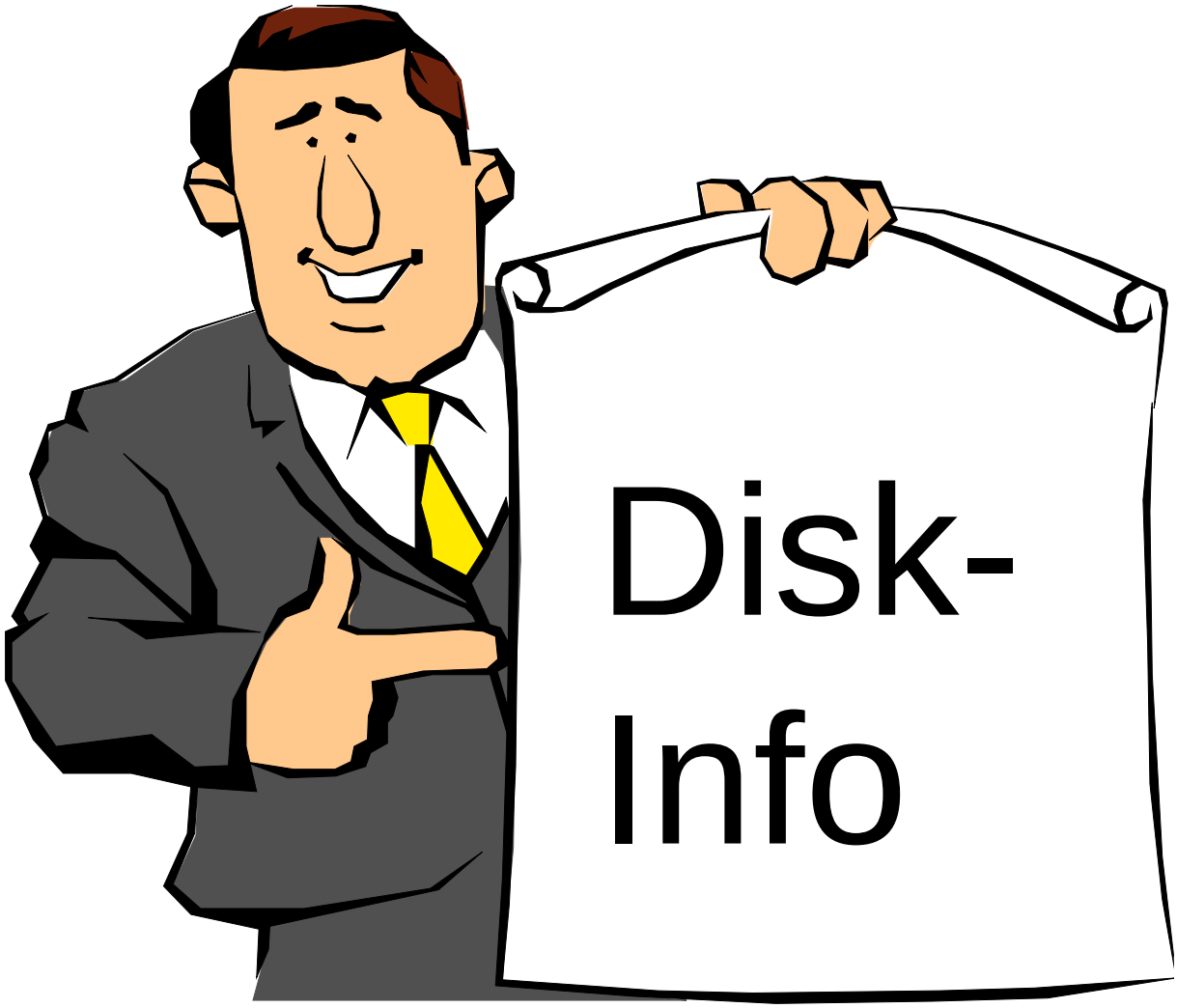


Eine Publikation der system 99 user-group

DISKINFO

Eine Informationssammlung zum Thema Datenträger, deren Verwaltung und Hardware-Ansteuerung beim

TI-99/4A

Stand: 6. Mai 1991

Das Kleingedruckte:

Diese Fassung von Diskinfo stellt nicht die endgültige Form sowohl vom Layout wie vom Inhalt dar. Einige Kapitel sind noch unvollständig bzw. lediglich betitelt. Es existiert die Wahrscheinlichkeit, daß es in Zukunft Erweiterungsblätter zur Diskinfo geben wird. Nur diejenigen Besitzer einer autorisierten Kopie von Diskinfo, die an die im Vorwort aufgeführte Adresse einen adressierten und frankierten Rückumschlag senden, erhalten im Falle der Fertigstellung weiterer Diskinfo-Blätter einen entsprechenden Hinweis.

Inhaltsverzeichnis

Urheberrechtsvermerke

Für Diskinfo gelten Regeln, wie für jedes andere Dokument. Das bedeutet den Verzicht des jeweiligen Besitzers auf die Anfertigung jeder Art von Kopien und/oder die Umsetzung des gedruckten Textes in maschinenlesbare Form bzw. deren Vorbereitung. Selbstverständlich gilt dies für Ausschnitte wie für die Gesamtheit des Werkes.

Auch wird jegliche Haftung bei der Anwendung der hier aufgezeigten Methoden und Zusammenhänge ausgeschlossen, ebenso die Gewähr für die unbeschränkte Richtigkeit und Vollständigkeit der gemachten Aussagen sowie deren Nutzbarkeit für bestimmte Zwecke.

Im Rahmen des Urheberrechtes sind Übersetzungen genauso zulässig wie Ausarbeitungen, die auf den in Diskinfo zusammengetragenen Daten beruhen, jedoch nur mit unmißverständlicher Quellenangabe.

Die Wiedergabe von Auszügen aus Diskinfo im Rahmen von Clubzeitungen wird ausdrücklich erlaubt, sofern diese Clubzeitungen ausschließlich in gedruckter Form vorliegen - ggfs. sind die entsprechenden Abschnitte den Disketten in Kopie der Diskinfo-Seiten beizulegen.

Sollten Sie eine offizielle Version von Diskinfo erwerben wollen oder Vorschläge zur Verbesserung von Inhalt und Struktur haben, so wenden Sie sich bitte an die folgende Adresse:

Christopher Winter	Harald Glaab
xxxxxxxxxxxxxxxx	xxxxxxxxxxx
xxxxx xxxxxxxxxx	xxxxx xxxxxxxxxxx

Bitte fügen Sie bei Anfragen, zu denen Sie eine Antwort haben möchten, einen adressierten und frankierten Rückumschlag bei. Bitte vermeiden Sie Geldsendungen (statt Rückumschlag) und telefonische Kontaktaufnahme.

Kontakt: http://s-n-u-g.de

Vorwort zur überarbeiteten dritten Auflage

Diese dritte Auflage entstand zu einer Zeit, als der TI auf dem besten Wege war, in die Bedeutungslosigkeit abzusacken. Möglich wurde diese überarbeitete Fassung nur durch den Einsatz eines hochleistungsfähigen PC sowie eines ebenfalls leistungsfähigen Textsystems. Hätte die inzwischen auf einen Umfang von 2 DS/DD-Disketten angewachsene Datensammlung, als die sich Diskinfo darstellt, auf einem TI bearbeitet werden müssen, wäre sie nie zustande gekommen.

Das Kapitelsystem wurde beibehalten, weshalb es keine kapitelübergreifend fortlaufenden Seitennummern gibt. Ergebnis des PC-Einsatzes ist das umfangreiche Stichwortverzeichnis, welches das alte Sachwortverzeichnis mehr als ersetzt.

Das alles mag den einen oder anderen TI-Freak wundern oder gar schmerzen, doch es kann wohl kaum angehen, den TI in der heutigen Zeit als Maß aller Dinge anzusehen. Indes war und ist die in Diskinfo zusammengetragene Informationsmenge, ob sie nun zu 100% korrekt ist oder nicht, einfach zu wertvoll und arbeitsintensiv, um sie solch profanen Weltanschauungen zu opfern.

Bei dieser Diskinfo wurde erneut die Gratwanderung versucht, trockenen Stoff mit Beispielen aus der Praxis und launischen Bemerkungen etwas aufzulockern.

Wem dieses Konzept oder Diskinfo im allgemeinen gefällt, der ist aufgefordert (aber nicht verpflichtet), dem Autor einen kurzen Brief mit seiner Kritik zukommen zu lassen. Es sollte bedacht werden, daß Kritik sowohl positive als auch negative Meinungsäußerungen umfaßt!

Auch oder gerade in dieser Fassung von Diskinfo gibt es noch etliche Themen, die nicht fertiggestellt oder so erschöpfend wie möglich beschrieben werden konnten. An dieser Stelle sei hierfür um Verständnis gebeten und die vage Hoffnung auf ein Update geweckt.

Die Vorworte zur zweiten und ersten Auflage wurden aus der Einleitung gestrichen - ihr Inhalt war nicht mehr gültig bzw. in seinem Neuigkeitswert inzwischen stellenweise überholt. Es sollte immer bedacht werden, daß Diskinfo inzwischen schon über 4 Jahre alt ist.

Obertshausen, im August 1990 Christopher Winter

Vorwort zur vierten Auflage

Nachdem jetzt mittlerweile noch einmal fast 10 Jahre ins Land gegangen sind, fanden wir es an der Zeit, jetzt endlich allen verbliebenen TI-Benutzern in aller Welt die Informationen der **DISKINFO** zukommen zu lassen.

Daher wurde – mit ausdrücklicher Erlaubnis des Autors – eine nochmalige, mehrwöchige Überarbeitung der Textdateien vorgenommen, zumal es damals mit einem heute nicht mehr verfügbaren Textsystem erstellt wurde.

Daher bitte ich zu entschuldigen, wenn eine Tabelle oder Grafik nicht so dargestellt wird, wie es vielleicht in der originalen Version zu sehen war.

Das bisher verwendete Kapitelsystem und die dazugehörige Seitennumerierung konnte nicht mehr wieder hergestellt werden, so daß entgegen der ursprünglichen Vorstellungen des Autors wieder auf eine fortlaufende Numerierung zurückgegriffen werden mußte. Dies auch im Hinblick darauf, daß es mit großer Wahrscheinlichkeit keine weiteren Updates der **DISKINFO** mehr geben wird.

Aus Gründen der Aufwandsminimierung wurde auf das Stichwortverzeichnis ganz und gar verzichtet.

Aschaffenburg, im Oktober 2000 – Harald Glaab, system-99 user-group

Hinweise zu verwendeten Begriffen und Methoden

In dieser Diskinfo werden einige neue Begriffe eingeführt und einige vielleicht bekannte in einem anderen Sinn verwendet. Der notwendigen einleitenden Klärung sollen die folgenden Absätze dienen.

Der Begriff der DSR

Der zentrale Angriffspunkt für Mißverständnisse ist das Kürzel DSR, was ursprünglich für DEVICE-SERVICE-ROUTINE(S) steht. In der Regel bezeichnet dies die Steuerungsroutinen, welche zur 'Bedienung' eines Peripheriegerätes nötig sind. Aufgrund der Komplexität der Funktionen bei der Bedienung von Diskettenstationen wird DSR hier als Synonym für die GESAMTHEIT aller Programme einer Diskettensteuerungssoftware benutzt, so daß sich der weniger versierte Anwender unter DSR am Besten den Speicherchip der Controllerkarte vorstellt.

Level-0,1,2-Routinen

Die Einteilung der DSR in Routinen unterschiedlicher Komplexitäts- und Funktionalitätsstufen verlangte, hierfür ein Bezeichnung zu definieren. Die Programmteile der DSR, welche hart an der Hardware arbeiten und demzufolge Funktionen geringer Komplexität erfüllen, werden Level-0-, jene, die auf File-Ebene ganze Informationsgefüge bearbeiten, werden Level-2-Routinen genannt. Die dazu benutzten Dienstprogramme der DSR, welche die Vermittlerrolle zur physikalischen Datenstruktur übernehmen, werden folglich Level-1-Routinen genannt. Die Routinen der Stufen (Levels) 1 und 2 sind extern aufrufbar, die Level-0-Routinen müssen ggfs. anhand der in den Kapiteln 18ff beschriebenen ROM-Listings nachprogrammiert werden (von direkten Einsprüngen ist dringend abzuraten!).

Bit-Byte-Word-Notation / Bit-Numerierung

Bei der Numerierung von Bits, bekanntlich der Basiseinheit, in welcher ein Computer 'denkt', hat Texas Instruments selbst für ziemliche Verwirrung gesorgt.

Üblich ist die Numerierung von Bits eines Bytes, Words oder was auch immer so geregelt, daß die Bit-Nummer die Wertigkeit im dualen Zahlensystem darstellt. So ist Bit 0 das Bit mit der niedrigsten Wertigkeit 1, Bit 7 eines Bytes etwa dasjenige mit der Wertigkeit 128.

Nicht so bei TI (oder nicht immer, wie es der Kenner verschiedener Veröffentlichungen aus dem Hause Texas Instruments kennt). Hier numeriert man nach dem Verfahren Links-Nach-Rechts, womit also das höchstwertige Bit ganz und gar nicht standesgemäß die Nullnummer ist. Daß dieses Verfahren so unsinnig nicht ist, wird weiter unten vielleicht deutlicher.

Das Bit mit der höchsten Wertigkeit wird MSB (Most Significant Bit), das mit der niedrigsten LSB (Least Significant Bit) genannt.

Wenn es um Bytes und Words (Worte) geht, so nimmt das Chaos leicht zu. Da die TMS9900 ein 16bit-Microprozessor ist, ist sie in der Lage, 2 Bytes auf einmal als ein Word (Wort) anzusprechen. Beim TI wird durch eine entsprechende Hardware in der Konsole zwar ein äußerlicher 8-Bit-ter daraus, doch die CPU sieht das anders. Sie kann aus einem Word heraus noch ein einzelnes Byte adressieren, das Low- oder das High-Byte. Das konfuse daran ist, daß das High-Byte zwar das Byte eines Words ist, welches die höhere Wertigkeit hat (analog zu den Bits), das High-Byte jedoch an der niedrigeren Speicheradresse im 8bit-Adreßbereich des TI liegt - das Pseudo-Adressbit A15 ist beim High-Byte logisch 0. In diesem Fall wäre also die umgekehrte TI-Zählweise für einzelne Bits taktisch klüger.

