des FDC ausgewählt. Die Bits 1 und 2 des DMA-Modus-Registers haben dann für den Zugriff auf das FDC_REG folgende Bedeutung:

Bit 1	2	Bedeutung
0	0	Man erhält beim Lesen den Status des FDC, beim Schreiben wird der Wert in das COMMAND-Register des FDC übertragen.

Bei den anderen Kombinationen wird jeweils ein Register des FDC ausgewählt, in das entweder der Wert des FDC_REG geschrieben oder dessen Inhalt ausgelesen wird.

0	1	Zugriff auf das TRACK-Register
1	0	Zugriff auf das SECTOR-Register
1	1	Zugriff auf das DATA-Register.

Bit	Bedeutung
0	Nicht benutzt.
1, 2	Wenn das Bit 4 gelöscht ist, dann wird hierüber das FDC-Register ausgewählt (s.o.).
3	0 – Der DMA greift auf den FDC zu. 1 – Zugriff auf den HDC (Harddisk-Controller).
4	0 – Über Bit 1 und 2 wird das FDC-Register gewählt. 1 – Das SCR wird angesprochen.
5	Dieses Bit muß gelöscht sein.
6	0 – Der DMA-Chip übernimmt den Datentransfer. 1 – Zugriff erfolgt ohne DMA.
7	0 – Zugriff auf den HDC (Harddisk-Controller). 1 – Der DMA greift auf den FDC zu.
8	0 – Datenrichtung auf Lesen. 1 – Datenrichtung auf Schreiben.
9-15	Nicht benutzt.

Bit-Belegung des DMA-Modus-Registers

Dem DMA muß nun noch mitgeteilt werden, wo er die Daten findet, bzw. wohin er die empfangenen Daten schreiben soll. Dafür besitzt der DMA-Chip

drei 8-Bit-Register, in die die Speicheradresse geschrieben wird. Diese Register stehen im Speicher an den Stellen

$FF8609, $FF860B und $FF860D (DMA_high, DMA_mid, DMA_low)

Die Startadresse der Daten wird in drei Schritten in diese Register geschrieben. Zuerst werden die unteren 8 Bits der Startadresse nach $FF860D, die mittleren 8 Bits nach $FF860B und die oberen 8 Bits nach $FF8609 geschrieben.

Wird der DMA-Puffer gelesen oder beschrieben, so wird anschließend die Adresse in diesen drei Registern neu gesetzt. Die maximale Anzahl der zu übertragenden Bytes wird dem DMA über das Sector-Count-Register mitgeteilt. Der Wert in diesem Register, mit 512 multipliziert, ergibt die maximale Anzahl der Bytes, die übertragen werden. Das Sector-Count-Register SCR wird über das FDC_REG angesprochen, falls das 4. Bit im DMA-Modus-Register gesetzt ist.

Da der ATARI ST zwei Laufwerke verwalten kann, werden noch Register des Soundchips für die FDC-Programmierung benötigt. Der Soundchip besitzt 16 interne Register, von denen das 15. für die Auswahl des Laufwerks und der Diskettenseite zuständig ist.

Um dieses Register ansprechen zu können, wird in das Select-Register des Soundchips, PSG, die Nummer des gewünschten Registers geschrieben. Danach wird an der Adresse $FF8802, PSG_DATA, die Auswahl des Laufwerks und der Seite getroffen. Dafür sind dessen untersten drei Bits relevant:

Bit 0 – Seite 1
Bit 1 – Laufwerk A
Bit 2 – Laufwerk B

Dieses Register ist low-aktiv, daher entstehen die angegebenen Werte, wenn die dazugehörigen Bits gelöscht sind.

Wenn eine Floppy-Operation beendet ist, wird das BUSY-Bit im Statusregister des FDC gelöscht, und ein Interrupt wird erzeugt. Um festzustellen, ob die Operation nun beendet wurde, darf man nicht das BUSY-Bit abfragen, da sonst das ausgeführte Kommando abgebrochen würde. Dadurch kommt ein neuer Baustein des ATARI ST ins Spiel, der Multi Function Periphal Chip, MFP. Dieser ist unter anderem für die Verwaltung der Interrupts zuständig.

Das erste Register des MFP wird General Purpose Input/Output Interrupt Port, GPIP, gennant. Ist das 5. Bit dieses Registers gesetzt, hat der FDC einen Interrupt erzeugt, und das FDC-Kommando wurde beendet.

Die Kommandos des Floppy Disk Controllers

Der FDC versteht elf Kommandos. Diese Kommandos sind in vier Typengruppen unterteilt. Die Kommandoworte werden dem CR übermittelt, wenn das zugehörige Kommando ausgeführt werden soll. Dabei sollte beachtet werden, daß die Ausführung eines Kommandos sofort abbricht, wenn ein neues Kommando in das CR geschrieben wird. Die einzige Ausnahme bildet der "Force Interrupt"-Befehl.

Typ I Kommandos:
- Restore
- Seek
- Step
- Step in
- Step out

Typ II Kommandos:
- Read Sector
- Write Sector

Typ III Kommandos:
- Read Adress
- Read Track
- Write Track

Typ IV Kommando:
- Force Interrupt

Die folgende Tabelle zeigt die Bedeutung der Bits bei den jeweiligen Kommandos:

Restore	0	0	0	0	h	v	r1	r0
Seek	0	0	0	1	h	v	r1	r0
Step	0	0	1	u	h	v	r1	r0
Step in	0	1	0	u	h	v	r1	r0
Step out	0	1	1	u	h	v	r1	r0
Read Sector	1	0	0	m	h	e	0	0
Write Sector	1	0	1	m	h	e	p	a0
Read Address	1	1	0	0	h	e	0	0
Force Interrupt	1	1	0	1	I3	I2	I1	I0
Read Track	1	1	1	0	h	e	0	0
Write Track	1	1	1	1	h	e	p	0

Das Spin-Up-Bit

h = 0 – Spin-Up Sequenz einschalten
h = 1 – Spin-Up Sequenz ausschalten

Ist das h-Bit gesetzt, überprüft der FDC vor der Ausführung der Kommandos (mit Ausnahme des Force Interrupt-Kommandos) den Zustand des Motor-On-

Pins. Falls dieses auf low-Pegel (Motor aus) liegt, aktiviert der FDC den Motor und wartet 6 Umdrehungen, bis der Motor ca. 300 Umdrehungen/min. erreicht hat. Danach führt er das Kommando aus, wartet 10 Umdrehungen und setzt den Motor-On-Pin wieder auf low-Pegel. Wenn das h-Bit gelöscht ist, legt der FDC keine Wartezeit ein und läßt auch das Motor-On-Pin im high-Zustand.

Das Verify-Bit

v = 0 – ohne Verify
v = 1 – mit Verify

Bei den Kommandos der Gruppe 1 kann eine Überprüfung der Position des Schreib-/Lesekopfs erfolgen. Ist das v-Bit gesetzt, vergleicht der FDC die aus dem ersten aufgetretenen ID-Feld gelesene Spurnummer mit dem Inhalt des TRACK-Registers. Wenn beide Werte übereinstimmen und auch die Prüfsumme des ID-Feldes keinen Fehler aufweist, ist die Überprüfung beendet, und das Kommando wird ohne Fehler beendet. Stimmen die beiden Werte überein, die Prüfsumme aber nicht, wird das CRC-Fehlerbit im STATUS-Register gesetzt, und das nächste ID-Feld wird für die Verify-Operation benutzt. Findet der FDC innerhalb von 5 Umdrehungen zusätzlich zur Übereinstimmung keine korrekte Prüfsumme, so wird das Kommando mit einem Seek-Error abgebrochen.

Die Step-Rate

r1	r0	
0	0	– 6 ms
0	1	– 12 ms
1	0	– 2 ms
1	1	– 3 ms

Diese beiden Bits geben bei den Kommandos des ersten Typs die Zeit an, die der FDC zwischen zwei Step-Impulsen wartet. Die Step-Rate des TOS liegt bei 3 ms. Diese STEP-Raten sind die offiziell von ATARI dokumentierten und entsprechen nicht denen von Western Digital.

Das Update-Bit

u = 0 – ohne Update
u = 1 – mit Update

Dieses Bit tritt nur bei den Step-Kommandos in Erscheinung. Ist es gesetzt, so wird bei einem Step zur Mitte der Diskette das TRACK-Register inkrementiert, bei einem Step zum Rand der Diskette dekrementiert.

Das Multiple-Bit

m = 0 – einen Sektor lesen/schreiben
m = 1 – mehrere Sektoren lesen/schreiben

Dieses Bit bestimmt bei den Kommandos des Typs II, ob jeweils nur ein Sektor bearbeitet werden soll oder mehrere, die direkt aufeinanderfolgen. Ist das Bit gesetzt, so inkrementiert der FDC nach jedem Sektor das SEKTOR-Register.

Die Adreßfeld-Markierung

a0 = 0 – Normales Adress Data Mark
a0 = 1 – Deleted Adress Data Mark

Beim Write-Sector-Kommando bestimmt das Bit 0, ob eine normale Adreßfeld-Markierung oder eine deleted (gelöschte) Adreßfeld-Markierung für den Sektor geschrieben wird. Mit der DELETED ADDRESS DATA MARK werden die Daten auf dem Sektor für gelöscht oder ungültig erklärt. Da das TOS diese Art Daten zu löschen nicht verwendet und diese Methode im Prinzip auch nur bei dem FM-Format vorkommt, sollte das Bit bei dem Write-Sector-Kommando immer gelöscht sein.

Das Verzögerungs-Bit

e = 0 – keine head settling delay time
e = 1 – 30ms head settling delay time

Ein gesetztes e-Bit veranlaßt den FDC, eine Pause von 30 ms einzulegen, damit das Laufwerk seinen Schreib-/Lesekopf vor der Ausführung des Kommandos beruhigen kann.

Die Prekompensation

p = 0 – Prekompensation einschalten
p = 1 – Prekompensation ausschalten

Da der Umfang der äußeren Spuren größer ist als der Umfang der inneren Spuren, kann man bei eingeschalteter Prekompensation einzeln nebeneinanderliegende Pulse dichter zusammenrücken und so eine gleichbleibende Datenkapazität bei höherer Aufzeichnungsdichte erreichen.

Die Interrupt-Bits

I0 = 1 – nicht benutzt
I1 = 1 – nicht benutzt
I2 = 1 – Interrupt bei Indexpulse
I3 = 1 – Sofortiger Interrupt

Das einzige Typ-IV-Kommando bewirkt das Auftreten von Unterbrechungen des in der Ausführung befindlichen Kommandos. Ist das I2-Bit gesetzt, so tritt bei jedem Auftreten des Index-Impulses bei gesetztem I3-Bit ein sofortiger Interrupt ein. Sind alle Ix-Bits gelöscht, wird der laufende Befehl sofort beendet.

Die Programmierung des FDC

Der Ablauf eines FDC-Kommandos hat im allgemeinen acht Schritte:

1. Laufwerk und Seite selektieren
2. DMA-Basisregister laden (DMA_low, DMA_mid, DMA_high)
3. DMA-Puffer und DMA-Statusregister löschen
4. Sector-Count-Register laden
5. FDC Register mit den entsprechenden Werten laden
6. Kommando senden
7. Warten, bis das Interrupt Bit gesetzt bzw. Timeout ist
8. Fehlermeldung abfragen

Mit Ausnahme des Force Interrupt-Kommandos folgt nun für jedes Kommando ein Beispielprogramm. Diese Routinen greifen auf das Laufwerk A und die Seite 0 zu. Vor der Ausführung der Kommandos sollte darauf geachtet werden, daß im TRACK-Register des FDC die korrekte Position des Schreib-/Lesekopfs enthalten ist.

Das TOS speichert für jedes Laufwerk die aktuelle Position in der Drive Configuration Table, DCT. In dieser DCT ist außer der Spur noch die Step-Rate enthalten. Für das ROM-TOS liegt die DCT0 bei $0A06 und die DCT1 bei $0A0A.

Das TOS paßt diese Werte nicht der aktuellen Kopfposition an! Der Programmierer muß daher nach jeder FDC-Routine entweder diese Variablen korrigieren oder den Kopf wieder auf die anfängliche Spur zurücksetzen.

Beispielprogramm 1, RESTORE

```
*               Registerdefinitionen

fdc_reg:        equ     $ff8604         ; FDC Register
dma_mode:       equ     $ff8606         ; DMA Modus Register
dma_low:        equ     $ff860d         ; DMA Basis Register
                                        ; low
dma_mid:        equ     $ff860b         ; DMA Basis Register
                                        ; mid
dma_high:       equ     $ff8609         ; DMA Basis Register
                                        ; high
sound_chip:     equ     $ff8800         ; Sound Chip
psg_data:       equ     $ff8802         ; Sound Chip Data
                                        ; Register
gpip:           equ     $fffa01         ; MFP I/O Port

*
*     Da auf reservierte Speicherbereiche zugegriffen wird, muß
*     die Routine im Supervisormodus des MC68000 ausgeführt
*     werden
*

          clr.l     -(sp)               ; in den
                                        ; Supervisor-Modus
          move.w    #$20,-(sp)          ; gehen
          trap      #1                  ;
          add.l     #6,sp               ; Stack aufräumen
          move.l    d0,ssp              ; User_Stack sichern

*
*    Dann ist es wichtig, die flopvbl-Routine zu sperren, da
*    in ihr das FDC-Statusregister selektiert wird, was zum
*    Abbruch des derzeitig ausgeführten Kommandos fährt.
*

          move.b    #$ff,$43e

*
*    Jetzt werden das Laufwerk A: und die Seite 0 angewählt.
*    Dazu werden die entsprechenden Bits im Port A des Sound-
*    chips gesetzt:
*
*    Bit 0: Seite der Diskette, '0' entspricht Seite 1,
*                               '1' entspricht Seite 0
*    Bit 1: Laufwerk A:   ( '0' Laufwerk wird selektiert    )
*    Bit 2: Laufwerk B:   ( '1' Laufwerk wird deselektiert  )
*

          move.w    sr,-(a7)            ; Prozessor-Status
                                        ; retten
          ori.w     #$700,sr            ; Interrupts sperren
          move.w    #0,D0               ; Laufwerks-Bit (= A:)

          addq.b    #1,D0               ; wird an die richtige
```

```
          lsl.b     #1,D0               ; Position gebracht
          ori.w     #0,D0               ; Seite 0
          eori.b    #7,D0               ; die unteren 3 Bits
                                        ; inv.
          andi.b    #7,D0               ; und die anderen
                                        ; löschen

          move.b    #$e,sound_chip      ; Port A ansprechen
          move.b    sound_chip,D1       ; alten Wert merken
          and.b     #$f8,D1             ; die unteren drei
                                        ; Bits
          or.b      D0,D1               ; setzen
          move.b    D1,psg_data         ; und zurückschreiben
          move.w    (a7)+,sr            ; Interrupts freigeben

          move.w    #$80,dma_mode       ; CR selektieren
          move.w    #$07,fdc_reg        ; RESTORE mit verify


*
*  Anstelle des RESTORE-Kommandos könnte auch
*
*         move.w    #$57,fdc_reg    fuer STEP_IN
*  oder   move.w    #$77,fdc_reg    fuer STEP_OUT
*
*  verwendet werden. Hierbei steht die STEP-RATE bei 3ms, das
*  Spin-up-Bit ist geloescht, das Verify- und Update-Bit
*  gesetzt.
*

*
*  Um das Weiterlaufen der Floppy bei einem Fehler zu
*  verhindern, wird ein Zähler benutzt.
*  Während des Schleifendurchlaufs wird auf eine Mitteilung
*  des FDC gewartet. Da das BUSY Bit nicht abgefragt werden
*  kann, ohne den FDC zu einem Abbruch des Kommandos zu
*  bewegen, wird das 5.Bit im I/O-Port des MFP abgefragt.
*  Dieses zeigt an, ob der FDC einen Interrupt gesendet hat.
*
          move.l    #$60000,D7          ; Time-out Zähler
                                        ; initial.
warten:   subq.l    #1,D7               ; Timer weiterzählen
          beq       status              ; wenn abgelaufen
                                        ; abbrech.
          btst      #5,gpip             ; Kommando fertig?
          bne       warten              ; nein, dann warten
status:   move.w    #$80,dma_mode       ; FDC-Statusreg.
                                        ; selektieren
          move.w    fdc_reg,d0          ; Status nach D0
                                        ; retten

*
*   Das RESTORE-Kommando ist nun beendet, die Floppy wird
*   wieder deselektiert. Sie kann jedoch nicht ohne weiteres
*   deselektiert werden, wenn der Motor noch läuft, da er
```

```
*  sonst endlos weiterlaufen würde.
*
motor:    move.w    fdc_reg,d3          ; Status nach D3
          btst      #7,d3               ; MOTOR ON Bit testen
          bne       motor               ; warten wenn gesetzt

          move.w  sr,-(a7)              ; wie beim Selektieren
          or.w    #$700,sr
          move.b  #$e,sound_chip
          move.b  sound_chip,D1

          and.b   #$f8,D1
          or.b    #7,D1                 ; beide Laufwerke aus!
          move.b  D1,psg_data
          move.w  (a7)+,sr

          clr.w   $43e                  ; flopvbl wieder
                                        ; freigeben

          move.l  ssp,-(sp)             ; und zurück in den
                                        ; User-
          move.w  #$20,-(sp)            ; modus gehen
          trap    #1
          add.l   #6,sp

          clr.w   -(sp)                 ; Programm beenden
          trap    #1

          .bss

ssp:         .ds.l  1                   ; zum Speichern des
*                                       ; Stack Pointers
             .end
```

Beispielprogramm 2, SEEK

In diesem Programm wird der Schreib-/Lesekopf über die Spur 79 positioniert. Im TRACK-Register muß die aktuelle Position des Schreib-/Lesekopfs enthalten sein.