Im folgenden wird die Bit-Byte-Word-Struktur graphisch dargestellt.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSB	High Byte						LSB	MSB	Low Byte						LSB
MSB	Word														LSB

Unsaubere Programmierung

Verschiedentlich ist in dieser Diskinfo von 'Unsauberer Programmierung' die Rede. Dies mag auf den Uneingeweihten wie Arroganz wirken, was es aber keinesfalls sein soll. Unsaubere Programmierung findet man überall da, wo sich Programmierer von Anwendersoftware nicht an die Richtlinien des Herstellers der be-

nutzten Hard- und / oder Software halten. Im Grundsatz fallen alle Versuche, das Betriebssystem zu hintergehen in diese Kategorie. Speziell beim TI ist dieser Begriff so zu interpretieren, daß sich diverse Hersteller von DSR-Software nicht an die TI-Konventionen bzw. -Vorschriften halten, und so Inkompatibilitäten mit anderer Hard-Software provozieren. Die bekanntesten Vertreter sind fast alle Karten der Firma Myarc sowie die früheren CorComp-Diskcontroller mit eigenem Titelbild, welche so manchem aufgerüsteten TI das 'Leben' schwer machten. Seit kurzer Zeit gehören einige RAM-Disks dazu, deren DSR-Schmiede (es ist manchmal fast anmaßend, von Programmierern zu reden...) es als innovativ ansehen, festgelegte DSR-Funktionen nach eigenem Gutdünken zu erweitern, ohne einen Standard zu beachten.

Der PAB

Wenn es um Disketten und Diskettensteuersoftware geht, dann führt am PAB kein Weg vorbei. Es handelt sich hier um eine wohldefinierte Gruppierung von Daten, welche über das VDP-RAM an eine DSR übergeben wird. Die DSR protokolliert hier ihre Aktionen und erlaubt so den Datenaustausch mit jedem beliebigen Anwenderprogramm.

Seitennummern

Da Diskinfo als offenes Dokument verstanden wird, sollten ggfs. notwendige Updates möglichst einfach durchgeführt werden können. Da zudem die Auffindbarkeit einzelner Themen oder Stichworte mittels einer Kapitelnumerierung wesentlich erleichtert wird, besitzt jedes Kapitel eine eigene Nummer und eine Zählweise der Seiten ab 1 bei jedem Kapitelwechsel. Inhalts- und Stichwortverzeichnis entsprechen diesem Muster; Einträge im Inhaltsverzeichnis werden jedoch nicht mit der Abschnitts- und Hierarchiestufe versehen, um bei einem Update keine Verwirrung zu verursachen.

Physikalischer Aufbau

Der Aufbau einer regulären Ausgabe von Diskinfo erlaubt es, illegale Kopien leicht zu identifizieren. Die Bindung erfolgt mittels Kunststoff-Bindeleisten mit Rechtecklochung. Als Schutzklappen werden transparente Kunststofffolien verwendet. Zwischen zwei Kapiteln befindet sich jeweils ein mit dem Kapitelnamen bedrucktes, blau eingefärbtes Blatt Papier, Inhalts- sowie Stichwortverzeichnis sind auf rosafarbenem Papier gedruckt. Jedes Kapitel besitzt eine gerade Anzahl von Seiten.

Einleitung

Die TI-Konsole und ein Kassettenrecorder - mit dieser Geräteanordnung haben wohl die meisten Besitzer eines TI 99/4A begonnen. Die Einschränkungen aber, die beim Betrieb eines Kassettenrecorders am TI bestehen, wie limitierte Programmlänge (ca. 13 KByte), kaum Maschinenspracheprogramme verfügbar und besonders die lange Lade- und Speicherzeit werden wohl sehr bald den Wunsch nach einem zeitgemäßeren Datenspeicher geweckt haben.

In diesem Punkt wird wieder einmal die Marktstrategie von Texas Instruments deutlich, mit welcher der 99er unters Volk gebracht werden sollte: erst mit der Konsole relativ preiswert ins Computern einsteigen, nur um dann den Gegenwert eines gebrauchten Mittelklassewagens investieren zu müssen, bis sich die Anlage auch wirklich Computer nennen darf.

In der Tat ist der TI 99/4A erst mit Diskettenlaufwerk und Speichererweiterung konkurrenzfähig, was sicher für den marktwirtschaftlichen Schiffbruch von TI mit verantwortlich war.

Die Leistungsfähigkeit von Floppy-Disk Laufwerken und dem Speichermedium 'Flexible Disk' liegt wohl in der Hauptsache in der recht hohen Übertragungsgeschwindigkeit begründet, daneben aber auch in der Möglichkeit, per Software direkt die Daten auszusuchen, mit denen zum beliebigen Zeitpunkt gearbeitet werden soll. Bezüglich gerade der Transfergeschwindigkeit kann der TI auch 1986, 3 Jahre nach Produktionseinstellung, immer noch im Homecomputerbereich als Vorbild angesehen werden. Dies liegt auch daran, daß TI sich hier auf kein nebulöses, hausgemachtes System verlassen hat, sondern konsequent nach Industriestandard vorging.

Als Besitzer einer TI 99/4A-Anlage ist man somit in der glücklichen Lage, prinzipiell beliebige auf dem Markt erhältliche Diskettenlaufwerke an die diversen Diskcontroller anschließen zu können.

Die Leistungsfähigkeit der Massenspeichertechnik bewirkt jedoch auch, daß der Überblick über Funktionen und Funktionsabläufe schon bei Disketten zunehmend eine Sache für Fachleute wird. Gerade im Bereich der Homecomputer ist es aber üblich, das Innenleben seines Rechners verstehen zu wollen. Soweit es das Feld der Diskettenoperationen betrifft, soll diese Diskinfo die vorhandenen Lücken in der Information über den TI 99/4A in kompakter und umfassender Form schließen, ohne dabei andauernd auf andere Publikationen zu verweisen, die dann (Murphy sei Dank) gerade nicht verfügbar sind.

Zum Schluß sei eines noch angemerkt: Die dem durchschnittlichen Anwender normalerweise unzugängliche Möglichkeit, einzelne Sektoren auf der Diskette zu lesen, zu interpretieren und zu manipulieren, schaltet bisher wirkungsvolle Kopierschutztechniken aus. Der Leser und Anwender dieser Diskinfo-Blätter ist angehalten, sich selbst Gedanken darüber zu machen, was sinnloses Raubkopierertum gerade dem empfindlichen Softwaremarkt des TI zufügt. Geringe Absatzzahlen haben schon mehr als einem Programmierer die Lust genommen!

Grundlegende Mechanismen und Konzepte des TI-DSR-Systems

DSRs beziehen sich immer auf externe Erweiterungen (Peripherien) des TI-Systems, die zur Funktion eigene Programme benötigen, welche der TI-Monitor im ROM und GROM der Konsole nicht liefert.

Jede Karte in der PE-Box (Peripheral-Expansion-Box, Peripherie-Erweiterungs-Box), die irgendwelche Verwaltungsarbeiten aktiv übernimmt (RS-232, Diskcontroller, Video-Karte, RAM-Disk etc.) benötigt eine DSR.

Dabei stellt jede DSR ein sog. GERÄT zur Verfügung, das einen einzig artigen Namen haben muß, durch den es identifiziert wird. Beim Zugriff auf ein bestimmtes Gerät wie einen Drucker oder eine Datei auf dem Gerät DISKETTENSTATION muß also der Name dieses Gerätes bekannt sein und angegeben werden.

Diese Daten werden zusammen mit der gewünschten Operation (Lesen, Schreiben, Öffnen, Schließen etc.) und einigen anderen Daten codiert in einem sog. PAB hinterlegt (Peripheral Access Block, Peripherie-Zugriffs-Block). Dieser PAB wird meist im CPU-RAM aufgebaut und kurz vor Aufruf der Gerätesuchroutine ins VDP-RAM kopiert.

Die DSRLNK-Routine

Die Adresse dieses PAB, der immer im VDP-RAM liegt, wird an eine Routine mit Namen DSRLNK übergeben (Device-Service-Routine-Link), welche das Gerät selbständig sucht und die entsprechende Funktion veranlaßt.

Diese Routine ruft die entsprechenden Programme der passenden DSR auf, wobei sie die passenden Adressen aus dem nach bestimmten Regeln aufgebauten sog. DSR-Header (siehe dort) ermittelt, und kehrt danach zum rufenden Programm zurück, dem es eine Rückmeldung liefert, wie die Funktion ausgeführt wurde. Sie meldet auch, wenn der angegebene Name zu keiner im System vorhandenen Gerätesteuersoftware (DSR) paßt.

Der weitere Datenaustausch findet ausschließlich anhand des PAB statt, der weiter unten beschrieben wird.

Alle DSRs belegen den Speicherbereich >4000 - >5FFF bzw. können maximal diesen Bereich nutzen. Größere DSRs arbeiten mit Paging, wobei mehrere Seiten eines oder mehrerer ROMs in den zugewiesenen Adreßbereich eingeblendet werden.

Die Auswahl der jeweiligen DSR erfolgt mittels sog. CRU-Seiten. Der CRU-Bereich ist ein spezieller I/O-Bereich der CPU TMS9900, auf den mit reservierten Befehlen zugegriffen wird. Vereinfacht kann man beim TI-99/4A sagen, daß der Adreßbus 3 weitere Bits erhält. Ursprünglich waren diese Bits übrigens statisch gepuffert in der Box vorgesehen.

Jede CRU-Seite einer PE-Box-Karte umfaßt 128 Bits, wobei das erste (Bit 0) zum ein- und ausschalten der jeweiligen DSR benutzt wird.

DSRLNK schaltet also nacheinander jede Karte ein, prüft auf den Gerätenamen und schaltet bei negativem Ergebnis die aktuelle Karte aus und ggfs. die nächste an. Wurde der gesamte CRU-Bereich (>1000 bis >1F00 in Schritten zu >100) abgesucht, so liegt ein I/O-Error-0 vor; das Gerät existiert im System nicht. Bei der Zählweise ist zu beachten, daß CRU-Bits bzw. deren Adressen erst durch Ignorieren von A15 entstehen, über das die 16bit-CPU TMS9900 ja eigentlich nicht verfügt, und A15 eigentlich A0 ist, Texas-Instruments jedoch früher immer falsch 'rum gezählt hat (entgegen der Wertigkeit eines Bits).

Der PAB

Der PAB ist, wie bereits oben angeführt, das Bindeglied zwischen dem Daten anfordernden oder absendenden Programm und dem zugehörigen Gerät, welches die Daten verarbeitet.

Der PAB enthält den sog. I/O-Code, das File-Attribut, die Pufferadresse, die Anzahl zu lesender oder zu schreibender Bytes, den Geräte-und ggfs. den Dateinamen sowie dessen Länge. Zusätzlich sind einige Bits für die Rückgabe von Statusinformationen durch die DSR selbst reserviert, sowie ein Byte, das es einer DSR erlaubt, ggfs. selbsttätig Meldungen auf den Schirm zu bringen, die auch sichtbar sind (Screen-Offset, wird jedoch nur von GPLLNK benutzt).

Werden Geräte- und Dateinamen verwendet, müssen diese mit einem Punkt getrennt werden. Nachfolgende Trennsymbole für Optionen des jeweiligen Gerätes werden von der DSR vorgegeben, müssen also keine Punkte mehr sein, obwohl dies sicher die sauberste Lösung wäre.

Der PAB muß im VDP-RAM liegen, sobald DSRLNK aufgerufen wird. Um der DSRLNK-Routine die Suche des Gerätes zu ermöglichen, zeigt das Wort an Adresse >8356 auf die Position des Bytes mit der Namenlänge des Geräte- & ggfs. Dateinamens. Dieser Pointer **muß** gesetzt werden.

Da der PAB den Informationsfluß zwischen Anwenderprogramm und DSR steuert (ein Zugriff direkt auf das VDP-RAM anhand der in Kapitel 8 beschriebenen Adressen ist unbedingt zu vermeiden), muß er für die gesamte Zeit, in der das betreffende File offen ist, im VDP-RAM vorliegen. Er darf nur in den Parametern für die zu verschiebende Byteanzahl, die Satznummer, die Pufferadresse sowie den I/O-Modus und das Fehlerbyte modifiziert werden; Gerätename und Satzlänge sind tabu!

Beispiel für einen PAB

Byte 0&1	I/O - Opcode	Flag / Status
Byte 2&3	Adresse Datenpuffer im VDP	
Byte 4&5	Satzlänge	Zeichenanzahl
Byte 6&7	Satznummer (0-32767) - Länge (Memory Image)	
Byte 8&9	Screen Offset	Namenlänge
Byte 10ff	Dateiname	

I/O-Opcode

Dieses Byte gibt die Operation an, die durchzuführen ist:

I/O Modus	Funktion	Beschreibung
00	OPEN	Datei öffnen wie im Flag-Byte angegeben
01	CLOSE	Datei schließen
02	READ	Datensatz lesen
03	WRITE	Datensatz schreiben
04	RESTORE/REWIND	Dateizeiger auf Dateianfang setzen
05	LOAD	Laden eines Memory-Image-Files
06	SAVE	Speichern eines Memory-Image-Files
07	DELETE	Löschen einer Datei(!)
08	SCRATCH RECORD	Löschen eines Datensatzes (bisher in keiner DSR unterstützt!)
09	STATUS	Ermitteln von Informationen zur Struktur

Flag/Status

Hier ist eine Bitmaske abgelegt anhand der die DSR erkennt, um welche Art von File es sich grundsätzlich handelt. Die drei höchstwertigen Bits sind zudem für die Übermittlung von sog. File-Error-Codes reserviert. Der Aufbau der Bitmaske ist wie folgt:

7 bis 5	4	3	2 und 1	0
Fehlercode	**Satzlänge**	**Datentyp**	**Operation**	**Dateityp**
000 = Ungültiger Gerätename 001 = Gerät Schreibgeschützt 010 = Ungültiger Datentyp, Satzlänge, Dateityp 011 = Von DSR nicht unterstützter Modus 100 = Gerät kann keine Daten mehr aufnehmen 101 = Versuch über das Ende hinaus zu lesen 110 = Gerätefehler 111 = Dateifehler, Programm als Datei gelesen	0 = fest 1 = variabel	0 = Display 1 = Internal	00 = Update 01 = Output 10 = Input 11 = Append	0 = Sequent. 1= Relativ

Für weitere Informationen lesen Sie bitte unter Ein-/Ausgabeoperationen weiter unten in diesem Kapitel. Dieses Byte wird zudem in leicht modifizierter Form in jedem sog. File-Directory-Sektor verwaltet, was im Kapitel 7 weiter ausgeführt wird.

Datenpufferadresse

Dieses Wort gibt an, ab welcher Adresse im VDP-RAM die zu schreibenden Daten abgelegt sind bzw. wo im VDP-RAM die aus der Datei zu lesenden Bytes an das Anwenderprogramm übergeben werden sollen.

Satzlänge

Anzahl Bytes in einem Datensatz fester Länge bzw. maximale Länge von Datensätzen variabler Länge.

Zeichenanzahl

Anzahl der Bytes eines Eintrages, z.B. aktuelle Satzlänge (Stringlänge o.ä.) beim Schreiben eines Satzes variabler oder fester Länge. Beim Lesen die Anzahl tatsächlich gelesener Zeichen im Datensatz (variable und feste Satzlänge).