Die Registerdefinitionen entnehmen Sie bitte dem vorigen Beispielprogramm.

```
          clr.l     -(sp)
          move.w    #$20,-(sp)

          add.l     #6,sp
          move.l    d0,ssp
          move.b    #$ff,$43e
          move.w    sr,-(a7)
          or.w      #$700,sr
          move.w    #0,D0               ; Laufwerk A:
          addq.b    #1,D0
```

```
          lsl.b     #1,D0
          or.w      #0,D0                ; Seite 0
          eori.b    #7,D0
          andi.b    #7,D0

          move.b    #$e,sound_chip
          move.b    sound_chip,D1
          and.b     #$f8,D1
          or.b      D0,D1
          move.b    D1,psg_data
          move.w    (a7)+,sr

*
* Für den SEEK Befehl mu das DATA Register des FDC die
* gewünschte Spurnummer enthalten.
*

          move.w    #$86,dma_mode        ; DATA Register selek.
          move.w    track,fdc_reg        ; Track -> DATA Reg.

          move.w    #$80,dma_mode        ; CR selektieren
          move.w    #$17,fdc_reg         ; Seek mit Verify und
                                         ; Update

          move.l    #$60000,D7
warten:   subq.l    #1,D7
          beq       status
          btst      #5,gpip
          bne       warten

status:   move.w    #$80,dma_mode
          move.w    fdc_reg,d0
motor:    move.w    fdc_reg,d3
          btst      #7,d3
          bne       motor

          move.w    sr,-(a7)
          or.w      #$700,sr
          move.b    #$e,sound_chip
          move.b    sound_chip,D1
          and.b     #$f8,D1
          or.b      #7,D1
          move.b    D1,psg_data
          move.w    (a7)+,sr
          clr.w     $43e
          move.l    ssp,-(sp)
          move.w    #$20,-(sp)
          trap      #1
          add.l     #6,sp
          clr.w     -(sp)
          trap      #1

          .data

track:    .dc.w     79                   ; Spur 79
```

```
            .bss
ssp:        .ds.l     1                    ; zum Speichern des
SSP

            .end
```

Damit wären alle Kommandos des Typs I an einem Beispiel abgehandelt.

Beispielprogramm 3, READ SECTOR

Diese Programm liest den Sektor 1 auf der aktuellen Spur.

```
            clr.l     -(sp)                ; In den
                                           ; Supervisormodus
            move.w    #$20,-(sp)           ;  gehen
            trap      #1
            add.l     #6,sp
            move.l    d0,ssp               ; User-Stack sichern
            move.b    #$ff,$43e            ; FLOCK auf -1 setzen
            move.w    sr,-(a7)             ; Prozessor-Status
                                           ; retten
            ori.w     #$700,sr             ; Interrupts sperren
            move.w    #0,D0                ; Laufwerks-Bit (= A:)
            addq.b    #1,D0                ; wird an die richtige
            lsl.b     #1,D0                ; Position gebracht
            ori.w     #0,D0                ; Seite 0
            eori.b    #7,D0                ; die unteren 3 Bits
                                           ; inv.
            andi.b    #7,D0                ; und die anderen
                                           ; löschen
            move.b    #$e,sound_chip       ; Port A ansprechen
            move.b    sound_chip,D1        ; alten Wert merken
            and.b     #$f8,D1              ; die unteren drei
                                           ; Bits
            or.b      D0,D1                ; setzen
            move.b    D1,psg_data          ; und zurückschreiben
            move.w    (a7)+,sr             ; Interrupts freigeben

*
*     Nun wird das Sektor-Register mit der gewünschten Sektor-
*     nummer geladen. Es wird dann auf der aktuellen Spur ein
*     Adressen-Feld gesucht, dessen Sektornummer mit der
*     gesuchten Nummer übereinstimmt und dessen Prüfsumme
*     korrekt ist. Wurde danach das Datenfeld innerhalb von 43
*     Bytes (MFM-Format) nicht gefunden, beendet der FDC das
*     Kommando nach weiteren 4 Versuchen mit einem RECORD NOT
*     FOUND Fehler.
*
            move.w    #$84,dma_mode        ; Sektor-Register
                                           ; selek.
            move.w    sector,fdc_reg       ; Sektornummer laden
```

```
          move.l    #record,-(a7)       ; Jetzt wird die
          move.b    3(a7),dma_low       ; Adresse des Daten-
          move.b    2(a7),dma_mid       ; bereichs der DMA
          move.b    1(a7),dma_high      ; mitgeteilt
          addq.l    #4,a7               ;

*
*   Löschen des DMA-Puffers und des DMA-Statusregisters
*   Datenrichtung der DMA auf LESEN setzen und das
*   Sector-Count Register adressieren
*
          move.w    #$90,dma_mode
          move.w    #$190,dma_mode
          move.w    #$90,dma_mode

*
*   Fuer den Write-Sector Befehl muss die Datenrichtung
*   auf Schreiben geaendert werden. Dafuer muessen die letzten
*   3 Zeilen durch die folgende Sequenz ersetzt werden:
*
*          move.w    #$190,dma_mode
*          move.w    #$90,dma_mode
*          move.w    #$190,dma_mode
*
*   Der Wert im SCR, mit 512 multipliziert ergibt die maximale
*   Anzahl der zu übertragenden Bytes
*

          move.w    #$1,fdc_reg         ; 512 Bytes

*
*   Das READ SECTOR Kommando wird ohne die MULTIPLE-Option
*   durchgeführt.
*

          move.w    #$80,dma_mode       ; CR selektieren
          move.w    #$80,fdc_reg        ; READ SECTOR -> CR

*
*   Fuer ein WRITE_SECTOR Kommando müssen die letzen beiden
*   Zeilen in
*
*          move.w    #$180,dma_mode
*          move.w    #$a0,fdc_reg
*
*   geändert werden.
*

          move.l    #$60000,D7          ; Time-out Zähler
                                        ; initial.
warten:   subq.l    #1,D7               ; Timer weiterzählen
          beq       status              ; wenn abgelaufen
                                        ; abbrech.
          btst      #5,gpip             ; Kommando fertig ?
          bne       warten              ; nein, dann warten
```

```
status:   move.w    #$80,dma_mode       ; FDC-Statusreg.
                                        ; selektieren
          move.w    fdc_reg,d0          ; Status nach D0
                                        ; retten
motor:    move.w    fdc_reg,d3          ; Status nach D3
          btst      #7,d3               ; MOTOR ON Bit testen
          bne       motor               ; warten wenn gesetzt

          move.w    sr,-(a7)            ; wie beim Selektieren
          ori.w     #$700,sr
          move.b    #$e,sound_chip
          move.b    sound_chip,D1
          and.b     #$f8,D1
          ori.b     #7,D1               ; beide Laufwerke aus!
          move.b    D1,psg_data
          move.w    (a7)+,sr

          clr.w     $43e
          move.l    ssp,-(sp)
          move.w    #$20,-(sp)
          trap      #1
          add.l     #6,sp
          clr.w     -(sp)
          trap      #1

          .data

sector:   .dc.w     1                   ; Sektor 1

          .bss

ssp:      .ds.l     1                   ; zum Speichern des
                                        ; SSP
record:   .ds.b     512                 ; 512 Bytes Daten

          .end
```

Beispielprogramm 4,READADRESS

Das READ ADRESS Komanndo liest das nächste Adressenfeldauf der derzeitigen Spur. Das Adressenfeld besteht ausvier Bytes Daten und 2 Bytes Prüfsumme.

```
          clr.l     -(sp)
          move.w    #$20,-(sp)
          trap      #1
          add.l     #6,sp
          move.l    d0,ssp
          move.b    #$ff,$43e
          move.w    sr,-(a7)
          or.w      #$700,sr
          move.w    #0,D0               ; Laufwerk A:
```

```
          addq.b    #1,D0
          lsl.b     #1,D0
          or.w      #0,D0                ; Seite 0
          eori.b    #7,D0
          andi.b    #7,D0
          move.b    #$e,sound_chip
          move.b    sound_chip,D1
          and.b     #$f8,D1
          or.b      D0,D1
          move.b    D1,psg_data
          move.w    (a7)+,sr

          move.l    #feld,-(a7)
          move.b    3(a7),dma_low
          move.b    2(a7),dma_mid
          move.b    1(a7),dma_high
          addq.l    #4,a7

          move.w    #$90,dma_mode        ; Datenrichtung auf
                                         ; Lesen
          move.w    #$190,dma_mode       ;
          move.w    #$90,dma_mode        ;

          move.w    #$1,fdc_reg          ; 512 Bytes
*
*  Da der DMA aber nur jeweils 16 Bytes überträgt, müssen mehr
*  als nur ein Adressenfeld eingelesen werden. Dazu wird
*  dieses Kommando dreimal an den FDC gesendet. Die ersten 6
*  Bytes des Datenbereichs enthalten dann das erste gelesene
*  Adressenfeld
*
          move.l    #3,d4                ; D4 als Zähler
          move.w    #$80,dma_mode        ; COMMAND-Register
read:     move.w    #$C0,fdc_reg         ; Read_Adress
          move.l    #$60000,D7
warten:   subq.l    #1,D7
          beq       status
          btst      #5,gpip
          bne       warten
          subq.l    #1,d4
          bne       read
status:   move.w    #$80,dma_mode

          move.w    fdc_reg,d0
motor:    move.w    fdc_reg,d3
          btst      #7,d3
          bne       motor
          move.w    sr,-(a7)
          or.w      #$700,sr
          move.b    #$e,sound_chip
          move.b    sound_chip,D1
          and.b     #$f8,D1
          or.b      #7,D1
          move.b    D1,psg_data
```

```
          move.w    (a7)+,sr
          clr.w     $43e
          move.l    ssp,-(sp)
          move.w    #$20,-(sp)
          trap      #1
          add.l     #6,sp
          clr.w     -(sp)
          trap      #1

          .bss

ssp:      .ds.l     1                    ; zum Speichern des
                                         ; SSP
feld:     .ds.b     16                   ; 16 Bytes Daten

          .end
```

Beispielprogramm 5,READTRACK

Dieses Programm liest die aktuelle Spur ein.

```
          clr.l     -(sp)
          move.w    #$20,-(sp)
          trap      #1
          add.l     #6,sp
          move.l    d0,ssp
          move.b    #$ff,$43e
          move.w    sr,-(a7)
          or.w      #$700,sr
          move.w    #0,D0                ; Laufwerk A
          addq.b    #1,D0
          lsl.b     #1,D0
          or.w      #0,D0                ; Seite 0
          eori.b    #7,D0
          andi.b    #7,D0

          move.b    #$e,sound_chip
          move.b    sound_chip,D1
          and.b     #$f8,D1
          or.b      D0,D1
          move.b    D1,psg_data
          move.w    (a7)+,sr
          move.l    #spur,-(a7)
          move.b    3(a7),dma_low
          move.b    2(a7),dma_mid
          move.b    1(a7),dma_high
          addq.l    #4,a7

          move.w    #$90,dma_mode        ; Datenrichtung auf
                                         ; Lesen
          move.w    #$190,dma_mode       ;
          move.w    #$90,dma_mode        ;

*
*  Für den Write-Track Befehl muß die Datenrichtung auf
```

```
*  Schreiben geändert werden. Dafür müssen die letzten 3
*  Befehle durch die folgende Sequenz ersetzt werden :
*
*        move.w    #$190,dma_mode
*        move.w    #$90,dma_mode
*        move.w    #$190,dma_mode
*
         move.w    #13,fdc_reg    ; 13 * 512 Bytes = $1a00
                                  ; Bytes Daten
         move.w    #$80,dma_mode
         move.w    #$E0,fdc_reg

*
*  Für ein Write-Track müssen die letzen beiden Befehle in
*
*         move.w    #$180,dma_mode
*         move.w    #$f0,fdc_reg
*
*  geändert werden.
*

         move.l    #$60000,D7
warten:  subq.l    #1,D7
         beq       status
         btst      #5,gpip
         bne       warten
status:  move.w    #$80,dma_mode
         move.w    fdc_reg,d0
motor:   move.w    fdc_reg,d3
         btst      #7,d3
         bne       motor
         move.w    sr,-(a7)
         or.w      #$700,sr
         move.b    #$e,sound_chip
         move.b    sound_chip,D1
         and.b     #$f8,D1
         or.b      #7,D1
         move.b    D1,psg_data
         move.w    (a7)+,sr
         clr.w     $43e
         move.l    ssp,-(sp)
         move.w    #$20,-(sp)
         trap      #1
         add.l     #6,sp
         clr.w     -(sp)
         trap      #1

         .bss

ssp:     .ds.l     1                    ; zum Speichern des
                                        ; SSP
spur:    .ds.b     $1A00                ; $1A00 Bytes Daten

         .end
```

Zusammenfassung

Folgende Register dürfen nicht beschrieben werden, wenn ein Kommando ausgeführt wird, da andernfalls der FDC unterbricht:

- TRACK-Register
- SECTOR-Register
- COMMAND-Register

Bei der Programmierung über die DMA werden, solange das Sector-Count-Register ungleich Null ist, jeweils 512 Bytes übertragen und das SCR dann dekrementiert. Deshalb sollte im SCR die Anzahl als Quotient von den zu übertragenden Bytes durch 512 stehen.

Da der DMA-Puffer immer 16 Bytes weitergibt, muß eine durch 16 teilbare Anzahl von Bytes übertragen werden.

Bei allen Kommandos muß die vbl-Routine ausgeschaltet werden, da in ihr das COMMAND-Register geladen wird.\Maßnahme: Die Speicherstelle $43E mit dem Wert $FF laden und nach dem Beenden des FDC-Programms wieder löschen.

Die DMA-Basis-Register müssen in der Reihenfolge low-mid-high geladen werden.

Das READ/WRITE-Bit im DMA-Modus-Register muß stimmen.

Um einen Bus-Error des MC68000 zu vermeiden, sollten sämtliche Routinen im Supervisor-Modus durchgeführt werden.

Vor der Ausführung jedes Kommandos sollte man togglen, um den Puffer zu löschen, da ansonsten noch Bytes aus vorhergegangenen Kommandos übertragen werden könnten.

Vor jedem Kommando wird die Seite und das Laufwerk im Port A des Soundchips eingestellt (LED leuchtet auf).

Das Laufwerk sollte man erst dann deselektieren, wenn der Motor aus ist.

Kapitel 5

Routinen des XBIOS

Zum besseren Verständnis der FDC-Programmierung werden in diesem Kapitel die XBIOS-Routinen

- floprd (Sektoren einlesen)
- flopwr (Sektoren schreiben)
- flopfmt (Spur schreiben, formatieren)
- flopver (Sektoren prüfen)

anhand eines Assemblerlistings erklärt. In diesem Listing werden symbolische Namen verwendet, damit man auf kein spezielles TOS festgelegt ist. Am Ende dieses Kapitels sind dann die Adressen in verschiedenen TOS-Versionen aufgeführt.