Satznummer

Logische Nummer des zu lesenden/schreibenden Datensatzes einer Direktzugriffsdatei. Beim Laden eines Memory-Image-Files die Puffergröße und damit die maximale Filelänge; beim Schreiben die Anzahl zu schreibender Bytes eines Memory-Image-Files.

Screen-Offset

Optionaler Zeichenoffset, falls die DSR selbst Meldungen ausgeben muß. Derzeit wird dies nur von der Cassetten-DSR benutzt (und wer, der Diskinfo liest, speichert schon auf Cassetten...).

Namenlänge

Gesamtlänge des folgenden Geräte- und ggfs. Filenamens (inkl. aller Sonderzeichen wie Punkte etc.).

Gerätename

Der erste Teil des Geräte- und Filenamen- bzw. Optionsstrings bezeichnet die DSR, die für diesen Vorgang zuständig ist. Dabei wird der Punkt ('.') als Trennsymbol bzw. Ende des Gerätenamens verwendet. Anhand dieses Namens sucht das DSRLNK die passende Karte in der Peripheriebox. Der ggfs. vorhandene Rest dieses Strings wird von der DSR selbst ausgewertet. Der Gerätename ist von TI auf eine maximale Länge von 7 Zeichen begrenzt worden. Ist nur der Gerätename vorhanden, kann auf den Punkt an dessen Ende verzichtet werden.

Filename / Option

Nach dem Gerätenamen folgt bei einer Disk-DSR i.d.R. ein Filename (Ausnahme Diskkatalog). Bei Schnittstellenkarten wie z.B. der RS-232-Karte kann hiermit der Kanal konfiguriert werden.

Beispiel für einen BASIC-PAB

Der Basic-PAB verfügt über 2 weitere Wörter (1 Wort + 2 Byte), welche primär als Verwaltungshilfen für Basic zu verstehen sind. Alle weiteren Einträge sind identisch zu den oben beschriebenen.

Bit 0 & 1	Link zum nächsten PAB	
Bit 2 & 3	Kanal- / Filenummer	Datenpufferoffset
Bit 4 & 5	I/O - Opcode	Flag / Status

PAB-Link

Basic benötigt diesen Zeiger zur Verwaltung mehrerer offener Files. Alle PABs sind hiermit in einer einfach verketteten Liste angeordnet, beginnend mit dem zuerst geöffneten File. Dies vereinfacht u.a. die sog. Garbage-Collection, mit der unbenutzter Speicherplatz (z.B. nach dem Schließen einer Datei) wieder zugänglich gemacht wird. Der PAB-Link ist ein Zeiger auf die nächste PAB-Link-Adresse des nachfolgend geöffneten Files.

Kanal-/Filenummer

Dies ist die Filenummer, unter der Basic alle Zugriffe zwischen dem Öffnen und Schließen der Datei verwaltet.

Datenpufferoffset

Insbesondere bei sog. Schwebenden Ausgabebedingungen wird ein Datenpuffer erst dann auf Diskette geschrieben, wenn er voll ist bzw. überläuft. Dies gilt analog beim Lesen, wenn ein Datenpuffer schrittweise ausgelesen wird. Näheres zur Umsetzung von blockstrukturierten Geräten finden Sie in Kapitel 4.

Ein-/Ausgabeoperationen

Beim TI-99/4A wird grundsätzlich zwischen zwei verschiedenen Dateitypen unterschieden:

- den sequentiellen Dateien mit dem Sonderfall P-File,
- und den Dateien mit direktem bzw. wahlfreiem Zugriff, auch als Relative Dateien bezeichnet.

Die sequentiellen Dateien werden von fast jedem Ein-/Ausgabegerät unterstützt (Drucker, serielle Kommunikation etc.), die Dateien für wahlfreien Zugriff (sog. RANDOM-Files) sind dagegen eine Spezialität des Diskettenbetriebssystems.

Folgende Schlüsselworte werden im Filesystem benutzt, um Daten- und Dateitypen zu beschreiben:

- Datentypen: DISPLAY oder INTERNAL
- Satzlänge.: VARIABLE oder FIXED
- Operation.: INPUT, OUTPUT, APPEND oder UPDATE

Datentypen

Das TI-File-System unterscheidet nicht zwischen reinen ASCII- und sog. Binärdaten (bezeichnet mit den Datentypen DISPLAY und INTERNAL), da es für die Speicherung auf einem Datenträger unerheblich ist, ob jedes Byte nun den gesamten Wertebereich von 0 - 255 umfaßt oder dieser Wertebereich nur teilweise genutzt wird. Überall da, wo ein solcher Unterschied auftritt, ist es eine Spezialität z.B. des BASIC-Interpreters o.ä.

Zusätzlich steht noch eine Sonderdateiform zur Verfügung, das sog. Memory-Image-Format, auch als P-File-Format bezeichnet. Hierbei wird ein zusammenhängender, meist unstrukturierter Block von Daten als Speicherabzug an einem Stück auf Diskette geschrieben bzw. von ihr gelesen. Der wesentliche Vorteil dieses Formats ist die hohe Lese- bzw. Schreibgeschwindigkeit.

Satzlängen

Bei der Eröffnung einer Datei im Ausgabemodus muß die Satzlänge vorgegeben werden, bei der Eröffnung einer Datei im Eingabemodus ist dies abhängig von der Diskcontroller-DSR - nicht nötig, da eine Länge von Null (0) vorgegeben werden kann und dann die DSR selbst die Länge in den PAB einsetzt.

Sequentielle Dateien können aus Datensätzen variabler oder fester Länge bestehen. Auf Datensätze einer sequentiellen Datei kann immer nur in aufsteigender Reihenfolge zugegriffen werden, da das File-System nach außen keinen Byte-Zeiger auf die Datei zur Verfügung stellt (der jedoch von der DSR verwaltet wird). Ein Zugriff auf irgendeinen vorhergehenden Datensatz ist somit nur möglich, wenn vom ersten Datensatz an entweder gelesen oder geschrieben wird, bis der Gewünschte erreicht ist.

Voraussetzung für den Einsatz von Dateien mit wahlfreiem Zugriff ist die eindeutige Lokalisierbarkeit eines jeden Datensatzes. Hierzu muß jeder Datensatz die gleiche Länge haben. Somit kann das Betriebssystem feststellen, in welchem Sektor und an welcher Stelle im Sektor ein bestimmter Datensatz, dessen Nummer angegeben wird, zu finden ist.

Abschließend bleibt zu bemerken, daß eine Datei explizit als aus Datensätzen gleicher Länge aufgebaut deklariert werden muß. Es reicht nicht, implizit eine Datei mit Datensätzen fester Länge dergestalt aufzubauen, daß sie im VARIABLE-Format aufgebaut wird und ausschließlich z.B. mit Leerzeichen aufgefüllte Datensätze einer einheitlichen Länge geschrieben werden. Auf mögliche Tricks, wie das Eröffnen einer solchen Datei als Direktzugriffsdatei mit einer Länge +1 soll hier nicht weiter eingegangen werden.

Operationen

Input- und Output-Modus dürfen als die elementaren Ein-/Ausgabeoperationen angesehen werden. Eine Datei, welche im Input-Modus geöffnet wird, kann nur gelesen werden, jeder Schreibversuch bewirkt eine Fehlermeldung des DOS.

Eine im Output-Modus geöffnete Datei wird, sofern Sie existierte, gelöscht und neu angelegt.

Der Append-Modus stellt eine Sonderform des Schreibens in eine sequentielle Datei dar. Dabei werden Datensätze ausschließlich am aktuellen Ende der Datei angehängt.

In diesen 3 Modi existiert außerhalb der DSR kein Datensatzzeiger. Diesen verwaltet die DSR intern, womit z.B. beim sequentiellen Zugriff immer auf die aktuelle Dateiposition für die folgende Lese- oder Schreiboperation gezeigt wird.

Der Update-Modus ist nur für Dateien mit wahlfreiem Zugriff bzw. solche mit festen Satzlängen zulässig. Zum Zugriff auf einen bestimmten Datensatz muß dessen logische Satznummer angegeben werden. Beim Lesen aus oder Schreiben in eine UPDATE-Datei inkrementiert die DSR den Datensatzzeiger selbsttätig. Die Satzanzahl kann 1 bis 32768 betragen, wobei der erste Datensatz die Nummer 0 bekommt. Daher reicht der Wertebereich des Satzzeigers von 0 bis 32767.

Aufbau des DSR-Headers

Jede DSR, ob sie nun per CRU in den Bereich >4000 - >5FFF eingeblendet wird oder im Modul-Port-ROM oder gar GROM vorliegt, muß einem einheitlichen Aufbauschema folgen, wenn sie korrekt eingebunden werden will. Damit soll sichergestellt werden, daß die Idee der Vereinheitlichung auch realisiert wird.

Der sog. Implementationsteil eines DSR-ROMs beginnt an der Adresse >4000 mit den 16 Bytes des DSR-Headers:

>4000	BYTE	>AA	Identifikation eines DSR-ROMs
>4001	BYTE	>01	Versionsnummer, beliebig
>4002	DATA	>0000	reserviert
>4004	DATA	PWRLNK	Power-Up-Link, Initialisierung der Karten Hard- und Software nach d. Einschalten bzw. Reset
>4006	DATA	>0000	Reserviert
>4008	DATA	DSRLNK	Startadresse der DSRLNK-Tafel
>400A	DATA	SBRLNK	Startadresse der SBRLNK-Tafel
>400C	DATA	INTLNK	Startadresse der INTLNK-Tafel
>400E	DATA	>0000	Reserviert für zukünftige Erweiterungen

Lediglich diese 16 Bytes sind in ihrer Adreßzuordnung fest vorgeschrieben - auf alle damit verknüpfen Listen und Tabellen wird über Zeiger zugegriffen, deren einzige Beschränkung darin liegt, daß sie im Adreßbereich der DSR auf der ersten ROM-Seite referenziert werden müssen.

Die Listen für PWRLNK, DSRLNK, SBRLNK und INTLNK sind alle gleich aufgebaut, wobei das Strukturmuster jeweils nur leicht modifiziert wird. Sogar die Aufruf-Sequenzen für die entsprechenden Funktionen ähneln sich weitestgehend. Dabei werden Einsprünge über diese Zeiger immer nur in bestimmten Situationen vorgenommen.

So wird beim Reset des Systems (Einschalten oder Hard- bzw. Software-Reset) die PWRLNK-Tabelle eines jeden DSR-ROMs abgesucht und eine entsprechende Routine abgearbeitet.

Wird ein externer Interrupt ausgelöst, so sucht der Interrupt-Handler der Konsole alle DSR-ROMs auf einen gültigen Wert ab und springt in die Interrupt-Routine der Karte.

Ein Zeiger mit dem Wert >0000 an einer dieser Adressen zeigt dem System an, daß entsprechende Funktionen von der Karte bzw. deren DSR nicht unterstützt werden.

Das Byte >AA an der Adresse >4000 stellt den sogenannten Validation-Code dar, anhand dessen ein DSR-ROM als gültig bzw. vorhanden erkannt wird. Es ist durchaus denkbar, daß eine beliebige Erweiterung den Adreßbereich >4000 - >5FFF selbst nicht als DSR nutzt und den Validation-Code anders belegt (jeder andere Wert als >AA). Dabei ist jedoch ein prinzipieller Fehler vieler DSRLNK-Routinen zu beachten:

Wird auf einer aktiven CRU-Seite kein gültiger Validation-Code erkannt, so wird das Seiten-Bit (CRU-Bit 0) NICHT wieder mit SBZ >0 deaktiviert! Eine danach eingeschaltete DSR einer anderen Karte würde somit mit dieser ungültigen DSR parallel aktiviert, was in der Regel einen Kurzschluß auf dem PEB-Bus auslöst.

Das Byte an der Adresse >4001 ist frei nutzbar und stellt meist die Revisionsnummer der jeweiligen Firmware dar.

Die Bytes an den Adressen >4002, >4006 und >400E sind für spätere Erweiterungen reserviert. Wie am Beispiel etwa des MSX-Video-Chips zu sehen ist, sollte man reservierte Bits und Bytes immer unangetastet lassen, um späteren Konflikten aus dem Weg zu gehen.

PWRLNK

Die mit PWRLNK adressierte Liste leitet den sog. Power-Up-Link ein, welcher zur Initialisierung von Soft- und Hardware der jeweiligen Karte benutzt werden sollte. Die Struktur ist wie folgt:

```
PWRLNK    DATA   >0000      Zeiger auf Folgeeintrag (keiner)
          DATA   PWRPRG     Zeiger auf Start der Routine
          BYTE   >00        Namenlänge (Kein Name, also 0)
          EVEN              wir haben eine 16bit-CPU
          *
PWRPRG    ...               hier die Init-Sequenz
          *
          RT                Rücksprung in den Monitor
```

Die Power-Up-Routine kann die beim Aufruf aktuellen CPU-Register R0 bis R10 benutzen; R12 bis R15 sind tabu bzw. müssen vor dem Rücksprung wiederhergestellt werden. Daß R11 gesichert wird ist klar, oder?

Das VDP-RAM kann von der Adresse >0000 bis zu dem VDPHI-Zeiger an @PAD+>70 bzw. WP - >E0+>70 (normal >8370) belegt werden (was z.B. der Diskcontroller mit Vorliebe tut).

Im PAD-RAM können alle Adressen mit Ausnahme der Offsets >55, >6D und >C0 - >DF (normal >8355, >836D und >83C0 bis >83DF) frei belegt werden.

INTLNK

Der Interrupt-Link ähnelt dem Power-Up-Link im Listenaufbau. Ein gravierender Unterschied ist jedoch der, daß der die INTLNK-Tabelle bearbeitende Monitor aufgrund der neutralen Einbindung jeder DSR nicht erkennen kann, welche Karte den Interrupt auslöste (auf die vom PC bekannten Probleme einer sauberen Interrupt-Verwaltung soll hier nicht eingegangen werden).