Zunächst folgt jedoch eine Erklärung der Variablen:

curr_error	der momentan getestete oder aufgetretene Fehler
curr_count	Zähler für die Anzahl der Sektoren beim Schreiben oder Lesen
e_dma	Ende des DMA Bereichs bei Read-Multiple
tmp_dma	Zwischenspeicher für die DMA-Adresse
sav_addr	Speicher zum Retten der Registerinhalte
retry_count	Zähler für die Anzahl der Versuche: 2 = 1. Versuch 1 = 2. und letzter Versuch
default_error	Standard-Fehlernummer
curr_sektor	der gewünschte Sektor (beim Aufruf übergeben)
curr_track	der gewünschte Track (beim Aufruf übergeben)
curr_side	die gewünschte Seite (beim Aufruf übergeben)
curr_dma	DMA-Startadresse (beim Aufruf übergeben)
spt	Sektoren pro Track (beim Aufruf übergeben)
interleave	Summand für die Reihenfolge der Sektoren (beim Formatieren übergeben)

virgin	Füllwert (beim Formatieren übergeben)
motor_on_flag	0 = Floppy arbeitet nicht FFFF = Floppy arbeitet
flock	0 = kein Floppy-Zugriff <>0 = Floppy-Zugriff , VBL-Routine ist gesperrt
frclock	Anzahl der ausgeführten VBL-Routinen
wp_st_tab	Write-Protect-Statustabelle
nflops	Anzahl der Floppies
wplatch	Write-Protect-Latch (wird von der Routine Mediach getestet)
deselflag	Diese Variable zeigt an, ob die Floppies selektiert sind.
curr_device	gewünschtes Drive (beim Aufruf übergeben)
acctim	Stand des 200-Hertz-Zählers beim letzten Floppyzugriff.
_hz_200	200-Hertz-Zähler
disknum	Nummer der zuletzt angesprochenen Floppy
dm_tab	Disk-Mode-Tabelle
etv_critic	Vektor auf die Error-Routine
dct0	Drive-Configuration-Table von Drive A word 0 = Position des Kopfes word 1 = Step Rate
dct1	Drive-Configuration-Table von Drive B word 0 = Position des Kopfes word 1 = Step Rate

Programm (floprd):

```
*************************************************************
*                                                           *
* floprd,  Einlesen von Sektoren                            *
*                                                           *
* Format:                                                   *
*                                                           *
*    +---+---+---+---+---+---+---+---+---+---+---+          *
*    |RSP    |puffer |filler |dev|sec|tck|sid|cnt|          *
*    +---+---+---+---+---+---+---+---+---+---+---+          *
*    ^                                                      *
*   SSP                                                     *
*                                                           *
*************************************************************

floprd bsr      change          * Diskettenwechsel?
       moveq.l  #$F5,D0         * D0 -> curr_error
                                * und default_error
                                * $F5 = Read Error
       bsr      param           * Parameter holen und
                                * setzen
next   bsr      select          * Laufwerk und Seite
                                * selektieren
```

```
        bsr       seek_tr              * Kopf auf Track
                                       * positionieren
        bne       whl                  * wenn Fehler, dann noch
                                       * mal
        move.w    #$FFFF,curr_error    * curr_error auf allg.
                                       * Fehler

*             Löschen des DMA-Puffers durch Togglen

        move.w    #$90,(A6)            * DMA-Modus-Register
                                       * Bits 4,7 gesetzt, also
                                       * FDC Sector-Count-Reg.
                                       * lesen/DMA ein
        move.w    #$190,(A6)           * Bits 4,7,8 gesetzt,
                                       * FDC/SCR/schreiben
        move.w    #$90,(A6)            * Bits 4,7 also SCR/lesen

        move.w    curr_count,$FF8604   * curr_count -> SCR
        move.w    #$80,(A6)            * Bit 7 -> Command-Reg.
                                       * oder Status-Reg.
        move.w    #$90,D7              * $90 -> Read Multiple
        bsr       FDC_out              * wird in das CR des FDC
                                       * geschrieben

        move.l    #$40000,D7           * Timeout-Zähler setzen
        move.l    e_dma,A2             * Endadresse DMA in A2
w1      btst      #5,$FFFA01           * Bit 5 des I/O-Ports des
                                       * MFP zeigt an, ob
                                       * FDC fertig?
        beq       fertig               * wenn ja, dann fertig
        subq.l    #1,D7                * Timeout-Zähler
                                       * dekrementieren
        beq       zeit                 * Timer = 0, dann zeit
        move.b    $FF8609,tmp_dma_high* laden der aktuellen
        move.b    $FF860B,tmp_dma_mid  * DMA-Adresse
        move.b    $FF860D,tmp_dma_low  * und vergleichen
        cmpa.l    tmp_dma,A2           * mit DMA-Endadresse
        bgt       w1                   * wenn ungleich, dann
                                       * weiter warten
        bsr       reset_1772           * Ende der Übertragung
                                       * (FDC - Reset)
        bra       fertig               *
zeit    move.w    #$FFFE,curr_error    * $FFFE -> curr_error
                                       * ist timeout - error
        bsr       reset_1772           * Ende der Übertragung
                                       * (FDC Reset)

        bra       whl
fertig  move.w    #$90,(A6)            * Bits 4,7 lesen/DMA/SCR
        move.w    (A6),D0              * DMA-Status -> D0
        btst      #0,D0                * Test, ob DMA-Fehler
        beq       whl                  * wenn DMA-Fehler, dann
                                       * noch einmal
        move.w    #$80,(A6)            * Bit 7 SR oder CR
        bsr       get_FDC              * lesen des Status-Reg.
        and.b     #$18,D0              * Bits 3,4 CRC Error
```

```
                                        * Record not found
        beq     flopok                  * kein Fehler, dann OK
        bsr     Fehlernr.               * Fehler! Nummer bestimm.

whl     cmpi.w  #1,retry_count          * schon zweiter Versuch?
        bne     w2                      * nein

        bsr     test_seek               * schon 2 Versuche also
                                        * testseek

w2      subq.w  #1,retry_count          * retry_count dekr.
        bpl     next                    * wenn retry_count  >  0
                                        * dann neuer Versuch
        bra     flopfail                * kein Versuch mehr, dann
                                        * flopfail
```

```
*************************************************************
*                                                           *
*                                                           *
*                  +----------------+                       *
*              D0  | Status des FDC |                       *
*                  +----------------+                       *
* ----------------------------------------------------------- *
*                  +----------------+                       *
*      curr_error  | Fehlernummer   |                       *
*                  +----------------+                       *
*                                                           *
*    $F3 - Schreibschutz                                    *
*    $F8 - record nicht gefunden                            *
*    $FC - Prüfsummenfehler (CRC)                           *
*                                                           *
*************************************************************
```

```
Fehlernr  moveq.l  #$F3,D1          * (Schreibschutz) -> D1
          btst     #6,D0            * wenn Schreibschutz,
          bne      w11              * dann w11
          moveq.l  #$F8,D1          * (record not found) -> D1
          btst     #4,D0            * wenn dieser Fehler,
          bne      w11              * dann w11
          moveq.l  #$FC,D1          * (Prüfsummenfehler) -> D1
          btst     #3,D0            * wenn CRC-Error,
          beq      w11              * dann w11
          move.w   default_error,D1 * wenn kein Fehler, dann
                                    * default_error nehmen


w11       move.w   D1,curr_error    * Fehlernr. -> curr_error
          rts
```

Programm (flopwr):

```
*************************************************************
*                                                           *
* flopwr, Schreiben von Sektoren                            *
*                                                           *
* Format:                                                   *
*                                                           *
*    +---+---+---+---+---+---+---+---+---+---+---+          *
*    |RSP    |puffer |filler |dev|sec|tck|sid|cnt|          *
*    +---+---+---+---+---+---+---+---+---+---+---+          *
*    ^                                                      *
*   SSP                                                     *
*                                                           *
*************************************************************

flopwr  bsr       change            * Diskettenwechsel?
        moveq.l   #$F6,D0           * F6 = Write Error
                                    * D0 -> default_error
                                    * und curr_error
        bsr       param             * Parameter holen
                                    * und setzen
        move.w    curr_sektor,D0    *
        subq.w    #1,D0             * (curr_sektor -1)
        or.w      curr_track,D0     * oder curr_track
        or.w      curr_side,D0      * oder curr_side
                                    * = 0?
                                    * also Bootsektor?
        bne       w21               * wenn kein Bootsektor
        moveq.l   #2,D0             * wenn Bootsektor, dann
        bsr       setdmtab          * media change = 'changed'
w21     bsr       select            * Drive und Seite
        bsr       seek_tr           * Track finden
        bne       w26               * wenn Fehler, dann w26
w22     move.w    #$FFFF,curr_error * kein Fehler, dann
                                    * curr_error = allg.Fehler


*           DMA-Puffer löschen durch Togglen


        move.w    #$190,(A6)        * Bits 4,7,8 SCR/schreiben
        move.w    #$90,(A6)         * Bits 4,7 SCR/lesen
        move.w    #$190,(A6)        * Bits 4,7,8 SCR/schreiben
        move.w    #1,D7             * 1 in das Sektor-Count-
        bsr       FDC_out           * Register schreiben
        move.w    #$180,(A6)        * Bits 7,8 SR/CR/schreiben
        move.w    #$A0,D7           * $A0 in CR des FDC
        bsr       FDC_out           * also Write-Sektor
        move.l    #$40000,D7        * Timer = $40000
w23     btst      #5,$FFFA01        * FDC fertig?
        beq       w24               * wenn fertig, dann w24
        subq.l    #1,D7             * Timer dekrementieren
        bne       w23               * wenn Timer > 0, dann
```

```
w23
        bsr       reset_1772        * Ende der Übertragung
                                    * (FDC-Reset)
        bra       w25
w24     move.w    #$180,(A6)        * Bits 7,8 SR/CR/schreiben
        bsr       get_FDC           * Status-Register lesen
        bsr       Fehlernr          * Fehlernummer?
        btst      #6,D0             * Schreibschutz?
        bne       flopfail          * Abbruch, da Fehler
        and.b     #$5C,D0           * Bits 2,3,4,6
                                    * Write Protect,RNF,
                                    * Checksum,Lost Data
        bne       w25               * Fehler, neuer Versuch
                                    * kein Fehler
        addq.w    #1,curr_sektor    * nächster Sektor
        addi.l    #$200,curr_dma    * DMA-Adresse um 512
                                    * erhöhen
        subq.w    #1,curr_count     * wieder einer weniger
        beq       flopok            * wenn alle Sektoren
                                    * geschrieben sind, dann
                                    * Ende ohne Fehler
        bsr       w171              * current_sector in Sektor
                                    * Register des FDC, und
                                    * DMA setzen
        bra       w22               * Sektor schreiben
w25     cmpi,w     #1,retry_count   * wenn nicht letzter
        bne       w27               * Versuch dann w27
w26     bsr       testseek          * Test auf Seek Error
w27     subq.w    #1,retry_count    * ein Versuch weniger
        bpl       w21               * noch ein Versuch
        bra       flopfail          * Beenden mit Fehler
```

Programm (flopfmt):

```
*************************************************************
*                                                           *
* flopfmt, Schreiben (Formatieren) einer Spur               *
*                                                           *
* Format:                                                   *
*                                                           *
* +---+---+---+---+---+---+---+---+---+---+---+---+---+---+ *
* |  RSP  |puffer |filler |dev|spt|trk|sid|ilv| magic |vrg| *
* +---+---+---+---+---+---+---+---+---+---+---+---+---+---+ *
* ^                                                         *
* Stackpointer                                              *
*                                                           *
*************************************************************

flopfmt cmpi.l    #$87654321,$16(A7) * Stimmt die magische Zahl
        bne       flopfail           * nein, Ende mit Fehler
        bsr       change             * Diskette gewechselt?
```

```
        moveq.l  #$FF,D0            * $FF = allg. Fehler als
        bsr      param              * default_error und
                                    * curr_error
                                    * Parameter holen und
                                    * setzen
        bsr      select             * Laufwerk und Seite
        move.w   $E(A7),spt         * spt holen
        move.w   $14(A7),interleave * interleave holen
        move.w   $1A(A7),virgin     * virgin holen
        moveq.l  #2,D0              * Drive_change_modus  auf
        bsr      setdmtab           * 'gewechselt' setzen
        bsr      find_tr            * Track suchen
        bne      flopfail           * bei Fehler Ende
        move.w   curr_track,$0(A1)  * aktuellen Track in DCT
        move.w   #$FFFF,curr_error  * curr_error=allg. Fehler
        bsr      fmtrack            * Track formatieren
        bne      flopfail           * Ende mit Fehler
        move.w   spt,curr_count     * spt als Zähler
        move.w   #1,curr_sektor     * Bei Sektor 1 beginnend
        bsr      verify             * verifizieren
        move.l   curr_dma,A2        * curr_dma ist Liste der
                                    * schlechten Sektoren
        tst.w    (A2)               * Wenn curr_DMA = 0
                                    * (Liste ist leer),
        beq      flopok             * dann Ende ohne Fehler
        move.w   #$FFF0,curr_error  * sonst 'Bad Sector' ($F0)
        bra      flopfail           * Ende mit Fehler



        ******************************************
        *                                        *
        *    Erstellen und Schreiben des Puffers *
        *                                        *
        ******************************************

fmtrack move.w   #$FFF6,default_error* default_error =
                                    * 'Write Error'
        move.w   #1,D3              * Start mit Sektor 1
        move.l   curr_dma,A2        * Adr. des Puffers in A2
        move.w   #$3B,D1            * 60mal
        move.b   #$4E,D0            * $FE in den Puffer
        bsr      fill_buf           * schreiben ('L')

*               L0,Ad_x,L1,Dat_x,L2 erstellen

w31     move.w   D3,D4              * Sektornummer in D4
w32     move.w   #11,D1             * 12mal
        clr.b    D0                 * $00 in den Puffer
        bsr      fill_buf           * schreiben ('L0')
        move.w   #2,D1              * 3mal
        move.b   #$F5,D0            * $F5 in den Puffer
        bsr      fill_buf           * schreiben ('L0')
        move.b   #$FE,(A2)+         * $FE  ('Ad_x-Marke')
```

```
        move.b   curr_track,(A2)+    * ('Ad_x-Spur')
        move.b   curr_side,(A2)+     * ('Ad_x-Seite')
        move.b   D4,(A2)+            * Sek.nr.('Ad_x-Sektor')
        move.b   #2,(A2)+            * $02  ('Ad_x-Größe')
        move.b   #$F7,(A2)+          * $F7  Checksum erzeugen
        move.w   #$15,D1             * 22mal
        move.b   #$4E,D0             * $4E in den Puffer
        bsr      fill_buf            * schreiben ('L1')
        move.w   #11,D1              * 12mal
        clr.b    D0                  * $00 in den Puffer
        bsr      fill_buf            * schreiben ('L1')
        move.w   #2,D1               * 3mal
        move.b   #$F5,D0             * $F5 in den Puffer
        bsr      fill_buf            * schreiben ('L1')
        move.b   #$FB,(A2)+          * $FB  ('Dat_x-Marke')
        move.w   #$FF,D1             * 256mal

w33     move.b   virgin_high,(A2)+   * virgin_high und
        move.b   virgin_low,(A2)+    * virgin_low in Puffer
        dbra     D1,w33              * ('Dat_x-Daten')
        move.b   #$F7,(A2)+          * $F7  Checksum erzeugen
        move.w   #$27,D1             * 40mal
        move.b   #$4E,D0             * $4E in den Puffer
        bsr      fill_buf            * schreiben ('L2')
        add.w    interleave,D4       * interleave addieren
                                     * ergibt nächsten Sektor
        cmp.w    spt,D4              * wenn nächster Sektor
        ble      w32                 * kleiner gleich spt,
                                     * formatiere den Sektor
        addq.w   #1,D3               * wenn Startsektor + 1
        cmp.w    interleave,D3       * <= interleave,
        ble      w31                 * dann weiter mit neuem
                                     * Startsektor
        move.w   #$578,D1            * 1401mal
        move.b   #$4E,D0             * $4E in den Puffer
        bsr      fill_buf            * schreiben
        move.b   dma_low,$FF860D     * DMA-Adresse laden
        move.b   dma_mid,$FF860B     *
        move.b   dma_high,$FF8609    *
        move.w   #$190,(A6)          * DMA-Puffer löschen
        move.w   #$90,(A6)           * und Modus auf
        move.w   #$190,(A6)          * SCR/schreiben
        move.w   #$1F,D7             * 31 in das Sector-Count-
        bsr      FDC_out             * Register schreiben
        move.w   #$180,(A6)          * CR/SR des FDC-
        move.w   #$F0,D7             * Kommando Write-Track
        bsr      FDC_out             * in CR des FDC
        move.l   #$40000,D7          * Zähler initialisieren

w34     btst     #5,$FFFA01          * FDC fertig?
        beq      test1               * wenn ja, dann test1
        subq.l   #1,D7               * sonst dekrementieren
        bne      w34                 * und wenn Zähler > 0
                                     * weiter warten
        bsr      reset_1772          * sonst Übertr. beenden
                                     * FDC-Reset
```

```
w35     moveq.l  #1,D7              * 1 nach D7
                                    * bedeutet, ein Fehler
                                    * ist aufgetreten
        rts                         * und Ende
```

```
        ****************************************
        *                                      *
        *                                      *
        *    Testen, ob Fehler aufgetreten ist *
        *                                      *
        *                                      *
        ****************************************
```

```
test1   move.w   #$190,(A6)         * SCR/schreiben
        move.w   (A6),D0            * DMA-Status
        btst     #0,D0              * auf DMA-Fehler testen
        beq      w35                * wenn Fehler, dann w35
        move.w   #$180,(A6)         * CR/SR/schreiben
        bsr      get_FDC            * Status-Register lesen
        bsr      Fehlernr           * Fehlernummer berechnen
        and.b    #$44,D0            * Bits 2,6 Lost Data und
                                    * Schreibschutz testen
        rts
```

```
************************************************************
*                                                          *
*    Puffer mit Daten füllen                               *
*                                                          *
*                                                          *
*   D0 - Wert, der geschrieben werden soll                 *
*                                                          *
*   D1 - Anzahl, mit der der Wert in den Puffer geschrieben *
*        werden soll                                       *
*                                                          *
************************************************************
```

```
fill_buf  move.b    D0,(A2)+        * D0 in Puffer und
                                    * A2 inkrementieren
          dbra      D1,fill_buf     * D1 dekrementieren und
                                    * wenn D1 > 0, nochmal
          rts
```

Programm (flopver):

```
*************************************************************
*                                                           *
*  flopver, Prüfen (Verifizieren) von Sektoren              *
*                                                           *
*  Format:                                                  *
*                                                           *
*    +---+---+---+---+---+---+---+---+---+---+---+          *
*    |RSP    |puffer |filler |dev|sec|tck|sid|cnt|          *
*    +---+---+---+---+---+---+---+---+---+---+---+          *
*    ^                                                      *
*   SSP                                                     *
*                                                           *
*************************************************************

flopver bsr     change              * Diskette gewechselt?
        moveq.l #$F5,D0             * Read Error als
                                    * default_error
                                    * und curr_error
        bsr     param               * Parameter holen und
                                    * setzen
        bsr     select              * Seite und Drive
                                    * selektieren
        bsr     seek_tr             * Track suchen
        bne     flopfail            * bei Fehler Ende
        bsr     verify              * verifizieren
        bra     flopok              * alles OK und Ende



                ********************************
                *                              *
                *   Verifizieren der Sektoren  *
                *                              *
                ********************************

verify  move.w  #$FFF5,default_error* default_error auf
                                    * Read error
        move.l  curr_dma,A2         * A2  = Adresse für die
                                    * Bad-Sector-List
        addi.l  #$200,curr_dma      * curr_dma auf nächsten
                                    * Sektor
w41     move.w  #2,retry_count      * 2 Versuche
        move.w  #$84,(A6)           * SECTOR-Register
                                    * adressieren
        move.w  curr_sektor,D7      * Sektornummer in das
        bsr     FDC_out             * SR schreiben
w42     move.b  dma_low,$FF860D     * DMA-Adresse setzen
        move.b  dma_mid,$FF860B     *
        move.b  dma_high,$FF8609    *
```

```
        move.w   #$90,(A6)          * DMA-Puffer löschen
        move.w   #$190,(A6)         * und
        move.w   #$90,(A6)          * SCR adressieren
        move.w   #1,D7              * Sector-Count-Reg. auf
        bsr      FDC_out            * 1 setzen
        move.w   #$80,(A6)          * CR/SR adressieren
        move.w   #$80,D7            * Read-Sektor in das
        bsr      FDC_out            * COMMAND-Reg. schreiben
        move.l   #$40000,D7         * Zähler initialisieren

w43     btst     #5,$FFFA01         * FDC fertig?
        beq      test2              * wenn ja dann test2
        subq.l   #1,D7              * Zähler dekrementieren
        bne      w43                * wenn Zähler >0 dann w43
        bsr      reset_1772         * sonst Übertr. beenden
                                    * (FDC-Reset)
        bra      repeat             * nächster Versuch


        ******************************************
        *                                        *
        *  Testen, ob ein Fehler aufgetreten ist *
        *                                        *
        *                                        *
        ******************************************

test2   move.w   #$90,(A6)          * DMA-Status-Register
        move.w   (A6),D0            * lesen und testen, ob
        btst     #0,D0              * DMA-Error
        beq      repeat             * wenn Fehler, dann
                                    * nächster Versuch
        move.w   #$80,(A6)          * SR adressieren
        bsr      get_FDC            * und lesen
        bsr      Fehlernr           * Fehlernummer?
        and.b    #$1C,D0            * Bits 2,3,4
                                    * Lost Data, CRC, Record
                                    * not found testen
        bne      repeat             * wenn Fehler dann
                                    * nächster Versuch

w44     addq.w   #1,curr_sektor     * nächster Sektor in
                                    * curr_sektor
        subq.w   #1,curr_count      * noch ein Sektor weniger
        bne      w41                * wenn noch ein Sektor zu
                                    * verifizieren ist, dann
                                    * W41
        subi.l   #$200,curr_dma     * DMA-Zeiger wieder um
                                    * einen Sektor zurück
        clr.w    (A2)               * Bad-Sector-List mit 0
        rts                         * beenden
```

```
*************************************************************
*                                                           *
*    Testen, ob zweiter Versuch                             *
*                                                           *
*    Wenn beim zweiten Versuch ein  Fehler  auftritt, wird  *
*    diese Sektornummer in die Bad-Sector-List geschrieben  *
*                                                           *
*************************************************************

repeat  cmpi.w   #1,retry_count      * Wenn kein Versuch mehr
        bne      w45                 * möglich, dann kein
reseek
        bsr      testseek            * Test, ob Seek Error
w45     subq.w   #1,retry_count      * ein Versuch weniger
        bpl      w42                 * wenn noch ein Versuch
                                     * möglich, dann w42
        move.w   curr_sektor,(A2)+   * curr_sektor in
                                     * Bad-Sector-List
        bra      w44



*************************************************************
*                                                           *
*         Floppy Vertical Blank Handler                     *
*                                                           *
*************************************************************


flopvbl suba.l   A5,A5               * A5 löschen
        lea      $FF8606,A6          * DMA-Modus/Status Reg.
        st       motor_on_flag       * motor_on_flag setzen
        tst.w    flock               * Systemvar. flock ist

                                     * dann <> 0, wenn gerade
                                     * ein Floppy-Zugriff
                                     * stattfindet
                                     * bei flock <> 0
        bne      w54                 * nichts machen
        move.l   frclock,D0          * die Systemvariable
                                     * frclock enthält die
                                     * Anzahl der ausgeführten
                                     * VBL-Routinen
        move.b   D0,D1
        and.b    #7,D1               * mod 8
        bne      w52                 * wenn noch nicht
                                     * 8. Interrupt
        move.w   #$80,(A6)           * sonst CR/SR adressieren
        lsr.b    #3,D0               * Bit 4 isolieren
        and.w    #1,D0
        lea      wp_st_tab,A0        * wp-Status-Tabelle
        adda.w   D0,A0               * D0=1 => A0 Adresse des
                                     * wp-Status von Floppy B
                                     * D0=0 => A0 Adresse des
                                     * wp-Status von Floppy A
```

```
        cmp.w    nflops,D0          * die Systemvariable
                                    * nflops enthält die
        bne      w51                * Anzahl der Floppies
        clr.w    D0                 * wenn Floppy B
                                    * angesprochen aber
                                    * nflops=1 -> D0 löschen
                                    * sonst w51
w51     addq.b   #1,D0              * 00 -> 010 => Drive A
        lsl.b    #1,D0              * 01 -> 100 => Drive B
        eori.b   #7,D0              * invert. da low-aktiv
        bsr      setdrive           * Drive selektieren
        move.w   $FF8604,D0         * SR lesen
        btst     #6,D0              * Schreibschutz testen
        sne      (A0)               * und in die wp-Status-
                                    * Tabelle schreiben
        move.b   D2,D0              * vorh. Select-Status
        bsr      setdrive           * wieder herstellen
w52     move.w   wp_st_tab,D0       * wp_st_tab in wp_latch
        or.w     D0,wp_latch        *
        test.w   deselflag          * Floppies schon
                                    * deselektiert
        bne      w53                * dann weiter
        bsr      get_FDC            * SR lesen
        btst     #7,D0              * wenn Motor on, dann
        bne      w54                * Ende
        move.b   #7,D0              * beide Drives
        bsr      setdrive           * deselektieren
        move.w   #1,deselflag       * deselflag setzen
w53     clr.w    motor_on_flag      * motor_on_flag löschen
w54     rts
```

```
*************************************************************
*                                                           *
*   Einstellen der Floppy-Parameter                         *
*                                                           *
*    retry_count - 2                                        *
*                                                           *
*************************************************************
```

```
param   movem.l D3-D7/A3-A6,sav_addr* Register retten
        suba.l  A5,A5               * A5 löschen
        lea     $FF8606,A6          * A6 = DMA-Modus/Status R.
        st      motor_on_flag       * motor_on_flag setzen
        move.w  D0,default_error    * D0 als default_error
        move.w  D0,curr_error       * und curr_error
        move.w  #1,flock            * VBL-Routine d. Setzen
                                    * der Systemvariable flock
                                    * sperren
        move.l  $8(A7),curr_dma     * curr_dma vom Stack holen
        move.w  $10(A7),curr_device * curr_device holen
```

```
        move.w  $12(A7),curr_sektor * curr_sektor holen
        move.w  $14(A7),curr_track  * curr_track holen
        move.w  $16(A7),curr_side   * curr_side holen
        move.w  $18(A7),curr_count  * curr_count holen
        move.w  #2,retry_count      * Anzahl d. Wiederholungen
                                    * auf 2 setzen
        lea     dct0,A1             * wenn curr_device = 0
        tst.w   curr_device         * dann A1 = dct0
        beq     w61                 *
        lea     dct1,A1             * sonst A1 = dct1


w61     moveq.l #0,D7               * 0000 im oberen Wort,
        move.w  curr_count,D7       * curr_count im unteren
                                    * Wort von D7
        lsl.w   #8,D7               * mal 512 ergibt Bytes
        lsl.w   #1,D7               *
        move.l  curr_dma,A0         * DMA-Startadresse in A0
        adda.l  D7,A0               * plus Bytes
        move.l  A0,e_dma            * ergibt DMA-Endadresse
        tst.w   $0(A1)              * aktueller Track > 0
        bpl     w63                 * dann w63
        bsr     select              * sonst select
        clr.w   $0(A1)              * aktueller Track = 0
        bsr     restore             * Track Null suchen
        beq     w63                 * kein Fehler, dann w63
        moveq.l #10,D7              * Track 10
        bsr     w80                 * suchen
        bne     w62                 * bei Fehler w62
        bsr     restore             * sonst Track Null suchen
        beq     w63                 * kein Fehler, dann w63

w62     move.w  #$FF00,$0(A1)       *

w63     rts



*************************************************************
*                                                           *
*  Fehler ist aufgetreten                                   *
*                                                           *
*                                                           *
*                                                           *
*************************************************************


flopfail  moveq.l   #1,D0           * media_change auf
          bsr       setdmtab        * 'unsure' setzen
          move.w    curr_error,D0   * curr_error in
          ext.l     D0              * D0.l schreiben
          bra       w71
```

```
*************************************************************
*                                                           *
* Kein Fehler ist aufgetreten                               *
*                                                           *
*************************************************************

flopok  clr.l    D0                    * kein Fehler
w71     move.l   D0,-(A7)              * 0 übergeben
        move.w   #$86,(A6)             * Bits 1,2,7  DATA-Reg.
        move.w   $0(A1),D7             * aktuellen Track in das
        bsr      FDC_out               * DR schreiben
        move.w   #$10,D6               * Seek-Befehl
        bsr      do_FDCcmd             * an den FDC senden
        move.w   curr_dev,D0           * Drive-Nummer
        lsl.w    #2,D0                 * als Index
        lea      acctim,A0             * Adresse acctim  in A0
        move.l   _hz_200,$0(A0,D0.w)   * _hz_200 ist der Zähler
                                       * für 200-Hz-Systemtakt
        cmpi.w   #1,nflops             * wenn eine Floppy, dann
        bne      w72                   *
        move.l   _hz_200,$4(A0)        * 200-Hz-Zähler für
                                       * anderes Drive
w72     move.l   (A7)+,D0              * Fehlernummer holen
        movem.l  sav_addr,D3-D7/A3-A6* gerettete Register
                                       * zurückholen
        clr.w    flock                 * VBL-Routine freigeben
        rts


*************************************************************
*                                                           *
* Kopf auf Track positionieren                              *
*                                                           *
*************************************************************

find_tr  move.w   curr_track,D7        * track in D7
w80      move.w   #$FFFA,curr_error    * bei Fehler Seek Error
         move.w   #$86,(A6)            * Bits 1,2,7 TR adress.
         bsr      FDC_out              * D7 in TR schreiben
         move.w   #$10,D6              * Bit 4 - SEEK-Befehl
         bra      do_FDCcmd            * an FDC senden



*************************************************************
*                                                           *
* Kopf auf Track 0 und dann auf 5                           *
*                                                           *
*************************************************************

testseek move.w   #$FFFA,curr_error    * curr_error='seek error'
         bsr      restore              * Kopf auf Track Null
         bne      w81                  * wenn Fehler, dann Ende
         clr.w    $0(A1)               * aktueller Track = 0
```

```
          move.w    #$82,(A6)           * TR adressieren
          clr.w     D7                  * 0 in
          bsr       FDC_out             * TR schreiben
          move.w    #$86,(A6)           * DR adressieren
          move.w    #5,D7               * 5 (Ziel-Track)
          bsr       FDC_out             * in DR schreiben
          move.w    #$10,D6             * Bit 4 - SEEK-Befehl
          bsr       do_FDCcmd           * an FDC senden
          bne       w81                 * bei Fehler Ende
          move.w    #5,$0(A1)           * aktuelle Track-Nummer=5
seek_tr   move.w    #$FFFA,curr_error   * curr_error='seek error'
          move.w    #$86,(A6)           * DR adressieren
          move.w    curr_track,D7       * Ziel-Track in DR
          bsr       FDC_out             * schreiben
          moveq.l   #$10,D6             * SEEK Befehl
          bsr       do_FDCcmd           * an FDC senden
          bne       w81                 * bei Fehler Ende
          move.w    curr_track,$0(A1)   * Kopfposition als
                                        * aktuellen Track
          and.b     #$18,D7             * Bits 3,4  Prüfsumme und
                                        * Record not found testen
w81       rts                           * Zero flag = 0
                                        * bedeutet Fehler


*************************************************************
*                                                           *
* Kopf auf track 0                                          *
*                                                           *
*************************************************************

restore   clr.w     D6                * Restore Kommando an
          bsr       do_FDCcmd         * FDC senden (D7 = SR)
          bne       w91               * bei Fehler Ende
          btst      #2,D7             * Track 00 Bit testen
          eori.b    #4,CCR            * Bit 4 = 'Zero-Flag'
                                      * invertieren
          bne       w91               * wenn Track<>00 -> Ende
          clr.w     $0(A1)            * sonst aktuellen Track
                                      * übernehmen
w91       rts


*************************************************************
*                                                           *
* Kommando an FDC senden                                    *
*                                                           *
*    Befehl in D6                                           *
*    Status Register in D7 zurück                           *
*                                                           *
*************************************************************

do_FDCcmd move.w    $2(A1),D0         * Step Rate in D0
          and.b     #3,D0             * Bit 0 und 1 isolieren
          or.b      D0,D6             * in den Befehl einbauen
          move.l    #$40000,D7        * Timer initialisieren
```

```
          move.w    #$80,(A6)        * CR/SR adressieren
          bsr       get_FDC          * SR lesen
          btst      #7,D0            * Bit 7  Motor_on testen
          bne       w101             * ja, dann weiter
          move.l    #$60000,D7       * sonst Timer erhöhen
w101      bsr       D6->FDC          * Befehl von D6 in CR
w102      subq.l    #1,D7            * Timer dekrementieren
          beq       w103             * wenn abgelaufen, dann
                                     * Ende
          btst      #5,$FFFA01       * IO-Port/FDC fertig?
          bne       w102             * wenn nein, dann weiter
                                     * warten
          rts
w103      bsr       reset_1772       * Übertragung beenden
                                     * (FDC Reset)
          moveq.l   #1,D6            * Zero Flag löschen
                                     * bedeutet Fehler
          rts



*****************************************************************
*                                                               *
*    FDC Reset                                                  *
*                                                               *
*****************************************************************

reset_1772  move.w  #$80,(A6)        * CR/SR adressieren
          move.w    #$D0,D7          * 4,6,7 Reset Befehl
          bsr       FDC_out          * an FDC senden
          move.w    #15,D7           * Verzögerungszähler von
w111      dbra      D7,w111          * 16 auf 0 dekrementieren
          bsr       FDC->D7          * SR in D7
          rts



*****************************************************************
*                                                               *
* Seite und Drive selektieren                                   *
*                                                               *
*****************************************************************

select  clr.w     deselflag             * deselflag löschen
        move.w    curr_device,D0        * Drive Bit
        addq.b    #1,D0                 * 00 -> 010  Drive A
        lsl.b     #1,D0                 * 01 -> 100  Drive B
        or.w      curr_side,D0          * Seitennummer in Bit
        eori.b    #7,D0                 * 0 invertieren, da
                                        * low-aktiv
        and.b     #7,D0                 * nur Bits 0 bis 2
        bsr       setdrive              * in Port A des
                                        * Soundchips setzen
```

```
        move.w   #$82,(A6)              * TR adressieren
        move.w   $0(A1),D7              * aktuelle Track-Nr.
        bsr      FDC_out                * in TR schreiben
        clr.b    tmp_dma                * tmp_dma High-Byte
                                        * löschen

w171    move.w   #$84,(A6)              * Sektor-Register
                                        * adressieren
        move.w   curr_sektor,D7         * Sektor in
        bsr      FDC_out                * Sektor-Re. schreiben
        move.b   curr_dma_low,$FF860D   * DMA-Register setzen
        move.b   curr_dma_mid,$FF860B
        move.b   curr_dma_high,$FF8609
        rts


************************************************************
*                                                          *
* Port A im Sound Chip setzen                              *
*                                                          *
*    Neue Bitkonfiguration in D0                           *
*                                                          *
************************************************************


setdrive  move.w    SR,-(A7)        * Statusregister retten
          ori.w     #$700,SR        * Interrupt-Mask = 111
                                    * setzen d.h. Interrupt-
                                    * Level 7, also sperren
                                    * aller Interrupts
                                    * da die Adresse $FF8800
                                    * beim Lesen immer das
                                    * zuletzt angesprochene
                                    * Register des Soundchips
                                    * adressiert
                                    * also Interrupt sperren,
                                    * damit kein anderes
                                    * Register angesprochen
                                    * werden kann
          move.b    #14,$FF8800     * Register 14 (Port A) des
                                    * Soundchips adressieren
          move.b    $FF8800,D1      * Port A lesen
          move.b    D1,D2           * und in D2
          and.b     #$F8,D1         * Bits 0 bis 2 löschen
          or.b      D0,D1           * und durch neue Bits
                                    * ersetzen
          move.b    D1,$FF8802      * und in Port A schreiben
          move.w    (A7)+,SR        * Statusregister
                                    * regenerieren
          rts
```

```
*************************************************************
* Die Verzögerungsschleife ist eingebaut da der FDC  nach   *
* Erhalt des Force Interrupt-Kommandos bei FM-Format        *
* 32 µs und bei MFM-Format 16 µs lang kein anderes          *
* Kommando empfangen darf, da sonst das Force Interrupt-    *
* Kommando nicht ausgeführt wird.                           *
*                                                           *
* Allerdings ergibt das einen Zeitverlust bei der Ausführung*
* der Befehle zumal die Ausführung der Warteschleife        *
* erheblich länger dauert  (422 Taktzyklen = ca. 52.75 µs)  *
*                                                           *
* Befehl                              Taktzyklen            *
* bsr                                 18    =>  18          *
* move.w SR,-(a7)               +     8+6   =>  14          *
* move.w D7,-(a7)               +     8     =>   8          *
* move.w #$20,D7                +     8     =>   8          *
* move.w (a7)+,D7               +     8     =>   8          *
* move.w (a7)+,SR               +     12+4  =>  16          *
* rts                           +     16    =>  16          *
* dbra D7,  32mal branch        +     10*32 => 320    ca.10 *
* dbra d7   1mal nicht branch   +     14    =>  14          *
*                                                 ---       *
*                                                 422       *
*                                                           *
*************************************************************


D6->FDC   bsr       wait          * Verzögerungsschleife
          move.w    D6,$FF8604    * D6 in adressiertes
                                  * Register schreiben
          bra       wait          * Verzögerungsschleife

FDC_out   bsr       wait          * Verzögerungsschleife
          move.w    D7,$FF8604    * D7 in adressiertes
                                  * Register schreiben
          bra       wait          * Verzögerungsschleife

FDC->D7   bsr       wait          * Verzögerungsschleife
          move.w    $FF8604,D7    * adressiertes Register
                                  * in D7 lesen
          bra       wait          * Verzögerungsschleife

get_FDC   bsr       wait          * Verzögerungsschleife
          move.w    $FF8604,D0    * adressiertes Register
                                  * in D0 lesen

wait      move.w    SR,-(A7)      * Statusregister retten
          move.w    D7,-(A7)      * D7 retten
          move.w    #$20,D7       * Zähler initialisieren

w181      dbra      D7,w181       * Schleife
          move.w    (A7)+,D7      * D7 zurückholen
          move.w    (A7)+,SR      * Statusregister
                                  * zurückholen
```

```
*************************************************************
*                                                           *
* Bei Aufruf einer Floppy-Routine                           *
* testen, ob Diskette von A nach B oder von B nach A        *
* gewechselt (bei einem Drive).                             *
* und wenn, dann Aufforderung zum Wechseln                  *
*                                                           *
*************************************************************


change    cmpi.w    #1,nflops        * wenn nicht eine Floppy,
          bne       term             * dann fertig
          move.w    $10(A7),D0       * sonst Drive-Nummer mit
          cmp.w     disknum,D0       * Disketten-Nummer
                                     * vergleichen
          beq       w191             * wenn gleich, dann w191
          move.w    D0,-(A7)         * sonst Drive-Nummer auf
                                     * den Stack
          move.w    #$FFEF,-(A7)     * Fehlernummer -17
                                     * 'Diskette einlegen'
          bsr       h_error          * Fehler berichtigen
          addq.w    #4,A7            * Drive und Fehlernummer
                                     * auf Stack vergessen
          move.w    #$FFFF,wp_latch  * Drives A und B sind
                                     * 'unsure'
          move.w    $10(A7),disknum  * Disketten-Nummer auf
                                     * Drive-Nummer

w191      clr.w     $10(A7)          * Drive-Nummer auf 0
                                     * Drive = A
term      rts
```

```
*************************************************************
*                                                           *
*  Drive-Modus setzen                                       *
*                                                           *
*    Drive-Modus in D0                                      *
*                                                           *
*************************************************************


setdmtab  lea       dm_tab,A0          * Disk_mode_tabelle in A0
          move.b    D0,-(A7)           * D0 auf den Stack
          move.w    curr_device,D0     * dm_tab mit Drive
                                       * indizieren und
          move.b    (A7)+,$0(A0,D0.w)  * Drive-Modus dorthin
                                       * schreiben
          rts
```

```
*************************************************************
*                                                           *
* Einsprung in die Error Handling  Routine , die versucht   *
* Fehler zu korrigieren ( z.B. 'Insert Disk B in Drive A )  *
*                                                           *
*************************************************************

h_error  move.l  etv_critic,-(A7)    * etv_critic ist ein
                                     * Systemvektor, der auf
                                     * die Routine zur
                                     * Fehlerbehandlung zeigt
         moveq.l #$FF,D0             *
         rts                         * dieser Vektor wird  bei
                                     * rts angesprungen
```