Daher muß die Interrupt-DSR selbst überprüfen, ob der Interrupt von der eigenen Hardware ausgelöst wurde. Entsprechend muß in den Monitor entweder mit RT (Interrupt stammt von anderem Gerät) oder mit INCT R11, RT (Interrupt kam von der eigenen Karte) zurückgesprungen werden. Ein ggfs. selbst ausgelöster Interrupt muß natürlich von der DSR zurückgesetzt werden.

```
INTLNK DATA >0000     Zeiger auf Folgeeintrag (keiner)
       DATA INTPRG    Zeiger auf Start der Routine
       BYTE >00       Namenlänge (Kein Name, also 0)
       EVEN           s.o.
       *
INTPRG ...            hier die Interrupt-Sequenz. Sie muß prüfen, ob der Interrupt
                      von dieser Karte stammt, sonst NIXINT!
       *
INTEND INCT R11       Weitersuchen verhindern
NIXINT RT             Rücksprung in den Monitor
```

Eine Interrupt-Routine darf alle Register bis auf R0, R9 und R12 bis R15 des beim Aufruf aktuellen Workspace benutzen. Belegbare Adressen im PAD sind die Offsets >4A bis >6D (normal >834A bis >836D)

DSRLNK

Die Device-Service-Routine-Link-Table (schrecklicher Anglismus...) stellt gewissermaßen das Herz des I/O-Systems des TI-99/4A dar. Hier finden sich die meisten und bedeutendsten Programmsequenzen. Im Gegensatz zu INTLNK und PWRLNK muß bei DSRLNK (und auch bei SBRLNK) bereits vom Monitor entschieden werden, welches Programm der DSR das Richtige ist. Dementsprechend ist die DSRLNK-Tafel aufgebaut:

```
DSRLNK DATA  DSRL02      Sog. Link-Zeiger auf weitere Geräte
       DATA  DSR001      Einsprungadresse für dieses Gerät
       BYTE  4           Länge des Gerätenamens (ohne Punkt!)
       TEXT  'DSK1'      Gerätename
       EVEN              Word-Boundary
DSRL02 DATA  DSRL03      Link-Zeiger
       DATA  DSR002      Einsprungadresse
       BYTE  4           Namenlänge
       TEXT  'DSK2'      Name
       EVEN
DSRL03 DATA  >0000       Kein weiteres Gerät
       DATA  DSR003      Einsprungadresse
       BYTE  3           Namenlänge
       TEXT  'DSK'       Name
       EVEN
DSR001 ...               Programm für Gerät DSK1
       B     @DSREND
DSR002 ...               Programm für Gerät DSK2
       B     @DSREND
DSR003 ...               Programm für Gerät DSK
       B     @DSREND
*
DSREND INCT  R11
NXTDSR RT
```

Erstmals wird dabei in der Funktionshierarchie ein Weiterleiten einer Geräteanfrage erlaubt, wie später bei der Muster-DSRLNK-Routine zu sehen sein wird. Dabei kann nach erfolgter (oder auch nicht erfolgter) Bearbeitung einer I/O-Anforderung das DSRLNK mit gleichem Namen zur weiteren Suche oberhalb der aktuellen CRU-Seite aufgefordert werden. Hierzu zählt die DSRLNK-Routine einen Index mit, welcher die Anzahl erfolgter Bearbeitungen zählt. Dieser liegt in R1 vor, weshalb R1 des beim Aufruf aktuellen Workspace innerhalb der DSR nicht geändert werden darf bzw. vor deren Ende restauriert werden muß, wenn über NXTDSR in die DSRLNK-Routine zurückgesprungen wird. An diesem Index kann eine DSR erkennen, ob sie die erste oder folgende in der Liste ist.

Diese Funktion wird z.B. von den Original TI-RS232-Karten benutzt, deren DSR identisch ist, jedoch auf verschiedene CRU-Seiten gelegt werden kann. Hier prüft die jeweilige DSR, ob sie als RS232/1,2 oder RS232/3,4 bzw. analog als PIO/1 oder PIO/2 reagieren soll.

SBRLNK

Die Tafel der Zeiger auf Subroutine-Links ist gegenüber der DSRLNK-Tafel nur in Bezug auf den Gerätebzw. Funktionsnamen modifiziert. Der Gerätename ist in der Regel nur 1 Zeichen lang und beginnt beim ASCII-Code >10. Der Rücksprung wird identisch zur DSRLNK-Routine gehandhabt, wobei auch das Weiterleiten ohne INCT R11 möglich ist. Bitte ziehen Sie das Beispiel zur DSRLNK-Struktur heran.

Beispiel für eine DSRLNK-Routine

Eine DSRLNK-Routine erfüllt viele Funktionen:

- Sie sucht den gesamten CRU-Adressraum ab >1000 nach vorhandenen DSR-ROMs ab.
- Sie sucht im DSR-ROM nach dem passenden Gerätenamen.
- Sie ermittelt die passende Einsprungadresse der zum Gerät passenden DSR-Sequenz.
- Sie bereitet den Gerätenamen für die DSR auf.
- Sie sorgt für eine Verkettung ähnlicher oder gleicher DSRs.

Bei der Besprechung der DSRLNK-Routine sind zwei Seiten zu beachten:

die eigentliche DSRLNK-Seite und die DSR-Seite. Für dieses Zusammenspiel sind die Pointer an den Offsets >56 und >54 (normal >8356 und >8354) im PAD-RAM von zentraler Bedeutung.

- Der Zeiger an >8356 (W) zeigt beim Aufruf des DSRLNK auf das Namenlängebyte im PAB (Abweichung GPLLNK; siehe dort). Beim Einsprung in die DSR zeigt er auf das erste Byte hinter dem Gerätenamen (Beispiel: bei DSK1.MUSTER auf den Punkt hinter DSK1).
- Der Zeiger an >8354 (W) ist beim Einsprung in das DSRLNK undefiniert; beim Einsprung in die DSR enthält er die gesamte Länge des Namens abzüglich des Gerätenamens.

SBRLNK und GPLLNK

SBRLNK (Subroutine-Link) ist eine recht geringfügige Abwandlung des DSRLNK. Hier wird der Offset-Pointer in den DSR-Header auf eine andere Tabelle umgeleitet (Offset >A statt >8), womit auch etwas andere PAB-Strukturen Anwendung finden.

GPLLNK ist die Variante des DSRLNK für GROMs und ROMs im Bereich >6000 - >7FFF. Die Header der betreffenden Speicher sind dabei exakt so wie die der DSR im Bereich >4000 - >5FFF aufgebaut. GPLLNK sucht i.d.R. jedoch zusätzlich noch den normalen DSR-Bereich ab. Dabei bedient es sich des normalen TI-Monitors.

Der Aufruf erfolgt dann mit der Sequenz

```
CALL  >10
DATA  >0008
```

wobei zuvor ein PAB wie bei DSRLNK aufgebaut werden muß. Der wesentliche Unterschied besteht in der Tatsache, daß der Zeiger am PAD-Offset >56 (normal >8356) nicht wie bei dem DSRLNK auf das Namenlänge-Byte des PABs, sondern direkt auf das erste Zeichen des Namens zeigt.

Abweichung des Myarc-DOS im 9640

Der Myarc-9640-Geneve, zu Anfang als beinahe 100% kompatibel bezeichnet, inzwischen als potentiell inkompatibel zum TI-99/4A erkannt und mit einer eigenen Softwarelinie versehen, benutzt aus verschiedenen Gründen das TI-DSR-System nicht.

Wie aus der DDCC-1-DSR (dem Myarc Diskcontroller) zu ersehen, hat sich Myarc von je her nie an die TI-Vorgaben gehalten, was eine DSR darf und was nicht.

Bei der Konzeption des Geneve war es zum einen diese Grundhaltung, zum anderen innovativer Geist und wohl auch Angst vor den eigenen Holzhammer-DSRs, welche das alte DSR-Konzept als unbrauchbar erscheinen ließen. Man kann mit der notwendigen Sachkenntnis aus dem DDCC-1 Listing sehr leicht sehen, daß schon der erste Disketten-Schreibzugriff den Monitor und alle XOP's bzw. Interrupt-Vektoren des MDOS gelyncht hätte, würde dieser Controller mit der normalen DSR laufen.

Da man bei Myarc zudem keine Hardwaresynchronisation mittels DRQ und INTREQ beherrschte, wurde der Bussteuerung des Geneve dieses Feature nicht mitgegeben (vergessen?). Damit war es also unmöglich, den eigenen Controller einzubinden (DSR zu gefährlich) und alle anderen Controller konnten keinen Datentransfer synchronisieren, weil die TMS9995 einfach über den DRQ 'drüberlief'. Soweit zum Lapsus der Herren um Mr. Phillips. Die innovative Idee war, anhand eines simulierten DSR-ROMs (der DSR- Bereich >4000 - >5FFF ist beim Geneve üblicherweise nicht transparent) alle Geräteanfragen umzuleiten und von MDOS-eigenen Routinen abarbeiten zu lassen. Damit konnte man einerseits Kollisionen verhindern, da alle Software aus einem Guß war (was immer das bei Myarc auch implizieren mag...), andererseits war ein Austausch evtl. fehlerhafter Software (man wußte schon, wozu das gut war...) über eine MDOS-Revision bis hin zu dem DSRs möglich.

Der prinzipielle Nachteil ist aber, daß sich neue Karten nun nicht mehr selbst im System 'anmelden', indem sie einen neuen Namen einführen, sondern nun jedesmal beim Erscheinen einer neuen Karte das MDOS erweitert werden muß. Da dies i.d.R. nur durch 'Reverse Engineering' geschieht, sind die Wartezeiten vorprogrammiert.

Zum Zeitpunkt des Abschluß' dieses Kapitels war jedoch zu hören, daß Myarc das alte DSR-Konzept wieder entdeckt hätte...

Aufgrund der abweichenden Busstruktur des 9640 ist es zudem nicht generell möglich, doch die eingebauten DSRs anzusprechen, da man damit rechnen muß, daß diese auf einen TI-Bus hin ausgelegt sind (Low-High-Byte-Sequenz ist vertauscht, CRU-Adressraum benötigt weniger Bits, Wait-States erlaubt, Data-Setup-Time ist länger etc.).

Arten des Zugriffs auf die Diskettendaten

Am Beispiel der Dateiverwaltung (z.B. in BASIC) soll gezeigt werden, wie die logische Einheit 'FILE' aufgebaut ist und wie die Daten abgelegt bzw. abgerufen werden.

Das DOS des TI bringt von Hause aus sehr leistungsfähige Routinen zur Dateiverwaltung mit, die sonst nur im PC-Bereich, keinesfalls aber für einen Home-Computer selbstverständlich sind.

So bleibt dem Anwender als einzige Planung, die Struktur seiner Datei festzulegen und die Satzlänge zu spezifizieren. Für einfache Anwendungen reichen die Standard-Werte DISPLAY-Datenformat, VARIABLE Satzlänge von maximal 80 Bytes bereits aus.

Eine Datei kann, wenn sie existiert, ohne genaue Kenntnis von Datenart und Satzlänge im Eingabemodus eröffnet werden, und es können Daten bis zum logischen Ende der Datei gelesen werden.

Bei Eröffnung im Ausgabemodus können Daten in beliebiger Reihenfolge und fast beliebiger Anzahl geschrieben werden, wobei nur die Speicherkapazität der Diskette beschränkend wirkt.

Dateien, welche eine feste Länge der Einträge besitzen, können im sogenannten UPDATE-Modus eröffnet werden, bei dem mit einfachsten Kommandos beliebige Einträge der Datei gelesen und/oder geschrieben werden können.

Dateien jeder beliebigen Blocklänge können im APPEND-Modus eröffnet werden, womit dann eine Veränderung bereits vorhandener Daten nicht möglich ist. Am derzeitigen logischen Ende der Datei können jedoch neue Datensätze angefügt werden.

Zusätzlich lassen sich Dateien per Kennzeichnung vor dem Überschreiben schützen, was einen besonderen Schutz der enthaltenen Daten bewirkt.

Das Arbeiten mit Dateien (Files) ist also die für den Benutzer einfachste Art, das Medium Diskette als Datenspeicher zu verwenden. Er muß nur darauf achten, daß der Platz auf der Diskette ausreicht - an welchen Orten auf der Diskettenoberfläche sich die Daten befinden ist für ihn dann ohne Belang; er sieht die Datei als zusammenhängende logische Einheit.

Daß aber einige Probleme bei dem Aufbau eines Files entstehen können, soll ein Beispiel zeigen:

Angenommen, die von Ihnen verwendete Diskette ist bis auf wenige Sektoren belegt. Um Platz für die Erweiterung einer Datei zu bekommen, kopieren Sie ein auf der Diskette befindliches Programm auf eine andere Diskette. Nun können Sie die freigewordenen Sektoren benutzen, um Ihre Datei zu erweitern. Es ist nun aber im Allgemeinen so, daß sich dieses nun gelöschte Programm nicht unbedingt direkt hinter dem zu erweiternden File befand - es sind also an verschiedenen Orten auf der Diskette Sektoren frei geworden. Für Sie als Anwender des logischen Verbundes 'FILE' war das aber bisher kein Problem und soll es auch nicht werden.

Es ist aber die Aufgabe des Diskettenbetriebssystems festzustellen, ob der zusammenhängende freie Raum auf der Diskette ausreicht, oder ob das File in mehrere, physikalisch getrennte Teile aufgespalten werden muß.

Es kann also passieren, daß ein 20 Sektoren langes File die Sektoren 10 bis 14, 100 bis 110 und 88 bis 91 belegt, obwohl es als logische Einheit ein zusammenhängendes Gebilde darstellt.

File-Operationen sind gleichermaßen in TMS9900 Assembler möglich, besonders einfach durch die DSRLNK-Routine. Das File ist also der einfachste und zugleich eleganteste Weg der Datenverwaltung.

Die Programmteile des DOS (der Diskcontroller DSR), welche z.B. die File-Verwaltung auf Basic-Ebene steuern, werden Level-2-Routinen genannt.

Die in der Hierarchie folgende Zugriffsart auf die Diskettendaten ist der Sektorzugriff mit den Level-1-Routinen (siehe hierzu das Kapitel 'Einteilung der Diskette in Sektoren').

Hier werden den Zellen, aus denen die Diskette besteht, fortlaufende Nummern von 0 bis 359 (Hex 0 bis 167) bei einfacher Dichte und jeweils bis zum doppelten bei doppelter Dichte und doppelseitigen Disketten zugeordnet. Die maximale Anzahl ist also bei 40 Spur Laufwerken 1440 Sektoren, entsprechend einer Numerierung von 0 bis 1439.

Aber auch hier ist es für den Anwender ohne Belang, auf welcher Spur und welcher Seite der Diskette sich der Sektor befindet, nur seine 'Hausnummer' muß angegeben werden.

Der Zugriff auf bestimmte Sektoren ist nur in Maschinensprache möglich.

Die letzte Zugriffsart stellt der Spurzugriff über Level-0-Routinen dar. Für die Anwendung dieser Funktion muß die Hardware des Diskettencontrollers vollständig bekannt sein. Damit ist ein Spurzugriff im Gegensatz zum Sektor- und Filezugriff von der Hardware abhängig, so daß solche Programme nur auf bestimmten Systemen arbeiten oder aber angepaßt werden müssen.

Die Möglichkeiten und Probleme des Spurzugriffs werden in den Kapiteln 13, 14 und 15, neben anderen Aspekten der Level-0-Routinen, besprochen. Die universellen Zugriffsmöglichkeiten auf Level-0 des DOS, also da, wo eigentlich kein DOS mehr verfügbar ist, werden mit einem im Vergleich zu Level-2 und Level-1 sehr hohen Programmieraufwand erkauft.