Ein XBIOS-Fehler

Wenn Sie die Leseroutine des XBIOS etwas genauer betrachten, können Sie einen groben Programmierfehler erkennen. Bei den Kommandos Read Multiple und Write Multiple bricht der FDC die Ausführung erst ab, wenn die Nummer des Sektors größer als die maximale Anzahl der Sektoren je Spur ist. Das XBIOS des ATARI vergleicht nun die aktuelle DMA-Adresse mit der DMA-Adresse, die sich aus der Anzahl der Sektoren, mit 512 multipliziert, errechnet. Ist diese Adresse erreicht, wird dem FDC ein RESET-Kommando gesendet, was die Datenübertragung von und zur Diskette sofort abbricht. Anschließend müßten die 2 CRC-Bytes zur Überprüfung der Daten gelesen werden, was der FDC nun jedoch unterläßt. Somit wird ein eventueller Prüfsummenfehler im letzten Sektor nicht mehr erkannt!

Dieser Bug tritt jedoch nur bei den Routinen auf, die die Multiple-Befehle verwenden. Die flopwr-Routine arbeitet z.B. ohne Multiple-Option. Dort wird jeder Sektor einzeln eingelesen.

Nun noch die Adressen der Routinen in den verschiedenen TOS-Versionen:

Name	neues TOS 196480 Bytes	altes TOS 207128 Bytes	ROM
floprd	$7424	$62D2	$FC159E
flopwr	$750A	$63B0	$FC167C

Name	neues TOS 196480 Bytes	altes TOS 207128 Bytes	ROM
flopfmt	$75C2	$6468	$FC1734
flopver	$775C	$6602	$FC18CE
flopvbl	$783E	$66E4	$FC19B0
Fehlernr	$74E8	$638E	$FC165A
curr_error	$09E0	$06A2	$09E0
curr_count	$09CA	$068C	$09CA
e_dma	$09D0	$0692	$09D0
tmp_dma	$09DA	$069C	$09DA
sav_addr	$09E2	$06A4	$09E2
retry_count	$09B0	$0672	$09B0
default_error	$09DE	$06A0	$09DE
curr_sektor	$09C6	$0688	$09C6
curr_track	$09C4	$0686	$09C4
curr_side	$09C8	$068A	$09C8
curr_dma	$09CC	$068E	$09CC
spt	$09D4	$0696	$09D4
interleave	$09D6	$0698	$09D6
virgin	$09D8	$069A	$09D8

Name	neues TOS 196480 Bytes	altes TOS 207128 Bytes	ROM
motor_on_flag	$09BE	$0680	$09BE
flock	$043E	$043E	$043E
frclock	$0466	$0466	$0466
wp_st_tab	$09B2	$0674	$09B2
nflops	$04A6	$04A6	$04A6
wplatch	$09B4	$0676	$09B4
curr_device	$09C2	$0684	$09C2
acctim	$09B6	$0678	$09B6
_hz_200	$04BA	$04BA	$04BA
disknum	$5622	$4692	$5622
dm_tab	$4DB8	$3E2A	$4DB8
etv_critic	$0404	$0404	$0404
dct0	$0A06	$06C8	$0A06
dct1	$0A0A	$06CC	$0A0A

Kapitel 6

Ein FDC-Fehler

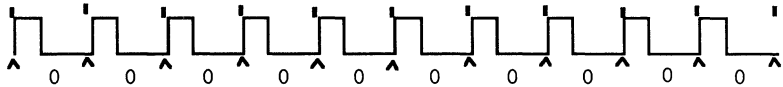
Eine besondere Eigenschaft des FDC ist ein Synchronisationsfehler, der "Hoppelfehler". Er tritt nur bei dem Read-Track-Kommando auf. Bei diesem Befehl werden bestimmte Bitfolgen als Synchronisationsmarkierungen interpretiert, die jedoch keine sind.

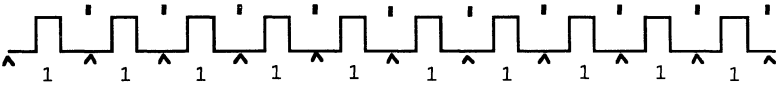
Dieses Phänomen hängt von der Art der Datenspeicherung, dem MFM-Verfahren, ab. Bei dem MFM-Verfahren kann normalerweise ein Datenbit drei verschiedene Zustände annehmen:

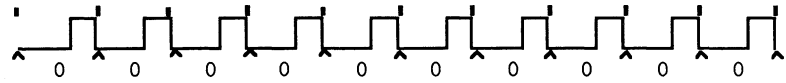
1. Das Bit ist gesetzt => 
   Das Bit ist gelöscht und
2. folgt nach einem Einsbit => 
3. folgt nach einem Nullbit => 

Um die Bedeutung der Synchronisation zu verdeutlichen, geben wir ein kleines Beispiel. Der Lesekopf liest folgende Drittelbitfolge:

Ohne Synchronisation kann die Folge folgendermaßen interpretiert werden:

1. 

2. 

3. 

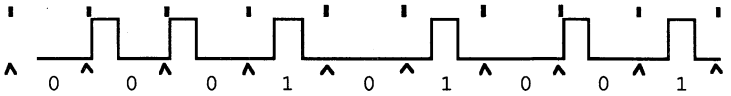
Die Lese-Elektronik muß also, wie man anhand des Beispiels leicht erkennt, synchronisiert werden. Das heißt ihr muß mitgeteilt werden, wann ein Bit beginnt und endet (im Beispiel sind die Bitgrenzen mit ^ gekennzeichnet).

Dafür werden bestimmte Bitfolgen als Marken definiert, die der Lese-Elektronik mitteilen, wann ein Bit beginnt. Als Sync-Markierungen sind beim FDC die Bytes $A1 mit fehlendem Takt zwischen Bit 4 und 5 und $C2 mit fehlendem Taktbit zwischen Bit 3 und 4 vorgesehen. Die Synchronisation erfolgt aber auch beim Lesen der Bitfolge 000101001 in folgenden Bytes:

1. ~~~~ ~~~0 0010 1001 = $29 mit vorherigem geraden Byte
2. ~~~~ ~~00 0101 001~ = $52/$53 mit dem vorherigen Byte ein Vielfaches von vier.
3. ~~~~ ~000 1010 01~~ = $A4 / $A5 / $A6 / $A7 mit dem vorherigen Byte ein Vielfaches von acht.

Da der ADDRESS MARK DETECTOR nur bei dem Read-Track nach den Sync-Markierungen sucht, tritt unter dem Read-Sector-Kommando der Fehler nicht auf!

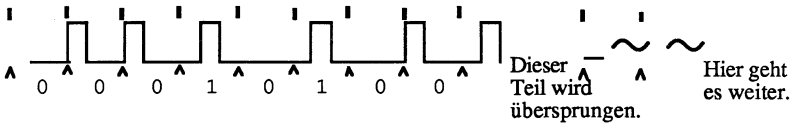
Betrachten wir das Byte $29 im MFM Format:



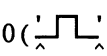
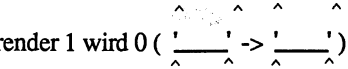
Diese Bitfolge erkennt der FDC als Adreßmarke.

Das letzte Bit des vorhergehenden Bytes und die ersten 7 Bits werden als nächstes Byte in das DATA-Register geschrieben (0001 0100 = $14).