Rolle der DSR-Software des Controllers

Das Kürzel DSR bedeutet 'Device Service Routine', was etwa mit 'Gerätesteuerungsroutine' übersetzt werden kann. Diese, bisher nur vom TI bekannte Art des Einbindens vollkommen unterschiedlicher und neuer Systemerweiterungen in das Betriebssystem des Rechners, macht seine besonders intelligente Konzeption deutlich. Durch eine genormte 'Softwareschnittstelle' ist es möglich, alle Erweiterungen identisch zu behandeln, indem lediglich deren Name geändert wird.

Eine Spezifizierung des Begriffs 'DSR' stellt das Kürzel 'DOS' dar, was 'Diskette Operating System' heißt und direkt mit 'Disketten Operations System' übersetzt werden kann. Diese besondere DSR steuert alle notwendigen Abläufe, die zum Betrieb einer Diskettenstation gehören.

Dies sind z.B.

- Formatieren neuer Disketten,
- Verwalten von Daten in Files, etc.

Eine genaue Kenntnis aller Befehle, aus denen sich diese Software zusammensetzt, ist nicht notwendig, da es Unterschiede zwischen den Controllern gibt, die man bei ausschließlicher Beachtung der notwendigen Pointer ignorieren kann.

Die DSR macht also die eigene Software unabhängig von dem Typ der vorhandenen Hardware. Es ist damit problemlos möglich, bestimmte Sektoren einer Diskette mit dem gleichen Programm auf einem TI, CorComp, Myarc oder Atronic Diskcontroller zu lesen oder zu schreiben.

Einteilung der Diskette in Sektoren

Die Daten, welche auf der Diskette abgelegt werden, befinden sich dort nicht 'am Stück', sondern werden in kleine Segmente zu je 256 Byte aufgespalten. Diese Einteilung orientiert sich an der 'Parzellierung' der Diskette nach dem Formatieren. Hier werden in jeder Spur Marken auf die Diskette gesetzt ähnlich denen, die man zur synchronen Steuerung eines Diaprojektors über ein Tonband benutzt. Erst danach ist es möglich, gezielt auf bestimmte Bereiche der Diskette zuzugreifen, da erst jetzt eine Art 'Wegenetz' existiert.

Die Länge von 256 Bytes pro Sektor stellt einen Kompromiß dar zwischen maximaler Ausnutzung des Speicherraums auf Diskette und minimaler Ladezeit. Das ist deshalb so, weil ein einziger Sektor nur Daten eines einzigen Files enthalten kann und keine Mischbelegung zulässig ist. Es ist daher in der Regel so, daß ein Sektor z.B. eines DIS/VAR 80 Files an seinem Ende maximal 81 Zeichen unbenutzt lassen muß, weil vielleicht der letzte Eintrag gerade 80 Zeichen lang war. Dieser Eintrag würde dann in den nächsten freien Sektor geschrieben, der dann bis auf 82 Zeichen (incl. Längenbyte!) leer wäre. Bei einer Sektorlänge von z.B. 1024 Byte wären dann 942 Byte unbenutzt! Ein Sektor mit 1024 Bytes Länge wird aber schneller geladen, als vier mit je 256 Bytes.

Wie diese Zahlen zustande kommen, wird in den folgenden Kapiteln besprochen.

Dieses Wissen um die Belegung der Sektoren ermöglicht es, bei speicherintensiven Dateien den verfügbaren Platz auf der Diskette optimal zu nutzen. Zwar wird bei der Beschreibung der Datensektoren noch darauf eingegangen, hier aber ein erster Hinweis. Bei Datensätzen variabler Länge wird ein Byte angefügt, das die Länge des Datensatzes jeweils neu angibt. Man sollte die Datensatzlänge dann so wählen, daß eine bestimmte Anzahl von Datensätzen inclusive des Längenbytes 256 ergibt. Bei FIXED-Dateien fehlt dieses Längenbyte, die Berechnung einer optimalen Satzlänge ist analog zu VAR-Dateien. Überhaupt ist die optimale Relation zwischen Platzökonomie und Ladezeit nur bei FIXED-Dateien gegeben. Besonders deutlich wird dies beim Suchen eines Datensatzes. Dazu später mehr.

Die Abfolge der Sektoren einer Spur ist nicht numerisch aufsteigend, näheres hierzu in den Abschnitten 'Formatierung von Disketten' und 'Aufzeichnungsverfahren'.

Neben der 'Feinstruktur' der Sektoren ist die Diskette in Spuren aufgeteilt. Diese sind konzentrische Ringe, deren Umfang nach der Diskettenmitte hin abnimmt. Die Datenspuren hängen also nicht in der Art einer Spirale zusammen, sondern sind voneinander getrennt. Über die Spur, in der ein Sektor zu finden ist, braucht sich jedoch kein Programmierer den Kopf zu zerbrechen, solange er sich an die fortlaufende Sektor-Numerierung hält.

Aus der Sektorstruktur resultierende Kompatibilität

In allen für den TI verfügbaren Diskcontrollerschaltungen, ob als Karte für die P-Box oder eigenständig, werden Floppy-Disk Controller/Formatter IC's nach dem Muster der Firma Western Digital eingesetzt, Typenbezeichnung 1771 oder 1773. Diese integrierten Schaltungen bzw. die formatkompatiblen NEC 765 werden

auch in vielen anderen Rechnern, so z.B. frühe IBM-Modelle, eingesetzt. Da die IC's bestimmte Sektoren selbständig suchen und an die CPU übergeben, muß das verwendete Diskettenformat bestimmte Bedingungen erfüllen. Das bedeutet praktisch, daß bei Übereinstimmung der Sektorlänge alle Disketten all jener Rechner gelesen werden können, die einen solchen Chip als Controller-IC besitzen. Zwar wird der Sektorinhalt von jedem Rechner anders interpretiert, mittels eines Konversionsprogramms ist es aber denkbar, z.B. Textdateien oder Speicherauszüge mit Besitzern anderer Systeme zu tauschen, was in manchen Fällen sicherlich interessant sein kann.

Die Controller von Atronic oder CorComp verwenden bei doppelter Dichte mit 18 Sektoren pro Spur ein Format, das den Mindestanforderungen des IBM System 34 Formats (MFM) entspricht. Hier stimmt auch die Sektorlänge von 256 Bytes überein. Das Original TI-Format bei Single Density ist jedoch in keiner Weise mit dem IBM 3740 Format (FM) kompatibel.

Leider erweist sich die Praxis als etwas schwieriger als das nun vielleicht aussehen mag. Die PC's von IBM mit den 360K-Laufwerken verwenden nämlich eine Abwandlung des MFM-Systems 34 mit 9 Sektoren pro Spur und 512 Bytes pro Sektor. Zusätzlich wechseln die logischen Spurnummern immer die Seiten, also gerade Nummern auf der Ober- und ungerade Nummern auf der Unterseite. Hier ist die DOS-Routine 'Sektor lesen' nicht anwendbar, sie muß neu geschrieben werden.

Formatierung von Disketten

Der Prozeß der Formatierung muß von jeder neuen Diskette durchlaufen werden, bevor sie als Speichermedium benutzt werden kann. Auf einer neuen Diskette befinden sich im Allgemeinen keine sinnvollen Informationen, außer den Bitfolgen, die bei der Einzelprüfung aufgespielt und gelesen wurden. Aufgabe des Formatierens ist es nun, Marken auf der Diskette aufzubringen, anhand derer sich die einzelnen Informationsblöcke, die Sektoren, wiederfinden und eindeutig erkennen lassen. Dieser Vorgang wird spurweise vorgenommen, wobei in jeder Spur die Sektoren mit Zahlen von 0 bis 8 bei Single Density und von 0 bis 17 bei Double Density (0 bis 15 bei Myarc DD) bezeichnet werden.

Neben den bekannten je 256 Bytes eines Sektors, die das 'Datenfeld´ darstellen, befinden sich noch folgende Daten auf der Diskette, die nur für den Floppy-Controller wichtig sind:

Am Anfang einer Spur befinden sich Synchronisierzeichen, ähnlich dem hohen Ton zu Anfang eines Cassettenfiles, anschließend folgen die Sektoren, die wie folgt aufgebaut sind:

- Startbyte für den Prüfsummengenerator, Kennung des Adressfeldes.
- Adreßfeld, bestehend aus Spur-, Seiten- und Sektornummer.
- Sektorlängenkennung,
- Prüfsumme des Adressfeldes mit anschließender Datenpause.
- Startbyte für den Prüfsummengenerator, Kennung des Datenfeldes.
- 256 Datenbytes, beim Formatieren zu >E5 gesetzt.
- Prüfsumme der Daten mit anschließender Pause zum nächsten Sektor.
- nächste Sektoren wie zuvor,
- Ende der Spur mit Taktsignalen.

Wie zu sehen ist, geht ein respektabler Anteil des prinzipiell verfügbaren Speicherplatzes auf der Diskette für Kennung und Prüfung den eigentlichen Daten verloren. Daraus erklärt sich der Unterschied zwischen der Diskettenkapazität formatiert und unformatiert. Eine genauere Beschreibung der Spurstruktur erfolgt in Kapitel 13.

Die Reihenfolge der Sektoren in einer Spur ist nicht auf- oder absteigend, die Ordnung unterliegt bestimmten Zwängen. Hierzu ein Gedankenbeispiel:

Es soll ein Programm geladen werden, welches aus mehreren Sektoren innerhalb einer Spur besteht. Nach Lesen des ersten Sektors, z.B. Nummer 3, muß dessen Inhalt abgespeichert und dann der nächste gelesen werden. Würde nun der Sektor 4 unmittelbar auf Sektor 3 folgen, dann hätte sich während der Zeit, in der Sektor 3 verschoben wurde, die Diskette natürlich weitergedreht und der Kopf befände sich beim nächsten Zugriff bereits über Sektor 5. Dann müßte nahezu eine ganze Umdrehung der Diskette abgewartet werden, um wieder auf Sektor 4 zu treffen. Dieses 'Spiel' würde sich bei jedem neuen Sektor wiederholen und die Zeit zum Laden dieses Programmes unnötig verlängern.

Hier bedient man sich nun eines Verfahrens, das mit 'Sector Interlace' bezeichnet wird. Die Abfolge der Sektoren ist nun nicht mehr in numerischer Reihenfolge, sondern verschachtelt. Am Beispiel einer Single Density Diskette sei dies kurz dargestellt:

Die Reihenfolge der Sektoren bei SD ist im Allgemeinen

0, 7, 5, 3, 1, 8, 6, 4, 2

d.h. auf Sektor 0 folgt Sektor 7 etc. Verfolgt man die Sektoren mit fortlaufenden Nummern, so ist erkennbar, daß zwischen zwei logisch aufeinanderfolgenden Sektoren 3 Stück mit abweichender Numerierung liegen. Die Anzahl der 'eingeschobenen' Sektoren wird mit 'Interleaving Factor' bezeichnet, der in diesem Fall 3 beträgt. Beim fortlaufenden Lesen innerhalb einer Spur kann nun die Behandlung der Daten des vorangegangenen Sektors während des 'Vorbeilaufens' von 3 Sektoren erfolgen, zum Lesen der gesamten Spur werden dann 4 Umdrehungen der Diskette benötigt. Bei Double Density wird ein Interleaving Faktor von 4 verwendet, da hier die Wartezeit aufgrund der höheren Datenrate sonst zu kurz wäre.

In gleichem Maß erhöht sich jedoch die Anzahl der Umdrehungen der Diskette die nötig sind, alle Sektoren in numerischer Reihenfolge (das ist der wahrscheinlichste Fall) zu lesen. Diese Anzahl berechnet sich aus Interleave-Faktor + 1.

Wie werden nun Disketten formatiert?

Man verwendet dazu eine Level-1-Routine.

Im VDP-RAM wird ein PAB mit den Daten >0111 aufgebaut. Hierbei handelt es sich um eine Subroutine mit Namen '>11' und der Namenlänge >01. Dementsprechend muß der Namelength Pointer an @>8356 auf das Byte >01 im VDP-RAM zeigen. An der Adresse >834C (Byte) steht die Laufwerknummer, in >834D (Byte) steht die Anzahl der Spuren, die formatiert werden sollen. Einige Diskcontroller (Myarc) erlauben hier nur den Wert >28 (dez. 40), andere formatieren auch 1 bis 80 Spuren. Nach dem Formatieren steht hier die Anzahl der Sektoren pro Spur, mit der formatiert wurde. An >834E steht die Adresse eines VDP-Pufferbereichs, der mindestens 6,5 KByte fassen können muß (etwa die Hälfte bei einfacher Dichte). An der Adresse >8350 steht im High-Byte die Dichte, mit der formatiert werden soll (>02 - doppelte Dichte, >01 - einfache Dichte), im Low-Byte steht die Anzahl der Seiten, die formatiert werden sollen (>01 - einseitig, >02 - doppelseitig.

Nachdem alle Vorbereitungen getroffen wurden, wird der Befehl mit einem Unterprogrammaufruf der Art

```
BLWP  @DSRLNK
DATA  >A
```

ausgeführt. Da das DATA von der DSRLNK-Routine als Pointer-Offset in das DSR-ROM verwendet wird, werden nun keine DSR's sondern Unterprogramme angesprungen, ähnlich z.B. CALL FILES, was die selbe Tafel verwendet.

Nach einer solchen Formatierung kann die Diskette aber noch nicht verwendet werden (außer für Sektorkopierprogramme), es müssen erst die Sektoren 0 und 1 mit für das DOS wichtigen Daten über die Diskette beschrieben werden. Verwandte Level-1-Routinen werden in Kapitel 10 vorgestellt.

Zugriff auf die Sektoren

Der Zugriff auf die einzelnen Sektoren einer Diskette ist in Maschinensprache mittels einer Level-1-Routine recht einfach möglich.

An der Adresse >834C befindet sich im High-Byte die Nummer des gewünschten Laufwerks, im Low-Byte befindet sich der sogenannte I/O-Opcode (>00 - Sektor schreiben, >01 oder generell <> 0 - Sektor lesen).

An der Adresse >834E steht die Adresse des 256 Byte-Puffers im VDP-RAM. An der Adresse >8350 steht die Nummer des gewünschten Sektors. Im High-Byte dieses Wortes wird nach Rückkehr der Fehlercode übergeben (>00 - kein Fehler).

Ein PAB wird mit >0110 aufgebaut, >8356 zeigt auf das Byte >01.

Die Ausführung erfolgt wie beim Formatieren mit

```
BLWP  @DSRLNK
DATA  >A
```

Nach dem Lesen befinden sich die Daten im VDP-Puffer und zwar selbst dann, falls beim Lesen ein Prüfsummenfehler auftrat. Es finden sich nur dann vollkommen sinnlose Daten, wenn der passende Sektor überhaupt nicht gefunden wurde.

Beim Schreiben von Sektoren muß beachtet werden, daß nur ein Schreibschutz an der Diskettenhülle Fehler unterbinden kann; ein etwaiger Fileschutz, der per Software das Überschreiben verhindern soll, ist bei Sektor-Operationen ohne Effekt!

Art und Inhalt der Sektoren

Wer gelernt hat, direkt auf Sektoren zuzugreifen, wird sich für dieses Themengebiet sicher besonders interessieren.

Prinzipiell gibt es beim TI 99/4A vier verschiedene Sektorstrukturen:

1. Sektor 0, auch Root Sector genannt,
2. Sektor 1, die Directory-Sektor Liste,
3. die Directory Sektoren, einer für jedes File,
4. die Datensektoren, welche die reinen Daten der Files beinhalten.

Zu 1.: Sektor 0 beinhaltet alle wichtigen Informationen über die Diskette. Dies sind: Diskname, Anzahl der vorhandenen Sektoren, Anzahl der Spuren, Anzahl der Seiten, Schreibdichte, Anzahl der Sektoren pro Spur, Kopierschutz sowie die sogenannte 'Sector-Bitmap', in der belegte Sektoren markiert sind, damit diese nicht aus Versehen überschrieben werden. Vorgesehen war ein Datumsfeld, das beim Formatieren geschrieben werden sollte; in Ermangelung einer Hardwareuhr sind diese Bytes üblicherweise zu null gesetzt.

Aufbau von Sektor 0

Anhand eines Ausdrucks mit einem Disk-Editor soll der Inhalt von Sektor Null analysiert werden.

```
Adr.  0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  ASCII
------------------------------------------------------------------
0000  47 52 41 50 48 49 4B 5F 5F 32 01 68 09 44 53 4B GRAPHIK__2*h*DSK
0010  20 28 01 01 00 00 00 00 00 00 00 00 00 00 00 00 (**************
0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 **************
0030  00 00 00 00 00 00 00 00 FF FF 47 FF FF FF FF FF **********F*****
0040  FF FF FF FF FF FF FF FF FF A5 FF FF FF FF FF FF **************
0050  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF **************
0060  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF **************
0070  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF **************
0080  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF **************
0090  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF **************
00A0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF **************
00B0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF **************
00C0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF **************
00D0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF **************
00E0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF **************
00F0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF **************
```

- Bytes 0 bis 9: Diskettenname, bei weniger als 10 Zeichen mit Blanks aufgefüllt.
- Bytes A und B: Gesamtzahl der Sektoren, hex 168, dez 360. Hier können, abhängig von der Formatierung, andere Werte stehen.
- Byte C: Anzahl der Sektoren pro Spur, hier 9, auch möglich sind 12 und 10 bei doppelter Dichte.
- Bytes D bis F: Formatierungsindex. Steht hier nicht 'DSK' dann hält ein Disk-Manager die Diskette für nicht formatiert.
- Byte >10: Protection Flag. Steht hier ein Blank, so ist die Diskette zum Kopieren freigegeben, steht hier ein 'P', ist ein normales Kopieren unmöglich.
- Byte >11: Anzahl der Spuren pro Seite, hier hex 28, dez 40.
- Byte >12: Seitenanzahl, >01 - einseitig, >02 - doppelseitig.
- Byte >13: Schreibdichte, >01 - einfache Dichte, >02 - doppelte Dichte.
- Byte >14: Sektoren Pro Cluster (-1), >00 - direkte Adressierung, >01 - 2 Sektoren pro Cluster
- Bytes >15 - >37: Nicht benutzt. Diese Bytes waren für Erweiterungen vorgesehen und sind zu null gesetzt.
- Bytes >38 - >FF: Sektor Bitmap. Jedes gesetzte Bit gibt einen belegten Sektor an. Diese Karte ist 200 Bytes lang und kann daher die Belegung von bis zu 1600 (dez) Sektoren verwalten, entsprechend 400 KByte Daten. Jedes Byte beschreibt 8 Sektoren wie folgt:

```
      Bit  7 6 5 4 3 2 1 0
Sektor X + 7 6 5 4 3 2 1 0
```

Die Einträge in der Sektor-Bitmap sind nur bis zu einer bestimmten Position gültig. Dies liegt einfach daran, daß bei 360 freien Sektoren eben nur 45 Bytes notwendig sind, um alle Sektoren zu beschreiben. Bei 1440 Sektoren sind dann nur 180 der 200 Bytes notwendig. Die Sektor-Bitmap enthält somit immer Bytes, die keinen vorhandenen Sektor beschreiben, also diejenigen Bytes, die auf das letzte gültige Byte folgen. Um Fehlfunktionen des DOS zu vermeiden, müssen jedoch die nicht benutzten Bytes >FF enthalten. Ist dies nicht der Fall, so kann es passieren, daß Sektoren außerhalb der Diskette gesucht werden, was logischerweise eine Fehlermeldung erzeugt.

Beim Arbeiten mit der Sektor-Bitmap sind zwei Fragen zu beantworten:

- Erstens: Ist ein Sektor, dessen Nummer bekannt ist, belegt oder nicht, und
- Zweitens: Welche Sektoren beschreibt ein bestimmtes Byte in der Sektor-Bitmap?

Beispiel zur ersten Frage: Es soll geprüft werden, ob Sektor 138 (dez) belegt ist. Jedes Byte der Sektor-Bitmap beschreibt, wie bereits gesagt, 8 Sektoren. Um das entsprechende Byte zu finden, muß die Nummer des Sektors durch 8 geteilt werden. 138 geteilt durch 8 ergibt 17 Rest 2. Der ganzzahlige Wert, 17, gibt an, in welchem Byte der Sektor-Bitmap sich das dem Sektor zugeordnete Bit befindet, der Rest gibt die Nummer des Bits in diesem Byte an. Die Dezimalzahl 17 ist gleich der Hex-Zahl 11. Da der Anfang der Sektor-Bitmap bei Byte >38 beginnt, muß >11 zu >38 addiert werden, um unser gesuchtes Byte zu finden. Es steht also an der Position >38 + >11 = >49. Byte >49 enthält >A5. Der Wert >A5 muß nun in eine Dualzahl umgewandelt werden. Man erhält

>A5 -> 1 0 1 0 0 1 0 1

Nun wird Bit 2 gesucht und festgestellt, daß dieses Bit gesetzt ist - der Sektor 138 ist also belegt!

Beispiel zur zweiten Frage: Was bedeutet das Byte an Position >3A?

Byte >3A enthält hex 47. Das Byte >3A ist 2 Byte vom Anfang der Sektor-Bitmap entfernt, das bedeutet, daß die Nummer des niedrigsten erfaßten Sektors 2 mal 8 ist, also dez 16 oder hex 10. Wandelt man nun den Inhalt des Bytes an >3A in die duale Schreibweise um, so erhält man

```
hex 47 -> 0 1 0 0 0 1 1 1 dual
```

Setzt man obiges Schema an, dann folgt:

```
Sektoren frei     x   x x x
Sektoren belegt     x       x x x
                    ¦       ¦ ¦ ¦
```

```
¦       ¦ ¦ +--- Sektor Nr. 10 hex
¦       ¦ +----- Sektor Nr. 11 hex
¦       +------- Sektor Nr. 12 hex
+--------------- Sektor Nr. 16 hex.
```

Wichtig ist noch zu wissen, daß beim Lesen von Files der Inhalt von Sektor 0 unwichtig ist, die Daten werden nur beim Schreiben verwendet!

Aufbau von Sektor 1

In Sektor 1 befinden sich nur Zahlen (12 Bit Worte). Jede Zahl gibt die Nummer eines Sektors an, in dem sich File-Daten befinden. Damit ist Sektor 1 das eigentliche Directory, eine Liste, in der die Adressen sämtlicher gültiger File-Directory-Sektoren zu finden sind. Die Reihenfolge der Zahlen scheint nicht geordnet zu sein, sie ist es aber trotzdem. Es darf als bekannt gelten, daß die Filenamen beim Aufbau eines Directories in ihrer alphabetischen Reihenfolge auftreten. Dies wird dadurch erreicht, daß dieser Directory Sektor auf die verschiedenen File-Directory Sektoren derart verweist, daß deren Namen in alphabetischer Reihenfolge stehen. Es spielt also keine Rolle, in welcher Reihenfolge die einzelnen Files die Diskette bevölkern, lediglich Sektor 1 wird jedesmal neu sortiert.

Hier nun ein Beispielausdruck eines solchen Sektor 1.

```
Adr.  0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  ASCII
-------------------------------------------------------------------
0000  00 07 00 08 00 02 00 09 00 06 00 0A 00 03 00 0B ****************
0010  00 04 00 0C 00 05 00 00 00 00 00 00 00 00 00 00 ****************
0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ****************
0030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ****************
0040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ****************
0050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ****************
0060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ****************
0070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ****************
0080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ****************
0090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ****************
00A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ****************
00B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ****************
00C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ****************
00D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ****************
00E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ****************
00F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ****************
```

- Wort 1: 0007 File-Directory Sektor 'ASSM1'
- Wort 2: 0008 File-Directory Sektor 'ASSM2'
- Wort 3: 0002 File-Directory Sektor 'BFIX'
- Wort 4: 0009 File-Directory Sektor 'BSCSUP'
- Wort 5: 0006 File-Directory Sektor 'BUGOUTO'
- Wort 6: 000A File-Directory Sektor 'DEBUG'
- Wort 7: 0003 File-Directory Sektor 'DSKNAV'
- Wort 8: 000B File-Directory Sektor 'EDIT1'
- Wort 9: 0004 File-Directory Sektor 'EDITOR'
- Wort A: 000C File-Directory Sektor 'SAVE'
- Wort B: 0005 File-Directory Sektor 'SBUG'
- Wort C: 0000 Ende der Directory Einträge.

Wird das erste Wort zu null gesetzt und alle weiteren nach hinten verschoben, so wird kein Directory erstellt (dies betrifft die gängigen Diskmanager wie das TI Modul, DM 1000, Atronic und Basic Katalogprogramme), es werden aber alle Files noch geladen. Werden zwei Worte vertauscht, so kann das File mit im Alphabet vorangestelltem Filenamen nicht mehr gelesen werden! Prinzipiell ist es sogar fast vollkommen unerheblich, wie viele Nullwörter zwischen zwei Sektornummern stehen, solange die alphabetische Reihenfolge eingehalten wird und der Suchalgorithmus nicht gestört wird.

Ein Blick auf den beim Suchen eines Files verwendeten Algorithmus soll diese Besonderheit aufzeigen.

Gesucht wird ein File immer dann, wenn sein Name und die Diskettenstation bekannt ist, wo das File zu finden ist. Der betreffende Sektor 1 wird gelesen und sodann von der MITTE dieses Sektors aus, also dem 64. Wort, eine Sektornummer ermittelt. Wird nun ein Nullwort gelesen, so wird die Schrittweite halbiert und 32 Byte tiefer gesucht, bei erneuter Null ggfs. 16 Byte tiefer usw. Wird eine Sektornummer gefunden, so wird der entsprechende FDS gelesen und der Filename mit dem angegebenen verglichen. Je nachdem, wie der Namensvergleich endet, kann aus den letzten beiden Bytes der Filenamen ermittelt werden, in welcher Richtung im Alphabet und damit im Sektor 1 weitergesucht werden muß; dabei wird permanent die Schrittweite halbiert. Das Suchen endet, wenn entweder der richtige FDS gefunden wurde, oder wenn die Schrittweite der Interpolation Null ist (File nicht gefunden).

Es handelt sich also um eine klassische Interpolation, oder aber um das Absuchen eines ausgeglichenen binären Baumes, wobei jede Sektornummer einen Knoten darstellt, von dem aus in der alphabetisch richtigen Richtung weiter verzweigt wird. Ein Nullknoten (-sektor) gibt dabei die Richtung 'nach unten' bzw. 'nach links' an.

Mit dieser Information läßt sich errechnen, daß ein beliebiger Filename, also der FDS eines beliebigen Files nach maximal 7 Versuchen gefunden wird, da der Logarithmus von 128 zur Basis 2, dem binären Zahlensystem, 7 ist.

Der weit verbreitete Disk Manager 1000 von Bruce Caron verwendet nicht die binäre Suche. Daher benötigt dieses Programm sehr viel mehr Zugriffe, um ein File zu finden bzw. umzukopieren. Bei einer großen Anzahl Files auf der Zieldiskette kann daher das DM-II Modul oder jedes andere Programm, das auf Controllerroutinen zugreift, Geschwindigkeitsvorteile gegenüber DM 1000 verbuchen.

Aufbau der File-Directory Sektoren (FDS)

Im Gegensatz zu den Sektoren 0 und 1, deren Bedeutung schon aus der Nummer zu erkennen ist, ist die Bedeutung der folgenden Sektoren 2 bis zum Ende der Diskette prinzipiell nicht festgelegt. Zwar werden die Sektoren 2 bis 21 (hex) bevorzugt mit File-Directory Sektoren belegt, je nach Belegung der Diskette können aber hier 'unten' auch Datensektoren zu finden sein, wie sich auch an höheren Sektoradressen File-Directory Einträge finden können. Zwar läßt es sich nicht für jeden Fall so einfach sagen, doch sind auf Disketten mit vielen kleinen Files auch mal FDS über >21 zu finden, große Dateien können das TI-DOS schon mal zwingen, Datensektoren in den FDS-Bereich zu verlegen. Wie Sektor 0 den Zustand der gesamten Diskette wiederspiegelt, so besitzt jedes File, ob nun Datei oder Programm, einen eigenen Sektor, der den Zustand des Files beschreibt.

Dieser jeweilige File-Directory Sektor gibt an, um welchen Typ von File es sich handelt, wie viele und welche Sektoren es belegt usw. Damit sind diese Sektoren wohl die interessantesten Objekte. Bei der Bestimmung der Anzahl der von einem File belegten Sektoren wird dieser Sektor immer dazugezählt.

Nun wieder ein Beispielausdruck eines solchen Sektors.

```
Adr.  0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  ASCII
------------------------------------------------------------------
0000  41 44 52 45 53 53 39 39 20 20 00 00 8A 01 00 3D ADRESS99  *****=
0010  EA FE 3D 00 00 00 00 00 00 00 00 00 0B C3 03 00 jß=**********C**
0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ****************
0030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ****************
0040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ****************
0050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ****************
0060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ****************
0070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ****************
0080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ****************
0090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ****************
00A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ****************
00B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ****************
00C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ****************
00D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ****************
00E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ****************
00F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ****************
```

Die Bytes und ihre Bedeutung:

Bytes >0 - >9:

Filename. Die Maximallänge ist wie bekannt 10 Bytes. Fehlende Bytes werden mit Blanks (>20) aufgefüllt. Es sind alle ASCII-Codes zulässig!

Bytes >A, >B:

Reserviert, zu >0000 gesetzt.