Dann wird die Bytegrenze um zwei Drittel nach rechts verschoben (die nächsten zwei Drittel werden nicht erkannt).



Die folgenden Bytes erscheinen für den Lesekopf bis zur nächsten Sync-Marke um ein Drittelbit nach rechts verschoben.

- aus 1 wird 0 ()
- aus 0 mit führender 0 wird 1 ()
- aus 0 mit führender 1 wird 0 ()

Beim Byte $52/$53 tritt diegleiche Bitfolge ein Bit eher auf, was dazu führt, daß die folgenden Bytes um vier Drittel (genauer 1 Bit – da die Sync-Folge ein Bit früher auftritt – und 1 Drittel) nach rechts verschoben werden.

Beim Byte $A4/$A5/$A6/$A7 werden die folgenden Bytes um 7 Drittel (2 Bit + 1 Drittel) nach rechts verschoben.

Um zu zeigen, wie groß die Veränderung der Bytes wird, geben wir zwei Beispiele:

Beispiel 1:

```
Original: 00 29 00 01 02 04 08 10 20 40 80 00
gelesen:  00 14 7F FE 7C F9 F3 F7 CF 9F 3F FF
```

Beispiel 2:

```
Original: 02 90 00 01 02 04 08 10 20 40 80 00
gelesen:  02 14 7F FF E7 CF 9F 3F 7C F9 F3 FF
```

Um sicherzugehen, daß die Daten korrekt gelesen werden, sollte daher auf den Gebrauch des Read-Track-Kommandos verzichtet werden.

Kapitel 7

Die Powerdisk

Die Powerdisk ist eine Beilage zum Floppy-Arbeitsbuch. Sie enthält Programme, die man im täglichen Umgang mit der Floppy des ATARI ST gut gebrauchen kann.

Einen komfortablen Diskmonitor, ein sehr gutes Filecopy, zwei schnelle und benutzerfreundliche Kopierprogramme und ein Programm zur Untersuchung und Manipulation der Diskette.

Außerdem sind Beispielprogramme und Source-Listings aus dem Buch auf der Powerdisk zu finden.

Wenn Sie nun das Inhaltsverzeichnis der Diskette angeklickt haben, so werden Sie zwei Ordner und ein Programm erkennen können. In dem Ordner *Beispiele* finden Sie die Programme und Source-Listings aus dem Floppy-Arbeitsbuch.

In dem Ordner Powerdisk sind die zur *Powerdisk* gehörenden Programme zu finden.

Bitte klicken Sie nun das Programm *Power* an. Nun wird das Hauptmenü der Powerdisk geladen. Sie haben folgende Auswahlmöglichkeiten:

```
Copy-Menü:
---------

- Filecopy
- Speedcopy
- Excopy
- Ende

Spezial-Menü:
------------
- Diskmon
- The Clone
```

Möchten Sie Dateien kopieren, so klicken Sie bitte *Filecopy* an. Möchten Sie normale Disketten kopieren, wählen Sie bitte *Speedcopy*. Möchten Sie mit größerer Kapazität formatierte Disketten kopieren, so nehmen Sie *Excopy*.

Der Diskmonitor erlaubt Ihnen, Ihre Diskette unter die Lupe zu nehmen und *The Clone* ist zur Manipulation und Untersuchung der Diskette zu benutzen.

Um die Powerdisk zu beenden, wählen Sie bitte *Ende*.

Filecopy V2.0

Mit Hilfe dieses Programms können Sie ein oder mehrere Files kopieren, umbenennen oder löschen.

Sie können außerdem Ordner erstellen, umbenennen, den Diskettennamen ändern und eine Diskette formatieren.

Haben Sie nun den Punkt *Filecopy* im Hauptmenu angeklickt, so haben Sie nach dem Ladevorgang folgende Auswahlmöglichkeiten:

```
File-Menü                    Laufwerk-Menü
---------                    -------------
- Files kopieren                   A
- Files umbenennen                 B
- Files löschen
- Diskette formatieren
- Directory
- Ende

Spezial-Menü
------------
- Ordner umbenennen
- Ordner erstellen
- Diskettennamen ändern
```

1. Files kopieren

Haben Sie diesen Punkt angewählt, so erscheint das Filecopy-Window, und die Directory wird vom Laufwerk A eingelesen.

Nun können Sie auswählen, mit welchen Laufwerken Sie arbeiten möchten. Klicken Sie nun im Punkt Quelldisk das Laufwerk B an, so wird das Inhaltsverzeichnis von Drive B ausgegeben.

Nachdem Sie die Laufwerkkonfiguration ausgewählt haben, klicken Sie bitte die gewünschten Dateien im Fenster an. Diese werden dann fett unterlegt und mit einem Häkchen versehen.

Sind mehr Dateien auf der Diskette, als in das Window passen, so können Sie den Slide-Bar an der rechten Seite des Fensters verschieben. Ausgewählte Dateien bleiben angeklickt.

Ordner werden *outlined* dargestellt und können *nicht* kopiert werden! Die Dateien, die sich in einem Ordner befinden, erscheinen unter dem Ordnernamen. Haben Sie nun alle gewünschten Dateien ausgewählt, so klicken Sie bitte den COPY-Button an.

Sie werden dann aufgefordert, die Disketten in die entsprechenden Laufwerke einzulegen. Drücken Sie nun die Return-Taste, oder klicken Sie den OK-Button an.

Jetzt können Sie auswählen, ob Sie in einen Ordner kopieren wollen oder in das Haupt-Directory. Möchten Sie in einen Ordner kopieren, so erscheint ein Fenster, das die auf der Zieldiskette existierenden Ordner anzeigt. Den gewünschten Ordner können Sie durch Anklicken auswählen. Schließen Sie dann bitte das Fenster. Nun beginnt der Kopiervorgang.

2. Files umbenennen

Haben Sie den Menüpunkt "Files umbenennen" angewählt, so erscheint das Inhaltsverzeichnis der Diskette. Jetzt können Sie die Datei, die umbenannt werden soll, durch Anklicken auswählen. Beachten Sie dabei aber, daß nur eine Datei selektiert sein kann. Durch Drücken des rechten Mausbuttons oder Anklicken der Close-Box verlassen Sie dieses Fenster.

Falls eine Datei angeklickt ist, erscheint nun ein Eingabefenster, in das Sie den neuen Namen der Datei eingeben können. Durch Anklicken der Exit-Box können Sie diesen Menüpunkt verlassen, ohne die Datei umzubenennen. Wenn Sie dieses Fenster durch die Rename-Box verlassen, wird die Datei umbenannt.

3. Files löschen

Möchten Sie Dateien löschen, wählen Sie bitte den Punkt "Files löschen" aus. In dem nun erscheinenden Fenster können Sie die Parameter "Laufwerk" und Sicherheit" einstellen. Wenn "Sicherheit AN" ist, dann wird der Name einer Datei angezeigt, bevor sie gelöscht wird, und auf eine Bestätigung gewartet.

Verlassen Sie dieses Fenster durch EXIT, wenn Sie keine Dateien löschen wollen.

Nach Anklicken der ERASE-Box wählen Sie die Datei durch Anklicken. Die angeklickten Dateien werden gelöscht, sobald Sie das Directory Window durch Drücken der rechten Maustaste verlassen.

4. Directory

Wenn Sie das Inhaltsverzeichnis sehen möchten, so wählen Sie bitte den Punkt "Directory" an. Beachten Sie bitte, daß sich in dem angegebenen Laufwerk auch eine Diskette befindet.

Es öffnet sich nun ein Fenster, das Ihnen folgendes anzeigt:

- den Diskettennamen (in der Fensterleiste)
- die Anzahl der freien und belegten Bytes (in der Infozeile)
- die Ordner und Programme, die sich auf der Disk befinden

Die Programme werden alphabetisch sortiert ausgegeben, Ordner kommen wie im Desktop zuerst. Sie sehen nun den Namen des Programms, das Attribut, die Dateilänge sowie Datum und Uhrzeit der Erstellung.

Falls sich mehr Programme auf der Diskette befinden, als in das Fenster passen, können Sie den Slide-Bar an der rechten Seite des Fensters verschieben.

Möchten Sie sehen, welche Programme sich in einem Ordner befinden, so klicken Sie diesen bitte einmal an.

Um den Ordner zu verlassen, schließen Sie bitte das Fenster. Dann befinden Sie sich wieder im Haupt-Directory. Durch Drücken der ESC-Taste (Escape) wird das Directory neu eingelesen.

Zum Verlassen des Directory-Fensters können Sie die rechte Maustaste drükken oder das Fenster schließen.

5. Ordner erstellen

Wählen Sie dann das Sub- oder Hauptdirectory, in dem der Ordner erstellt werden soll. Beenden Sie Ihre Auswahl durch Drücken der rechten Maustaste. Danach geben Sie den Namen des Ordners ein. Dieser wird dann auf der Diskette erstellt.

6. Diskettennamen ändern

Dieser Menüpunkt erlaubt Ihnen, den Namen der Diskette zu ändern. Geben Sie den neuen Diskettennamen ein, der mit RENAME auf der Diskette erzeugt oder, falls schon vorhanden, geändert wird.

7. Ordner umbenennen

Zuerst erscheint ein DIRECTORY-Fenster, in dem Sie den Ordner auswählen. Der momentane Ordnername erscheint in der INFO-Zeile des Fensters.

Durch Drücken der rechten Maustaste wird der Name dann in ein DIALOG-Fenster übernommen, in dem Sie dann den neuen Namen eingeben.

8. Laufwerk auswählen

Die Bestimmung des aktuellen Laufwerks erfolgt im Auswahlmenü in der Kopfleiste unter "LAUFWERK".

Speedcopy V2.0

Mit diesem Kopierprogramm können Sie Ihre Disketten schnell, komfortabel und sicher kopieren.

Dies ist vor allem nötig für das Erstellen von Sicherungsduplikaten von wichtigen Disketten. Es ermöglicht Ihnen außerdem, defekte Diskettten zu kopieren, um wenigstens die noch funktionierenden Programme benutzen zu können.

Dieses Kopierprogramm wurde entwickelt, da das eingebaute Desktop-Copy einige Wünsche offenläßt und manchmal auch nicht zur vollen Zufriedenheit arbeitet.

Speedcopy V2.0 eignet sich nicht zum Kopieren geschützter Software. Es lassen sich nur Backups von "normalen" Disketten erstellen.

Nachdem Sie im Hauptmenü den Punkt Speedcopy angeklickt haben, erscheint nach dem Ladevorgang die Copyright-Meldung des Programms.

Durch Drücken der Return-Taste gelangen Sie in das Hauptmenü des Copys.

Sie sehen nun den Bildschirm in vier Fenster aufgeteilt. Das Fenster links oben, das COPY-Fenster, bietet die Auswahl zwischen dem Kopieren einer einseitigen Diskette (SINGLE SIDE COPY) und dem Kopieren einer zweiseitigen Diskette (DOUBLE SIDE COPY). Außerdem haben Sie die Möglichkeit, sich mitteilen zu lassen, ob Ihre Diskette eventuelle Fehler aufweist.

Das zweite Fenster rechts oben, das CATALOG-Fenster, ermöglicht es Ihnen, das Inhaltsverzeichnis Ihrer Diskette anzeigen zu lassen.

In dem Fenster links unten, dem OPTIONS-Fenster, wählen Sie die Laufwerkkonfiguration aus, stellen ein, ob die Zieldiskette formatiert wird und ob die Multi-Option (Mehrfachkopien) aktiviert sein soll.

Das vierte Fenster, das FAST PARAMETER-Fenster, bietet nun die Einstellmöglichkeiten, ob Sie schnell lesen, schreiben oder formatieren möchten.

1. Das COPY-Fenster

Wenn Sie nun eine einseitige Diskette kopieren möchten, klicken Sie bitte den entsprechenden Button an. Um eine zweiseitige Diskette zu kopieren, klicken Sie bitte diesen Button an.

Wollen Sie sich mitteilen lassen, ob Fehler auf der Diskette vorhanden sind, so klicken Sie bitte "Fehler ignorieren AUS" an. Jetzt wird bei einem auftretenden Fehler eine Meldung ausgegeben, und der Kopiervorgang wird abgebrochen.

Möchten Sie eine defekte, fehlerhafte Diskette kopieren, wählen Sie bitte "Fehler ignorieren AN" aus. Dann werden eventuelle Fehler auf der Diskette überlesen, aber sie werden nicht kopiert. Am Ende des Kopiervorgangs wird dann, falls Fehler aufgetreten sind, ein kurze Meldung ausgegeben.

2. Das CATALOG-Fenster

Wenn Sie das Inhaltsverzeichnis sehen möchten, so wählen Sie bitte das gewünschte Laufwerk aus und klicken dann die Box mit dem ATARI-Symbol an. Beachten Sie bitte, daß sich in dem angegebenen Laufwerk auch eine Diskette befindet.

Es öffnet sich nun ein Fenster, das Ihnen folgendes anzeigt:

- den Diskettennamen (in der Fensterleiste)
- die Anzahl der freien und belegten Bytes (in der Infozeile)
- die Ordner und Programme, die sich auf der Disk befinden.

Die Programme werden alphabetisch sortiert ausgegeben. Ordner kommen wie im Desktop zuerst. Sie sehen nun den Namen des Programms, das Attribut, die Dateilänge sowie Datum und Uhrzeit der Erstellung.

Falls sich mehr Programme auf der Diskette befinden, als in das Fenster passen, können Sie den Slide-Bar an der rechten Seite des Fensters verschieben.

Möchten Sie sehen, welche Programme sich in einem Ordner befinden, so klicken Sie diesen bitte einmal an.

Um den Ordner zu verlassen, schließen Sie bitte das Fenster. Dann befinden Sie sich wieder im Haupt-Directory.

Durch Drücken der ESC-Taste (Escape) wird das Directory neu eingelesen.

Zum Verlassen des Directory-Fensters können Sie die rechte Maustaste drükken oder das Fenster schließen.

Hinweis: Bei der Anzeige der freien und belegten Bytes kommt es zu Abweichungen gegenüber der Anzeige des Desktops, da das Desktop den Platz von Ordnern und den darin enthaltenen Dateien im Haupt-Directory nicht mitberechnet. Das Desktop zeigt immer nur den Platzbedarf an, der sich aus den Größen der einzelnen Dateien addiert.

3. Das OPTIONS-Fenster

Hier wählen Sie, mit welchen Laufwerken Sie arbeiten möchten, ob die Zieldiskette formatiert werden soll und ob die Multi-Option (Mehrfachkopien) eingeschaltet werden soll.

Klicken Sie dazu bitte die gewünschten Felder an.

4. Das FAST PARAMETER-Fenster

In diesem Fenster können Sie einstellen, ob Sie schnell lesen, schreiben und formatieren wollen. Sie können auch einzelne Parameter ausschalten z.B. "Schnell Schreiben" auf AUS.

Diese Parameter verdoppeln die Geschwindigkeit der Floppy-Routinen. Eine komplette einseitige Diskette wird, falls genügend Speicher vorhanden ist, in ca. 36 Sekunden kopiert (lesen und schreiben).

Falls Probleme auftreten, sei es beim Lesen, Schreiben oder Formatieren, so schalten Sie bitte die Parameter auf AUS.

Es können vor allem beim Schreiben Probleme auftreten, zu 99% aber nur bei Fremdlaufwerken, die eine langsamere Step-Rate als die ATARI-Laufwerke besitzen und deshalb diese Geschwindigkeit nicht richtig verkraften.

5. Der Kopiervorgang

Haben Sie nun entweder das Feld für SINGLE SIDE COPY oder DOUBLE SIDE COPY angeklickt, so wird dieses Menü verlassen, und es erscheint ein Fenster, das Sie auffordert, die Quelldiskette einzulegen, wenn Sie nur mit einem Laufwerk kopieren oder wenn Sie mit zwei Laufwerken kopieren wollen, werden Sie aufgefordert, beide Disketten in die Laufwerke einzulegen. Drücken Sie die Return-Taste, um den Kopiervorgang zu starten, oder klicken Sie mit der Maus das Feld ABBRUCH an, um wieder ins Menü zu gelangen.

Wurde der Kopiervorgang gestartet, sehen Sie ein Fenster, das Ihnen anzeigt, auf welchem Track die Floppy gerade arbeitet.