Byte >C:

File Status, gibt den Filetyp und einen etwaigen Schreibschutz an. Die Bedeutung der einzelnen Bits ist wie folgt:

	0	**1**
Bit 7	FIXED – Feste Satzlänge	VARIABLE – variable Satzlänge
Bit 6	nicht benutzt	
Bit 5	nicht benutzt	
Bit 4	*Archivbit bei MYARC-Controller*	
Bit 3	Kein Schreibschutz	Schreibschutz gesetzt
Bit 2	nicht benutzt	
Bit 1	DISPLAY	INTERNAL
Bit 0	Daten-File	Program-File

Byte >D:

Anzahl (VAR: Minimalzahl) der Datensätze (Records) pro Sektor. Im Gegensatz zu anders lautenden Veröffentlichungen wird dieses Byte sowohl bei VAR wie auch bei FIXED Dateien verwendet. Bei PROGRAM-Files ist dieses Byte immer >00.

Bytes >E, >F:

Gesamtzahl der vom File belegten Datensektoren. Hierzu muß aber noch der File-Directory Sektor addiert werden!

Byte >10:

EOF-Offset im letzten Sektor. Dieses Byte wird nur benutzt bei PROGRAM-Files und Dateien mit variabler Satzlänge. Es gibt die Anzahl der Bytes im letzten belegten Sektor an und berechnet sich bei P-Files aus Länge MOD 256 oder einfach durch Ausmaskieren der beiden niederwertigsten Hex-Ziffern. Alle satzorientierten Dateien werden mit einem EOF-Symbol abgeschlossen, und zwar mit dem Byte >FF, das sich bei VAR-Dateien am Ende jedes Sektors befindet. PROGRAM-Files werden entgegen anderslautenden Angaben nicht mit >AA abgeschlossen, ein bestimmtes Kennungsbyte existiert nicht, da ja bereits dieser Zeiger auf das letzte Byte diese Information liefert. Damit ist eine Möglichkeit gegeben, die Länge von PROGRAM-Files auf das Byte genau zu bestimmen.

Byte >11:

Satzlänge, wie sie bei der File-Eröffnung spezifiziert wurde. Bei P-Files zu Null gesetzt.

Bytes >12, >13:

VAR-Dateien: Anzahl der mit Daten belegten Sektoren.

FIXED-Dateien: Anzahl der Datensätze (Records) im File.

Beide Bytes sind vertauscht, Low-Byte zuerst, daher müssen sie ge'swap't werden.

Bytes >14 bis >1B:

Reserviert. Diese 8 Bytes sind zu null gesetzt.

Vermutlich werden beim CC40 und dem TI 99/5 die Bytes >14 und >15 für die Dateilänge in Bytes für Basic-Programme verwendet.

Bytes >1C und folgende:

Dieser Bereich wird mit File-Link Map bezeichnet, gelegentlich auch mit Cluster-Block. Hier wird in Blöcken zu je 3 Byte aufgezeichnet, aus welchen örtlich getrennten Bereichen das File besteht. Hierzu wird in jedem 3 Byte-Block angegeben, wo ein Teilblock des Files beginnt und bis zu welchem Sektor-Offset der Teilblock reicht. Dabei hat der erste Datensektor logischerweise den Sektor-Offset 0. Die Reihenfolge der Nibbles ist vertauscht. Folgendes Schema ist dabei anzuwenden:

Vorliegende Reihenfolge: Ss2 Ss1 So1 Ss3 So3 So2

wobei Ss= Startsektor und So= Sektor-Offset bedeuten.

Diese Reihenfolge muß in Ss3 Ss2 Ss1 und So3 So2 So1 umgestellt werden.

Wie man sieht, ergeben sich 12 Bit Worte. Mit dem Beispiel des o.a. Sektors ergibt sich:

Vorher 0B C3 03, nachher >30B und >03C.

Da die nachfolgenden Bytes nur Nullen sind, ist zu erkennen, daß das betrachtete File aus nur einem Cluster (Block) besteht, der im Sektor >30B beginnt und an den sich >3C Sektoren anschließen (der Sektor mit dem Offset 0, also die Nummer >30B zählt natürlich mit). Der Sektor-Offset ist bei verteilten Files fortlaufend, so daß immer der letzte Offset-Wert von dem folgenden abgezogen werden muß, um die Anzahl der Sektoren im Cluster zu erhalten. Diese File-Link Map kann fast bis zum Ende des File-Directory Sektors wachsen! Fast deshalb, weil ein >000000-Tripel als Endesymbol nötig ist.

Ausnutzung der Bytes am Ende eines FDS

Es wird nur äußerst selten passieren, daß die Block-Link Map bis ans Ende eines File-Directory-Sektors reicht. Daher ist es denkbar, daß man z.B. die letzten 80 Bytes eines FDS benutzt, um dort zusätzliche Daten zum File unterzubringen. Das ist jedoch nicht ganz ohne Risiko, denn wenn die Block-Link Map dann irgendwann in diesen Bereich vorstößt, was bei jeder normalen Schreiboperation in das betreffende File passieren kann, dann werden nicht dazugehörige Datensektoren überschrieben.

Der dabei greifende Mechanismus ist bei Kenntnis der FDS-Struktur recht einfach zu verstehen:

Das beschriebene Szenario soll eine FIXED-Datei zugrunde legen.

Das I/O-System versucht bei einer APPEND- oder RANDOM-WRITE-Operation den Platz für einen neuen Eintrag zu ermitteln. Dabei wird anhand der Anzahl der Datensätze und deren Länge den derzeitigen EOF-Offset berechnet. Dann wird in der Block-Link-Map der EOF-Offset ermittelt, der bis dahin nicht existierte. Üblicherweise trifft die DSR nun auf einen >000000-Eintrag und fordert einen neuen Cluster an, der korrekt über Sektor 0 (Bitmap) zugewiesen wird, sofern Platz dafür ist. Im Falle des falschen Clusterblocks wird, basierend auf einem total falschen Startsektor, ein File-Offset als zugewiesen angesehen, der es de facto nicht ist. Die DSR läßt sich in diesem Fall den Sektor nicht (mehr) zuweisen und schreibt folglich auf einen nicht zugewiesenen Sektor, der nach Murphy ein immens wichtiges File ruiniert. Wenn man Glück hat, liegt der Sektor-Offset außerhalb der Diskette, was einen Fehler auslöst.

Auf Disketten, auf denen aber nichts mehr geändert wird, also z.B. Back-Up's oder sowieso ganz volle Floppies, ist von Byte >22 bis zum Ende aller Platz für File-Informationen nutzbar. Ein 3-Byte Block mit Nullen muß aber mindestens als Trennung vorhanden sein, um das Ende der Block-Link Map zu indizieren!

Aufbau der Datensektoren

Inhalt und Aufbau der Datensektoren sind für die DSR-Software von geringer Bedeutung. Besondere Informationen enthalten nur Datei-Sektoren, da bei ihnen der Anfang und das Ende eines Datensatzes (Eintrags bzw. Records) bekannt sein müssen. Auch ist die optimale Ausnutzung des Speicherraums von der Datenstruktur abhängig.

Bei PROGRAM-Files fällt jegliche Strukturierung weg. Hier wird jeder Sektor soweit möglich vollkommen mit Daten gefüllt, lediglich im letzten Sektor können an dessen Ende einige Bytes übrig bleiben. Es wird aber auch dann ein neuer Sektor belegt, wenn nur ein Byte nicht mehr in den vorangegangenen Sektor passte. Bei der Strukturierung von P-Files in eigenen Assemblerprogrammen sollte daher die File-Länge immer ein Vielfaches von 256 (>100) betragen.

Bei satzorientierten Dateien variabler Länge steht vor jedem Datensatz ein Byte, welches die Länge des folgenden Eintrages bezeichnet. Bei Dateien vom Typ FIXED, also solchen mit fester Satzlänge, fehlt dieses Byte natürlich, da alle Einträge gleich lang sind und damit deren Position im Sektor einfach berechnet werden kann. Wichtig ist auch die EOS-Kennung (End-Of-Sector), die ebenfalls bei FIXED-Dateien nicht existiert. Sie steht ggfs. am Ende eines belegten Sektors.

Da ein einzelner Datensatz nicht aufgebrochen werden darf, wird er, wenn im vorhandenen Sektor auch nur um 1 Byte zu wenig Platz ist, komplett in den nächsten Sektor geschrieben. Die optimale Raumausnutzung liegt dann vor, wenn die Länge der Datensätze (ggfs. inclusive Längenbyte) ein ganzzahliger Bruchteil von 256 ist. So wird z.B. bei einer FIXED 64 Datei jeder Sektor optimal genutzt, bei VAR-Dateien ist ein Optimum wegen der variablen Satzlänge sehr schwer zu erreichen.

Aus dieser Verwaltung der Daten resultiert die Einschränkung, daß VAR-Dateien Satzlängen von maximal 254 Byte haben dürfen, FIXED dagegen 255 - es muß eben immer noch Platz für das Längen- und/oder EOS-Byte bleiben.

Es besteht nun die Möglichkeit, bei einer VAR-Datei ein beliebiges Längenbyte durch ein >FF zu ersetzen. Der 'Erfolg' liegt auf der Hand: alle nachfolgenden Datensätze werden als nicht vorhanden angesehen (>FF setzt EOF-Flag), werden aber beim Kopieren mittels eines Diskmanagers, der sektororientiert vorgeht, mit kopiert. Damit lassen sich besonders TI-Writer Files schützen um z.B. persönliche Einträge unsichtbar zu machen. Man sollte sich aber zwecks Wiederherstellung der Daten die alten Längenbytes merken!

Eleganter ist aber die Methode, einen ganzen Sektor verschwinden zu lassen, in dem er in der File-Link Map gewissermaßen 'weggeclustert' wird.

Hierzu ein Beispiel:

Es soll der vorletzte Sektor aus unserem vorhergehenden Beispiel versteckt werden. Dazu muß der erste Block des Files verkürzt und ein zweiter erzeugt werden, der einen Sektor später beginnt. Aus den in die richtige Reihenfolge gebrachten Clusterdaten

>30B, >03C, >000, >000 muß werden

>30B, >03A, >347, >03B, Sektor >346 fehlt nun.

Nachdem diese Bytes in der richtigen Weise verdreht und geschrieben wurden, muß nun noch die Gesamtanzahl korrigiert werden, und der Sektor ist verschwunden. Wird das File gelöscht, so bleibt der so isolierte Sektor bestehen, da er von keiner Link-Map überdeckt wird! Er wird allerdings auch nicht mehr kopiert.

Außer z.B. mit dem Diskmanager-1000 (Option Sweep Disk) kann dieser Sektor nicht mehr gelöscht werden, es sei denn durch Formatieren.

Wie zu sehen ist, ist es nicht möglich die Struktur eines Datensektors anhand eines Beispiels zu verdeutlichen. Daher wird auf den gewohnten Sektor-Dump hier verzichtet. Der Leser ist aufgefordert, selbst seine Analysen vorzunehmen.

Zugriff auf einzelne Datensätze

Auf der höchsten Programmierebene des TI werden einzelne Datensätze ohne Beachtung ihres Inhalts, unabhängig von ihrer physikalischen Position auf der Diskette und ungeachtet einer eventuellen Fragmentierung der Datei gelesen. Beim Lesen von VAR-Dateien wird ein prinzipiell nicht vorhersagbares Längenbyte übergeben, die Optionen bei FIXED-Dateien erlauben neben dem einfachen APPEND auch ein Update bereits gespeicherter Datensätze.

Welche 'Kopfstände' die DSR der obersten Ebene (Level 2) machen muß, um an die Daten zu kommen, sei im Folgenden kurz umrissen. P-Files sind, da nicht satzorientiert, nicht angeführt.

Zugriff auf bestimmte Datensätze bei FIXED-Dateien

Bei FIXED-Dateien wird in der Regel ein bestimmter Datensatz über eine Satznummer angesprochen, ob nun zum Lesen oder Schreiben. Wichtig ist der Hinweis, daß auch beim sequentiellen Lesen einer FIXED-Datei diese Satznummer benötigt wird, die jedoch ggfs. vom Interpreter/Compiler selbständig verwaltet wird.

Da jeder Datensatz die gleiche Länge aufweist, kann die DSR anhand der Sektorlänge, der Satzlänge und der Satznummer den Offset im File ermitteln, d.h. den Abstand in Sektoren und Bytes (im Zielsektor) vom Beginn der Datei. Diese Zahl wird gespeichert und es wird dann anhand der Block-Link-Map so lange gesucht, bis der Startsektor und der Offset in diesem Sektorblock bekannt sind. Sodann wird über die Satzlänge direkt der Sektor ausgelesen (die oben berechneten Bytes geben den Offset im Sektor an). Ein Überschreiten des EOF ist nicht möglich, da dieser im FDS vermerkt ist und leicht abgefragt werden kann.

Das Schreiben läuft ähnlich ab, jedoch gibt es hier einen Sonderfall, der besondere Beachtung verdient: Wird ein Datensatz angesprochen, dessen Satznummer über der des derzeitig letzten Eintrages in der Datei liegt, so werden alle Sätze bis zum gewünschten Satz hin erzeugt, ggfs. also auch solche mit ungültigem Eintrag dazwischen, denn das Füllen der Datensätze ist Aufgabe des Anwenderprogramms, nicht der DSR. Wertet die DSR dabei ausschließlich die Block-Link-Map aus, so wirken sich Zusatzinformationen im FDS möglicherweise fatal aus (s.o.)

Zugriff auf bestimmte Datensätze bei VAR-Dateien

Der Zugriff auf Datensätze einer Datei mit variabler Satzlänge gestaltet sich etwas schwieriger. Hier ist es nicht möglich, allein aus Satzlänge, Sektorlänge und Satznummer die Position eines Eintrages zu ermitteln, da ja jeder Eintrag Längen von 1 Byte bis zur spezifizierten maximalen Satzlänge aufweisen kann. Einzig das Anfügen neuer Datensätze ans Ende der Datei ist möglich, da ja der EOF-Offset im FDS vermerkt wird. Aus diesem und der ebenfalls im FDS vermerkten Sektoranzahl kann die Position eines neuen Eintrags einfach errechnet werden.

Der Zugriff auf eine bestimmte Eintragnummer ist nur durch sequentielles Lesen aller davor befindlicher Datensätze möglich. Zwar ermittelt die DSR intern, wieviele Sätze bei der aktuellen Länge in einen Sektor passen, doch ist das ziemlich witzlos, da es bei VAR-Dateien allenfalls statistischen Wert hat.

Datenverwaltung im VDP-RAM

Es dürfte mittlerweile bekannt sein, daß das RAM des Video-Prozessors TMS 9929A/9918/9935 ... in der Konsole der einzige größere und zusammenhängende Speicherbereich des TI in der Grundversion ist.