Nach dem Einlesen der Diskette werden Sie aufgefordert, die Zieldiskette einzulegen (nur beim Kopieren mit einem Laufwerk).

Haben Sie die Multi-Option eingeschaltet, so erscheint dieses Fenster so lange, bis Sie genug Kopien gemacht haben. Dann müssen Sie ABBRUCH anklicken, um das Schreiben der Daten zu beenden.

Falls die Diskette nicht ganz eingelesen wurde, erscheint wieder die Aufforderung, die Quelldiskette einzulegen.

Die Prozedur wiederholt sich.

Fehlermeldungen:

Folgende Fehlermeldungen können auftreten:

- `Dies ist keine einseitige Disk`

Sie haben versucht, eine zweiseitige Diskette mit dem SINGLE SIDE COPY zu kopieren.

- `Dies ist keine zweiseitige Disk`
Sie haben versucht, eine einseitige Diskette mit dem DOUBLE SIDE COPY zu kopieren.

- `Dies ist keine zweiseitige Floppy`
Sie haben versucht, eine zweiseitige Diskette auf einem einseitigen Laufwerk zu kopieren.

- `Drive not ready`
Ihr Laufwerk reagiert nicht. Überprüfen Sie gegebenfalls die Stecker und die Stromversorgung.

- `Track nicht gefunden`
Dies ist ein Fehler auf der Diskette. Der Track ist nicht formatiert oder nicht vorhanden.

- `Sektor nicht gefunden`
Ebenfalls ein Fehler auf der Diskette.

- `Lesefehler`
Der Track kann nicht korrekt eingelesen werden (Dies deutet meistens auf einen Fehler im Header des Tracks).

- `Schreibfehler`
Es wurde versucht, auf einen fehlerhaften Track oder Sektor zu schreiben.

- `Disk ist schreibgeschützt`
Bitte den Schreibschutz entfernen.

- `Allgemeiner Fehler`
Systemfehler.

Excopy V1.0

Mit Excopy können Sie mit Extended-Format formatierte Diskette kopieren. Extended-Format bedeutet, daß die Diskette mit 82 Tracks und 10 Sektoren pro Track formatiert ist.

Als einzige Änderung gegenüber dem Speedcopy kann man mit diesem Copy das Directory nicht einlesen, sondern Sie können sich stattdessen das Format der Diskette anzeigen lassen. Dazu wählen Sie bitte das Laufwerk und klicken dann die Box mit dem ATARI-Symbol an.

Diskmonitor

Wenn Sie aus dem SPECIAL-Menü den "DISKMON" gewählt haben, so wird der Disketten-Monitor und das dazugehörende Resource-File nachgeladen. Kurz darauf befinden Sie sich im View Disk-Modus.

Der Bildschirm ist in drei Teile unterteilt, der hexadezimalen und der ASCII-Anzeige sowie der Anzeige für Spur, Sektor, Seite und Laufwerk (Drive).

Der Diskmonitor lädt nun den Sektor 3 von der Seite 0 auf Spur 1 in den Rechner und zeigt diesen dann auf dem Bildschirm an. Sie können nun das Directory der POWERDISK erkennen.

Der Diskmonitor versteht folgende Kommandos:

_ (Space) Der aktuelle Sektor wird noch einmal eingelesen.

r (Read) Es erscheint nun ein Fenster, in dem Sie die aktuellen Parameter ändern können. Diese sind Spur, Sektor, Seite, Laufwerk, die maximale Spur und der maximale Sektor.

\+ (Plus) Der Sektor wird um eins erhöht. Ist der Sektor größer als der maximale Sektor, so wird der Sektor 1 auf der nächsten Spur angesprungen.

\- (Minus) Der Sektor wird um eins erniedrigt. Wenn der Sektor dann kleiner gleich null ist, wird der maximale Sektor auf der vorherigen Spur angesprungen.

↑ (Cursor Up) Die Spur wird um eins erhöht. Falls die Spur größer als die maximale Spur ist, so wird auf die Spur 0 gesprungen.

↓ (Cursor down) Die Spur wird um eins erniedrigt. Ist die Spur kleiner als null, so wird auf die maximale Spur gesprungen.

s (Side) Dieses Kommando wechselt die Seite.

d (Drive) Dieses Kommando wechselt das Laufwerk. Es können Laufwerk A oder B angewählt werden.

f (Format) Mit dieser Routine können Sie eine Spur mit beliebiger Sektorenanzahl (maximal 11) formatieren. Der von ATARI empfohlene Füllwert zum Formatieren ist $E5.

F (Fill) Damit füllen Sie den im Speicher befindlichen Block. Dieser kann dann mit dem W-Kommando auf die Diskette geschrieben werden.

w (Write) Dieser Befehl schreibt den im Speicher befindlichen Block auf die Diskette. Sie können hier noch Spur und Sektor wählen.

e (Edit) Mit diesem Kommando gelangen Sie in den Editier-Modus. Sie können nun im Hexadezimal-Feld oder im ASCII-Feld die Daten durch Überschreiben ändern. Verlassen Sie den Edit-Modus mit RETURN oder ENTER, so werden die geänderten Daten in den Speicher übernommen, und Sie können den Block dann mit dem W-Kommando auf die Disk schreiben. Mit dem Verlassen über die UNDO-Taste wird der alte Block wieder angezeigt.

b (Boot) Dieser Befehl ermöglicht die Erstellung einer "boot"-fähigen Diskette. Wenn Sie bei COMMAND.PRG das JA-Feld wählen, so wird bei einem Booten der Diskette das COMMAND.PRG nachgeladen. Unter der Seriennummer können Sie eine beliebige hexadezimale Zahl eingeben. Ist diese größer oder gleich $01000000, so wird eine zufällige Zahl auf die Diskette geschrieben.

Zum Erstellen einer doppelseitigen Systemdiskette gehen Sie folgende Schritte durch:

1. Formatieren Sie Ihre Diskette zweiseitig mit 80 Spuren und 9 Sektoren je Spur.

2. Kopieren Sie von der einseitigen Sytemdiskette die Datei TOS.IMG auf Ihre Diskette.

3. Starten Sie nun den Diskmonitor.

4. Laden Sie den Bootsektor (Spur 0, Sektor 1) von der einseitigen Systemdiskette in den Speicher.

5. Legen Sie Ihre doppelseitige Diskette in das Laufwerk und schreiben den Block mit dem W-Kommando auf Spur 0, Sektor 1, Seite 0.

6. Drücken Sie die b-Taste und wählen dann den zweiseitigen Typ aus.

7. Wenn Sie nun das OK–Feld anklicken und Ihre Diskette nicht schreibgeschützt ist, erstellt der Diskmonitor eine doppelseitige Systemdiskette.

S (Search) Es erscheint ein Fenster, in dem Sie den zu suchenden String und die Suchrichtung eingeben können. Fängt der String mit $ an, so wird die-

ser als Hexadezimalzahl interpretiert. Diese muß in jedem Fall eine gerade Anzahl von Stellen aufweisen (z.B. $12AB). Ist die Anzahl ungerade, so wird die letzte Stelle bei der Suche ignoriert.

Die Suche erstreckt sich je nach Richtung von der aktuellen Position über den maximalen Sektor und die maximale Spur, bis entweder der String gefunden oder in die Ausgangsposition zurückgekehrt wurde.

Ist der String gefunden, so können Sie mit der r-Taste die Suche fortsetzen oder mit der e-Taste den gefundenen String editieren.

Der Suchvorgang kann über einen Tastendruck abgebrochen werden.

p (Print) Falls Sie den Inhalt Ihrer Disketten auch auf dem Papier haben wollen, können Sie mit diesem Befehl den aktuellen Block entweder über die parallele (Drucker) oder serielle (Modem) Schnittstelle ausgeben lassen. Mit einem Tastendruck kann der Ausdruck abgebrochen werden. Der Ausdruck hat folgendes Aussehen:

```
Track: xx   Sector: xx
0000: 41 42 43 44 45 46 47 ..... 77 78 79 7A : ABCDEFG . wxyz
0010: 30 31 32 33 34 ............. 3C 3D 3E : 01234 .... <=>

 ...        ...        ...

01F0: 61 62 63 64 ..................... 6F : abcd ....... o
```

v (View File) Mit diesem Kommando gelangen Sie in den View File-Modus. Hier können Sie eine beliebige Datei von einem beliebigen Laufwerk (Floppy, Harddisk, RAM-Disk) auswählen. Danach wird diese Datei geöffnet, und die ersten 512 Bytes werden auf dem Bildschirm angezeigt. Die Anzeige für Spur, Sektor, Seite und Laufwerk ist verschwunden, stattdessen erscheint nun der Name der Datei und die Position des Blocks.

q, x und UNDO beenden den Diskmon.

Im View File-Modus existieren folgende Kommandos:

\+ Die nächsten 512 Bytes der Datei einlesen.

\- Die vorherigen 512 Bytes einlesen.

→ An das Ende der Datei gehen.

← An den Anfang der Datei gehen.

g An eine beliebige Position innerhalb der Datei gehen.

x Datei schließen und neue Datei auswählen.

e Block editieren (wie unter View Disk). Wird der Edit-Modus mit RETURN oder ENTER verlassen, erscheint eine Abfrage, ob der geänderte Block auf die Diskette zurückgeschrieben werden soll. Wenn Sie diese Frage mit JA beantworten und die Datei den read only-Status hat, so erhalten Sie eine Fehlermeldung. Eine Datei mit read only-Status kann nicht editiert werden!

s Durchsuchen der Datei nach einem beliebigen String. Es gilt hier das gleiche wie unter View Disk, nur wird hier nicht über die Sektoren gesucht, sondern über den File-Pointer.

p Auch eine Datei können Sie zu Papier bringen. Es wird der angezeigte Block ausgedruckt, per Tastendruck können Sie den Vorgang abbrechen. Eine genauere Beschreibung des p-Kommandos wurde bereits unter View Disk gegeben. Mit q oder UNDO kehren Sie in den View Disk-Modus zurück.

The Clone

Dieses Programm erlaubt dem Benutzer vielfältige Möglichkeiten der Manipulation und des Untersuchens von Disketten.

Es können ein oder mehrere Tracks formatiert werden, ein Track kann kopiert werden, Tracks können nach Fehlern untersucht werden, eine ganze Diskette kann auf Fehler überprüft und nach Wahl formatiert werden.

Nach dem Ladevorgang haben Sie folgendes Auswahlmenü:

```
- Format Track
- Copy Track
- Scan Track
- Scan Disk
- Format Disk
- Ende
```

1. Format Track

Mit diesem Befehl haben Sie die Möglichkeit, Tracks zu formatieren und dabei alle Parameter nach Wahl einzustellen.

Die Parameter dürfen dabei folgende Werte annehmen:

Starttrack: von 0 bis 83
Endtrack: von 0 bis 83

Der Endtrack darf nie kleiner als der Starttrack sein.

Anzahl der Sektoren:	von 1 bis 11
Interleave:	von 1 bis 11
Data:	Zahlen von 0 bis 9 und Buchstaben von a bis f bzw. A bis F.
Laufwerk:	A oder B
Seite:	0 oder 1

Nach Eingabe der Parameter müssen Sie den Formatieren-Button anklicken.

Nun werden Sie aufgefordert, die Diskette einzulegen. Drücken Sie Return, um den Formatiervorgang zu starten, oder klicken Sie mit der Maus auf den Stop-Button, um abzubrechen.

2. Copy Track

Damit können Sie einen Track von einer Spur und Diskette auf eine beliebige andere Spur und Diskette kopieren.

Es stehen Ihnen folgende Parameter zur Verfügung:

Lese Track:	von 0 bis 83
Schreibe Track:	von 0 bis 83
Startsektor:	von 0 bis 11
Anzahl der Sektoren:	von 1 bis 11

Bitte beachten Sie, daß die Summe von Startsektor und Anzahl der Sektoren den Wert 12 nicht übersteigt, sonst erhalten Sie eine Fehlermeldung.

Quelldisk: A oder B
Zieldisk: A oder B
Seite: 0 oder 1

Bitte klicken Sie nun den Kopieren-Button an. Sie werden dann aufgefordert, die Quelldiskette einzulegen. Wurde der Track korrekt eingelesen, erscheint die Meldung, die Zieldiskette einzulegen.

Drücken Sie nun ebenfalls auf OK.

3. Scan Track

Diese Funktion erlaubt, Tracks und Sektoren auf Fehler und auf Vorhandensein zu untersuchen.

Folgende Parameter stehen zur Verfügung:

Starttrack: von 0 bis 83
Endtrack: von 0 bis 83

Der Endtrack darf nicht kleiner als der Starttrack sein.

Startsektor: von 0 bis 11
Endsektor: von 0 bis 11

Der Endsektor darf nicht kleiner als der Startsektor sein.

Laufwerk: A oder B
Seite: 0 oder 1

Wählen Sie nun den Scan-Button, um den Vorgang zu starten.

In der unteren Zeile wird dann angezeigt, auf welchem Track und Sektor gerade gelesen wird. Tritt ein Fehler auf, so wird eine Meldung ausgegeben. Drücken Sie bitte die Return-Taste. Nun können Sie wählen, ob Sie weitermachen wollen oder ob Sie unterbrechen wollen.

Klicken Sie dazu bitte den entsprechenden Button an. Am Ende des Scan-Vorgangs oder nach Abbruch drücken Sie bitte eine Taste.

4. Scan Disk

Hiermit wird eine ganze Diskette nach Fehlern durchsucht. Auf dem Bildschirm wird angezeigt, ob ein Sektor belegt, frei oder fehlerhaft ist.

Treten beim Lesen Fehler auf, so wird dies durch ein Fragezeichen (?) angezeigt. Ein belegter Sektor wird durch ein Pluszeichen (+), ein freier Sektor durch ein Minuszeichen (-) angezeigt.

Am Ende des Scan-Vorgangs müssen Sie eine Taste drücken.

5. Format Disk

Sie können eine Diskette mit 40, 80 oder 82 Tracks einseitig oder zweiseitig formatieren. Bitte legen Sie eine nicht schreibgeschützte Diskette in das gewählte Laufwerk und klicken den Format-Button an.

Um dieses Menu zu verlassen, wählen Sie *Ende*.

Außer im Punkt "Format Disk" kann überall das Directory eingelesen werden.

Im Farbmodus muß beachtet werden, daß die Programme nur in der mittleren Auflösung funktionieren.

Bei *The Clone* sind in Farbe die angegebenen Buttons gesetzt:

– Format Track	Die hellen Buttons sind angewählt.
– Copy Track	Die dunklen Buttons sind angewählt.
– Scan Track	Die dunklen Buttons sind angewählt.
– Disk Scan	Die hellen Buttons sind angewählt.
– Format Disk	Die roten Buttons sind angewählt.

Anhang A

Die Fehlermeldungen des TOS

Nummer	Abkürzung	Bedeutung
0	Ok	Kein Fehler
-1	Error	Fundamentaler Fehler
-2	Device Not Ready	Gerät nicht betriebsbereit
-3	Unknown Cmd	Unbekannter Befehl
-4	CRC Error	Sektor nicht lesbar
-5	Bad Request	Falscher Befehlstext
-6	Seek Error	Spur wurde nicht gefunden
-7	Unknown Media	Falsches Diskettenformat
-8	Sector Not Found	Sektor wurde nicht gefunden
-9	No Paper	Kein Papier im Drucker
-10	Write Fault	Allgemeiner Fehler beim Schreiben
-11	Read Fault	Allgemeiner Fehler beim Lesen
-12	General Error	Allgemeiner Fehler
-13	Write Protect	Diskette ist schreibgeschützt
-14	Media Change	Diskette wurde gewechselt
-15	Unknown Device	Gerät ist unbekannt
-16	Bad Sectors Defekte	Sektoren beim Formatieren
-17	Insert Disk	Diskette einlegen
-32	Invalid Function	Ungültige Funktionsnummer
-33	File Not Found	Datei wurde nicht gefunden
-34	Path Not Found	Ordner wurde nicht gefunden
-35	No Handle Left	Zu viele Dateien sind offen
-36	Access Denied	Zugriff nicht möglich
-37	Invalid Handle	Ungültiges Handle
-39	Insufficient Memory	Zu wenig Speicherplatz
-40	Invalid Memory-Block Address	Ungültige Speicherblockadresse
-46	Invalid Drive	Ungültiges Laufwerk
-49	No More Files	Keine weiteren Dateien möglich

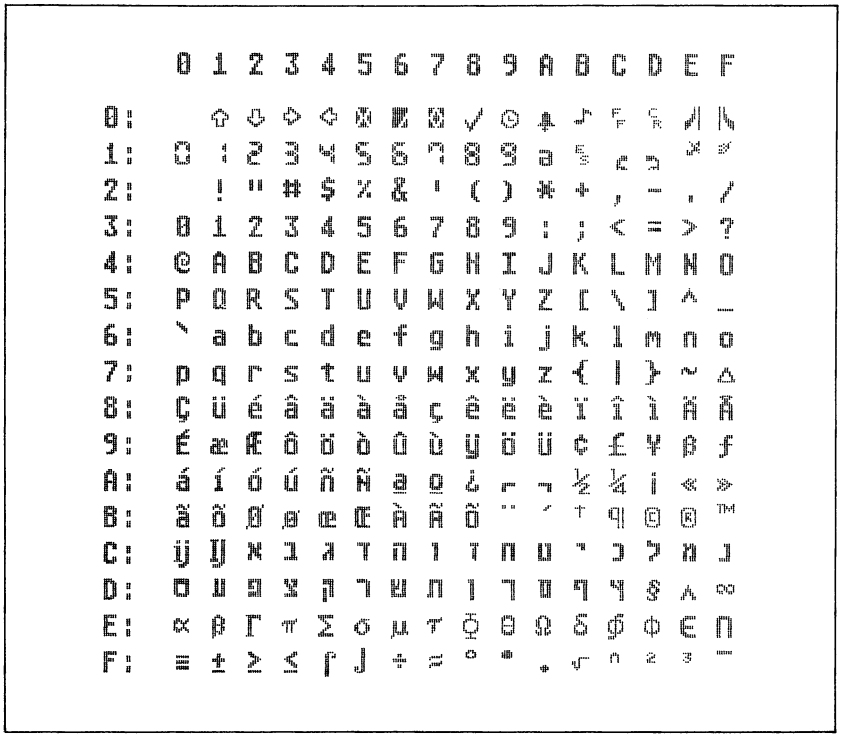
Anhang B

Der Bootsektor

$00 - $01	BRA	Sprung zum Ladeprogramm
$02 - $07	FILLER	Diskettenbezeichnung
$08 - $0A	SERIAL	Seriennummer
$0B - $0C	BPS	Bytes je Sektor
$0D	SPC	Sektoren je Gruppe
$0E - $0F	RES	Reservierte Sektoren
$10	NFATS	Anzahl der FATS
$11 - $12	NDIRS	Directory-Einträge
$13 - $14	NSECTS	Anzahl der Sektoren
$15	MEDIA	Diskettenformat
$16 - $17	SPF	Sektoren je FAT
$18 - $19	SPT	Sektoren je Spur
$1A - $1B	NSIDES	Anzahl der Seiten
$1C - $1D	NHID	Versteckte Sektoren
$1E - $1FF	BOOTER	Ladeprogramm

Anhang C

Der ATARI-Zeichensatz



Anhang D

Die Scan-Codes der Tastatur

	0	1	2	3	4	5	6	7	8	9
0		ESC	1	2	3	4	5	6	7	8
1	9	0	ß	'	BSP	TAB	Q	W	E	R
2	T	Z	U	I	O	P	Ü	+	RET	
3	A	S	D	F	G	H	J	K	L	Ö
4	Ä	#	SHL	~	Y	X	C	V	B	N
5	M	,	.	-	SHR		ALT	SPC	CPS	F1
6	F2	F3	F4	F5	F6	F7	F8	F9	F10	
7		CLR	↑	[-]	←		→	[+]		
8			INS	DEL						
9							<	UNDO	HELP	[(]
10	[)]	[/]	[*]	[7]	[8]	[9]	[4]	[5]	[6]	[1]
11	[2]	[3]	[0]	[.]	ENT					

Die Abkürzungen bedeuten:

BSP	–	Backspace
RET	–	Return
SHL	–	Shift links
SHR	–	Shift rechts
ALT	–	Alternate
SPC	–	Space, Leertaste
CPS	–	Caps Lock
F1/F10	–	Funktionstasten
CLR	–	Clr/Home
INS	–	Insert
DEL	–	Delete
ENT	–	Enter

In eckige Klammern eingeschlossene Tastatursymbole sind auf der10er-Tastatur zu finden.

Anhang E

Die GEMDOS-Funktionen

$00	Pterm0()
$01	Cconin()
$02	Cconout(char)
$03	Cauxin()
$04	Cauxout(char)
$05	Cprnout(char)
$06	Crawio(word)
$07	Crawcin()
$08	Cnecin()
$09	Cconws(string)
$0A	Cconrs(buffer)
$0B	Cconis()
$0E	Dsetdrv(drive)
$10	Cconos()
$11	Cprnos()
$12	Cauxis()
$13	Cauxos()
$19	Dgetdrv()
$1A	Fsetdta(pointer)
$20	Super(stack)
$2A	Tgetdate()
$2B	Tsetdate(date)
$2C	Tgettime()
$2D	Tsettime(time)
$2F	Fgetdta()
$30	Sversion()
$31	Ptermres(keep, ret)
$36	Dfree(buffer, drive)
$39	Dcreate(path)
$3A	Ddelete(path)
$3B	Dsetpath(path)
$3C	Fcreate(name, attribute)
$3D	Fopen(name, mode)

$3E	Fclose(handle)
$3F	Fread(handle, count, buffer)
$40	Fwrite(handle, count, buffer)
$41	Fdelete(name)
$42	Fseek(offset, handle, mode)
$43	Fattrib(path, mode, attribute)
$45	Fdup(stdhandle)
$46	Fforce(stdhandle, handle)
$47	Dgetpath(pathbuffer, drive)
$48	Malloc(amount)
$49	Mfree(address)
$4A	Mshrink(zero, memory, size)
$4B	Pexec(mode, path, cmdline, environment)
$4C	Pterm(code)
$4E	Fsfirst(spec, attribute)
$4F	Fsnext()
$56	Frename(zero, oldname, newname)
$57	Fdatime(handle, buffer, set)

Anhang F

Die BIOS-Funktionen

$00	Getmpb(p_mpb)
$01	Bconstat(device)
$02	Bconin(device)
$03	Bconout(device, char)
$04	Rwabs(flag, buffer, count, recno, device)
$05	Setexc(vecnum, vector)
$06	Tickcal()
$07	Getbpb(device)
$08	Bcostat(device)
$09	Mediach(device)
$10	Drvmap()
$11	Kbshift(mode)

Anhang G

Die XBIOS-Funktionen

$00	Initmous(type, params, vector)
$01	Ssbrk(amount)
$02	Physbase()
$03	Logbase()
$04	Getrez()
$05	Setscreen(logLoc, physLoc, resol)
$06	Setpallete(palpointer)
$07	Setcolor(colnumber, color)
$08	Floprd(buf,filler,dev,sectno,trkno,side,count)
$09	Flopwr(buf,filler,dev,sectno,trkno,side,count)
$0A	Flopfmt(buf,fil,dev,spt,trkno,side,inter,mag,vir)
$0C	Midiws(count, pointer)
$0D	Mfpint(interno, vector)
$0E	Iorec(device)
$0F	Rsconf(speed, flowctl, ucr, rsr, tst, scr)
$10	Keybtl(unshift, shift, capslock)
$11	Random()
$12	Protobt(buffer, serial, dtype, execflag)
$13	Flopver(buf,filler,dev,sectno,trkno,side,count)
$14	Scrdmp()
$15	Cursconf(function, operand)
$16	Settime(datetime)
$17	Gettime()
$18	Bioskeys()
$19	Ikbdws(count, pointer)
$1A	Jdisint(intno)
$1B	Jenabint(intno)
$1C	Giaccess(data, regno)
$1D	Offgibit(bitno)
$1E	Ongibit(bitno)
$1F	Xbtimer(timer, control, data, vector)
$20	Dosound(pointer)
$21	Setprt(config)

$22 Kbdvbase()
$23 Kbrate(initial, repeat)
$24 Prtblk()
$25 Vsync()
$26 Supexec(codeptr)
$27 Puntaes()

Anhang H

Der BIOS-Parameter-Block

$00	RECSIZ	Größe der Sektoren in Bytes
$02	CLSIZ	Größe der Cluster in Sektoren
$03	CLSIZB	Größe der Cluster in Bytes
$04	RDLEN	Größe des Directories in Sektoren
$06	FSIZ	Größe der FAT in Sektoren
$08	FATREC	Erster logischer Sektor der FAT
$0A	DATREC	Erster logischer Datensektor
$0C	NUMCL	Anzahl der Cluster auf der Diskette

Anhang I

Die FDC-Kommandos

Typ	Name	Bitmap							
		7	6	5	4	3	2	1	0
1	RESTORE	0	0	0	0	h	v	r1	r2
	SEEK	0	0	0	1	h	v	r1	r2
	STEP	0	0	1	u	h	v	r1	r2
	STEP IN	0	1	1	u	h	v	r1	r2
	STEP OUT	0	1	1	u	h	v	r1	r2
2	READ SECTOR	1	0	0	m	h	e	0	0
	WRITE SECTOR	1	0	1	m	h	e	p	a0
3	READ ADDRESS	1	1	0	0	h	e	0	0
	READ TRACK	1	1	1	0	h	e	0	0
	WRITE TRACK	1	1	1	1	h	e	p	0
4	FORCE INTERRUPT	1	1	0	1	I3	I2	I2	I0

Stichwortverzeichnis

Die SYBEX-Bibliothek

Atari

ARBEITEN MIT DEM ATARI ST
von Karl-Heinz Hauer vermittelt Ihnen notwendige Kenntnisse zum Umgang mit den ATARI ST-Computern, z. B. System-Hardware, Betriebssystem-Adressen, TOS, Kernel-Routinen, ATARI-BASIC, ATARI-Logo. 432 Seiten, 172 Abb. Best.-Nr. **3623** (1986)

ATARI ST – ARBEITEN MIT GEM, Bd. 2: DIE VDI-BIBLIOTHEK
von Holger Danielsson/Andreas Volkmann – Der ATARI-ST-Nutzer wird anhand einer Vielzahl kleiner C-Routinen mit dem Aufruf der VDI-Bibliothek von GEM und der Einbindung in eigene Programme bekannt gemacht. 240 Seiten, ca. 48 Abb., Best.-Nr. **3627** (1986), Mit integrierter Programm-Diskette, die Programme und Unterroutinen enthält.

ATARI ST – EINFÜHRUNG IN WORDSTAR
von Arthur Naiman – Das Originalwerk „Einführung in WordStar" ist seit Erscheinen 1983 ein SYBEX-Bestseller. Um der Arbeit in der speziellen System-Umgebung des ATARI ST unter Kontrolle der CP/M-2.2-Emulatoren gerecht zu werden, wurde das Buch für ST-Nutzer überarbeitet und durch Zusatz-Informationen ergänzt. 280 Seiten, mit Abb., Best.-Nr. **3666** (1986/87)

ATARI ST – TIPS UND TOOLS ZU C
von Olaf Hartwig – führt den ATARI ST-Nutzer in die Sprache C ein und zeigt anhand vieler Beispiele, wie der Programmierer sich eigene Tools in C entwickeln kann: Einsatz von Makros, Modifikation der grammatischen Sprachstruktur, Entwicklung von ST-Sonderbibliotheken, Terminalverwaltung, Adressierung aller BIOS-, XBIOS- und GEMDOS-Funktionen. 224 Seiten, mit Abb., Best.-Nr. **3674** (1986)

Kommunikation

DAS MODEMBUCH ZUR DFÜ
von Bruno Hurth und Manfred Hurth – Jetzt steht Ihnen ein Nachschlagewerk mit einer Fülle unentbehrlicher Informationen zur Verfügung, auf das Sie immer wieder zurückgreifen werden: Möglichkeiten der DFÜ im Bereich der DBP, Mailboxen, Akustikkoppler, Btx u.v.m. 224 Seiten, Best.-Nr.: **3619** (1985)

SYBEX MAILBOX FÜHRER
Alles über den Zugang zu elektronischen Briefkästen in Deutschland: Voraussetzungen für die Teilnahme an der Datenkommunikation; Telefon-Nummern, Eingangs- und Untermenüs wichtiger Mailboxen und was diese Ihnen nutzen können, u.v.m. 272 Seiten, Best.-Nr.: **3663** (überarbeitete Ausgabe 1986)

ONLINE DATENBANKEN
Zugang zum Wissen der Welt mit Mikrocomputern von **Steffen Schubert** – Manager aller Unternehmensgruppen, Wissenschaftler und Ingenieure erhalten einen Überblick über alle Datenbank-Typen, bekommen wichtige Anbieter in Europa und Übersee genannt und lernen, Datenbanken effektiv und kostengünstig zu nutzen. 200 Seiten, Best.-Nr. **3621** (1986)

Einführende Literatur

CHIP UND SYSTEM: Einführung in die Mikroprozessoren-Technik
von Rodnay Zaks – eine sehr gut lesbare Einführung in die faszinierende Welt der Computer, vom Microprozessor bis hin zum vollständigen System. 2., überarbeitete und aktualisierte Ausgabe. 568 Seiten, 325 Abbildungen, Best.-Nr.: **3601** (1985)

Pascal

EINFÜHRUNG IN PASCAL UND UCSD/PASCAL
von Rodnay Zaks – das Buch für jeden, der die Programmiersprache PASCAL lernen möchte. Vorkenntnisse in Computerprogrammierung werden nicht vorausgesetzt. Eine schrittweise Einführung mit vielen Übungen und Beispielen. 535 Seiten, 130 Abbildungen, Best.-Nr.: **3004** (1982)

DAS PASCAL HANDBUCH
von Jacques Tiberghien – ein Wörterbuch mit jeder Pascal-Anweisung und jedem Symbol, reservierten Wort, Bezeichner und Operator, für beinahe alle bekannten Pascal-Versionen incl. Turbo Pascal. 520 Seiten, 270 Abbildungen, Format 23 x 18 cm, Best.-Nr.: **3614** (1986)

Assembler

PROGRAMMIERUNG DES 68000
von C. Vieillefond – macht Sie mit dem 32-bit-Prozessor von leistungsstarken Rechnern wie Macintosh, Amiga, ATARI ST und Sinclair QL vertraut; erläutert die Struktur des 68000, den Aufbau des Speichers, die Adressierungsarten und den Befehlssatz. 456 Seiten, 150 Abb., Best.-Nr. **3060** (1985)

Andere Programmiersprachen

ERFOLGREICH PROGRAMMIEREN MIT C
von J. A. Illik – ein unentbehrliches Handbuch für jeden, der mit der universellen Sprache C erfolgreich programmieren will. Aussagekräftige Beispiele, auf verschiedenen Mini- und Mikrocomputern getestet. 408 Seiten, Best.-Nr.: **3055** (1984)

SYBEX

ATARI ST

Das Floppy Arbeitsbuch

Die Floppy des ATARI ST-Systems ist kein Geheimnis mehr für Sie, wenn Sie dieses Buch gelesen und sich anhand der integrierten Programmdiskette mit den Interna auseinandergesetzt haben.

Die Autoren gehen detailliert auf das Datenhandling und die Programmierung des Floppydisk-Controllers ein, stellen die wichtigen Routinen des GEMDOS, des ATARI BIOS und XBIOS vor und erläutern anhand von Programmbeispielen den Umgang mit den leistungsfähigen Massenpeichern.

Auszug aus dem Inhalt:

- Die Datenorganisation der Floppydisk
- Diskettenprogrammierung unter TOS
- Die Floppydisk-Schnittstelle
- Programmierung des FDC
- Das ATARI BIOS und XBIOS

In das Buch integriert ist eine 3,5-Zoll-Diskette, randvoll mit unentbehrlichen Utilities (Filecopy, Speedcopy, Diskmonitor usw.) und Programmbeispielen einschließlich deren Quelllistings.

Buch und Diskette bilden ein leistungsfähiges Entwicklungssystem, das jeder engagierte ATARI ST-Besitzer für seine Arbeit mit dem System benötigt.

ISBN 3-88745-642-4

unverbindliche Preisempfehlung

DM 69,–
sFr 69,–
öS 614,–

9 783887 456429