Da es bei der Konzeption des TI-99/4A gefordert war, alle Erweiterungen wie Schnittstelle und Diskettenstation unabhängig vom Ausbauzustand der Anlage benutzen zu können, kommt eben diesem Speicherraum, der nur durch eine Pipeline von 2 Adressen zu erreichen ist, große Bedeutung zu. Erfahrene Assemblerprogrammierer werden jetzt vielleicht einwenden, daß es doch 4 Adressen gibt, über die man den Video Prozessor ansprechen kann. Das ist zwar nicht falsch, durch die Benutzung von 4 Adressen werden aber lediglich bestimmte Registeranwahlleitungen des VDP gesetzt, so daß die Register logisch getrennt angesprochen werden können.

In diesem Kapitel soll gezeigt werden, welche für die Diskettenverwaltung relevanten Informationen sich im VDP-RAM befinden, was sie bedeuten und was mit ihnen angestellt wird.

Die Belegung des VDP-RAMs wird mittels einiger Pointer im Scratch Pad RAM der Konsole verwaltet, das bekannterweise der einzige CPU-adressierbare 16 Bit Bereich des TI 99/4A in der Grundversion ist. Strenggenommen ist dabei nur der Pointer >8370 festgelegt - alle anderen werden speziell von der Diskcontroller-DSR beim Aufruf oder beim File-Handling verwendet.

Diese Pointer sind:

- >8356 Arbeitszeiger auf die momentan zu bearbeitenden Daten. Dieser Zeiger ist als POINT in Sachen DSRLNK bekannt, wird aber so stark frequentiert und geändert, daß diese eine Bedeutung fast nebensächlich ist.
- >8358 Zeiger auf den Volume ID Block. Dieser Pointer zeigt auf den Beginn des Speicherbereiches, in dem sich eine Kopie von Sektor 0 der zuletzt angesprochenen Diskette befindet.
- >8366 VDP-Stackpointer. Aus den eingangs gezeigten Gründen ist es für die DSR Software, welche es auch sei, problematisch, mit den effektiv nur 12 verfügbaren Registern auszukommen. Daher werden Rückkehradressen oder globale Pointer zwischenzeitlich im VDP-Stack Bereich abgelegt. Dieser Pointer wird mit jedem neuen Eintrag nach unten, also zu niedrigeren Adressen hin, verschoben, und zeigt immer auf eine freie Speicherstelle.
- >8370 Zeiger auf die höchste verfügbare VDP-Adresse bzw. den VDP-Area-Link. Nach einem CALL FILES(3), das ist der Zustand nach Anschalten des Rechners und Ausführung der Power-Up-Routine des Diskcontrollers, enthält dieser Pointer üblicherweise den Wert >37D7, also ein Byte unterhalb der Disk-Puffer Zone. Ist ein Modul gesteckt, welches über eine eigene Power-Up-Routine verfügt, so zeigt >8370 auf das Byte im VDP-RAM unterhalb dieses Bereiches. Befindet sich eine DSR mit Power-Up-Sequenz auf einer CRU-Seite vor dem Diskcontroller, ist der gesamte Disk-Pufferbereich um die hier reservierte Byteanzahl nach unten verschoben.

Wie die folgende Speicherbelegung zu zeigen scheint, tummeln sich im oberen VDP-RAM Bereich ausschließlich Disketteninformationen. Das ist aber nur verständlich, wenn man sich die vielfältigen Funktionen vor Augen führt, die ein Diskettenmanagement haben muß. Näheres hierzu enthalten die ROM-Listings zu den Diskcontrollern. Die Adreßangaben gelten nur, sofern kein Gerät auf CRU >1000 VDP-RAM für sich allokiert!

Funktion des VDP-Area-Links

Wie eingangs dieses Kapitels angedeutet, steht das VDP-RAM prinzipiell allen anderen DSRs ebenfalls zur Verfügung. Um diese Mehrfachnutzung ohne Kollisionen oder Überschneidungen der einzelnen Bereiche zu gewährleisten, hat sich TI etwas Trickreiches einfallen lassen:

Beim Power-Up (dem Einschalten des Systems), wird vom Konsolen-Betriebssystem die (festgelegte) höchste freie Adresse im VDP-RAM an die CPU-Adresse >8370 geschrieben. Diese Adresse ist bekannterweise >3FFF, was aus dem Design des VDP-Chips bzw. dessen Adressierungslogik herrührt. Jede DSR bzw. jedes Modul, welche(s) über eine Power-Up-Routine verfügt, kann durch Vermindern des Inhaltes von >8370 und Pflegen einer einfach verketteten Liste (s.u.) VDP-Speicher 'abzweigen'. Die jeweilige Routine muß dazu lediglich den aktuellen Inhalt von >8370 als VDP-RAM-Obergrenze akzeptieren, diesen Wert in die Kette im VDP-RAM eintragen, den Inhalt von >8370 um den Betrag reduzieren, der an Speicher benötigt wird und einen Vermerk in die Liste aufnehmen, anhand dessen die DSR später den selbst reservierten Bereich erkennt. Bei DSRs auf P-Box-Karten ist dies sinnvollerweise die CRU-Seite, bei Modulen kann dies der GROM-Zähler o.ä. sein (TE-II vermerkt z.B. eine >40).

Sobald nun eine DSR auf das VDP-RAM zugreifen will, sucht sie den ersten Eintrag der Liste (auf den >8370 zeigt), prüft das ID-Byte und sucht ggfs. an der Adresse weiter, welche in der Liste vorhanden ist (VDP-Area-Link).

Memory-Map des oberen Teils des VDP-RAMs

VDP-Area-Link, 4 Bytes:

- >37D8 Ident-Code eines allokierten Blocks >AA
- >37D9 Zeiger auf Vorgängerliste (>8370 alt) >3FFF
- >37DB CRU-Seite als Bereichskennung >11
- >37DC Maximalzahl gleichzeitig offener Files >03

Filedaten des zuerst geöffneten Files - 518 Bytes

Zeigerblock zum File - 6 Bytes

- >37DD Zeiger auf den aktuellen File Record (Offset)
- >37DF Sektornummer des File-Directory Sektors
- >37E1 Logischer File Record Offset bei VAR-Dateien
- >37E2 Laufwerknummer, in dem sich die Diskette mit dem File befindet

Beginn des Puffers für den File-Directory Sektor - 256 Bytes

- >37E3 Filename - 10 Bytes
- >37ED Reserviert >0000
- >37EF File Status
- >37F0 Maximalzahl der Einträge (Records) pro Sektor
- >37F1 Gesamtzahl bisher belegter Sektoren, incl. File-Dir Sektor!
- >37F3 EOF-Offset (Eintrag Nr.) im letzten belegten Sektor
- >37F4 Satzlänge der Einträge (Record Length)
- >37F5 Record Anzahl bei FIXED-Dateien, Anzahl der Sektoren bei bei VARIABLE-Dateien. Die Bytes müssen vertauscht werden (SWAP)!
- >37F7 Reserviert >0000,>0000,>0000,>0000
- >37FF File Link Daten. Geben an, ob das File aufgebrochen wurde und wenn ja, wo die Teile beginnen und wie lang sie sind. Näheres siehe Beschreibung der File-Directory Sektoren.
- >38E3 Beginn des Datenpuffers - 256 Bytes

Filedaten des an zweiter Stelle geöffneten Files - 518 Bytes

- >39E3 Beginn des Zeigerblocks – 6 Bytes
- >39E9 Beginn des File-Dir Puffers - 256 Bytes
- >3AE9 Beginn des Datenpuffers - 256 Bytes

Filedaten des an dritter Stelle geöffneten Files - 518 Bytes

- >3BE9 Beginn des Zeigerblocks - 6 Bytes
- >3BEF Beginn des File-Dir Puffers - 256 Bytes
- >3CEF Beginn des Datenpuffers - 256 Bytes

VDP-Stack Bereich -252 Bytes

- >3DEF Unteres Ende

- >3EEA Oberes Ende

Laufwerk Informationsblock - 4 Bytes

- >3EEB Nummer des zuletzt angesprochenen Laufwerks
- >3EEC Aktuelle Spur, über der der Kopf von LW 1 steht
- >3EED Aktuelle Spur, über der der Kopf von LW 2 steht
- >3EEE Aktuelle Spur, über der der Kopf von LW 3 steht

Unbenutzter Bereich - 5 Bytes

- >3EEF bis >3EF3

Sektor 0 Puffer - 257 Bytes

- >3EF4 Laufwerknummer, von der Sektor 0 stammt -1 Byte
- >3EF5 Sektor 0 Puffer der Diskette, auf die zuletzt GESCHRIEBEN wurde - 256 Bytes

Puffer für den Vergleich von Filenamen - 11 Bytes

- >3FF5 Laufwerknummer - 1 Byte
- >3FF6 Filename - 10 Bytes

Detaillierte Beschreibung der Daten

- Bereich >37D8 - >37DC

Dieses Datenfeld kennzeichnet für die Diskcontroller-Software den Beginn des mit CALL FILES( ) freigegebenen Bereiches. Diese 5 Bytes **müssen** am Anfang des Disk-Pufferbereiches stehen, sonst droht bei bestimmten Controllern ein Absturz. Es reicht also nicht, ein CALL FILES durch direktes Beeinflussen des Pointers an >8370 zu simulieren, auch der Header des Pufferbereiches muß stimmen! Aus der Speicherbelegung ist direkt zu sehen, was bei einem CALL FILES passiert: Der Header wird um 518 Bytes (oder ein mehrfaches davon) nach unten oder oben versetzt und ein Disk Puffer Bereich entfernt oder hinzugefügt.

Bereich >37DD - >37E2

Der Record Zeiger (es gibt eigentlich 2) zeigt, wie aus dem Basic wohl bekannt, auf den aktuellen Datensatz. Besondere Bedeutung kommt aber der Sektornummer des File-Dir Sektors zu. Wird ein File mit der 'DELETE' Option geschlossen, so wird mithilfe dieser Zahl im Sektor 1 der betroffenen Diskette der Sektor gesucht und aus der Link-Map entfernt. Ist bei der Laufwerknummer das höchstwertige Bit gesetzt, so wurde der Datensektor beim letzten Zugriff aktualisiert. Nach einem RESTORE ist der File-Record Zeiger >FFFF und der File-Record Offset >00, ebenso direkt nach der Eröffnung im Output- oder Append-Mode.

Bereich >37E3 - >39E2

Puffer für den File-Directory Sektor und den aktuellen Datensektor. Diese Daten wurden bereits im Kapitel 5 abgehandelt. Es gibt jedoch Besonderheiten, welche die Kennung freier bzw. belegter File-Control Blöcke betreffen. Ist das erste Byte des Filenamenbereiches gleich >00, so ist dieser Puffer nicht belegt und kann neu verwendet werden. Ist das erste Byte ein Blank (Leerzeichen >20), dann ist dieser Puffer für die Daten beim Aufbau eines Directory reserviert. Ist das höchstwertige Bit im 1. Byte des Filenamens gesetzt, so ergaben sich Änderungen im FDS. Dieser muß dann beim Abschluß einer File-Operation (CLOSE o.ä.) neu geschrieben werden. Da dies im Allgemeinen vom DOS erledigt wird, ist ein gesetztes High-Bit nur bei Fehlfunktionen des DOS zu erkennen.

Bereich >39E3 - >3DEE

Daten der weiteren Files, Bedeutung wie 1. File.

Bereich >3DEF - >3EEA

Hier werden zwischendurch Rückkehradressen und globale Registerinhalte gesichert. Bei Verwendung dieses Stacks in eigenen Programmen ist darauf zu achten, daß er so verlassen wird, wie er angetroffen wurde, der Pointer auf >8366 stimmt und insbesondere der zulässige Bereich nicht überschritten wird, wodurch die File Daten zerstört würden. Fassungsvermögen 126 Worte.

Bereich >3EEB - >3EEE

Dieser Bereich wird nur von solchen Diskcontrollern benutzt, die kein eigenes System-RAM besitzen, so z.B. der TI-Controller. Andere Controller aktualisieren lediglich die Laufwerknummer. Für die Steuerung der Kopfposition der Diskettenlaufwerke ist es nötig zu wissen, wo sich der Schreib/Lesekopf im Moment befindet. Diese Bytes dienen als Puffer dafür.

Bereich >3EEF - >3EF3

Eine Verwendung dieser 5 Bytes ist momentan nicht bekannt, der Bereich ist aber wohl zu klein, um für eigene Anwendungen interessant zu sein.

Bereich >3EF4 - >3FF4

Hier befindet sich immer eine Kopie des Sektor 0 derjenigen Diskette, auf die zuletzt geschrieben wurde. Alle Änderungen der Sektor Bitmap werden in diesem Bereich vorgenommen, anschließend wird der Sektor zurückgeschrieben. Die Laufwerknummer gibt wiederum im höchstwertigen Bit ein evtl. durchgeführtes Update an.

Bereich >3FF5 - >3FFF

File Name Compare Buffer (FNCB). Hier werden bei Bedarf die vom Programm geforderten Filenamen mit den auf der Diskette angetroffenen verglichen. Wird ein Filename also irgendwann spezifiziert, dann wird er hier hineinkopiert, die Laufwerknummer vorangestellt und auf der Diskette nach diesem Namen gesucht. Die Funktionsabläufe im Einzelnen sind sehr komplex, hier muß wieder auf detaillierte ROM-Listings verwiesen werden.

Noch einige Hinweise und Tips:

Soll ein File gelöscht werden, so sucht die Diskcontroller DSR nach dem Namen zuerst bei den offenen Files. Soll keines von denen gelöscht werden, so wird nach einem nicht geöffneten File gesucht, um in diesem Bereich die Daten zu bearbeiten. Existiert kein freier Platz im VDP-RAM, so kann kein File gelöscht werden! In diesem Fall muß ein File geschlossen werden.

Beim Löschen eines Files wird dann KEINE Fehlermeldung gegeben, wenn das File nicht existierte. Das ist eine Nachlässigkeit der Softwareentwickler von TI gewesen, die auch bei allen Fremdherstellern zu finden ist.

Beim Eröffnen einer Datei wird deren FDS gelesen, sofern er existierte, oder ein leerer, neuer FDS auf der Diskette erzeugt. In jedem Fall aber befindet sich die einmal ermittelte Laufwerknummer vor dem Beginn der Filenamenzone. Diese Nummer bleibt bestehen, solange das File offen ist und wird nicht mehr verifiziert. Gleiches gilt für den Volume-ID Block, dessen Laufwerknummer bis zum Schreiben des ID-Blockes oder dem Lesen eines neuen gültig bleibt. Wird nun nach dem Öffnen einer Datei im Output- oder Append-Mode die Diskette gewechselt, ein Datensatz geschrieben und die Datei geschlossen, so bekommt die eingelegte Diskette den Sektor 0 der ursprünglich eingelegten Diskette 'verpaßt'. Die Folgen sind weitreichend.

Bestand das File auf beiden Disketten in allen Sektoren identisch, dann ist das File auf der eingelegten Diskette noch in Ordnung. War jedoch auf der zweiten Diskette ein Kopierschutz über 'P' in Sektor 0 und keiner auf der ersten Diskette, so ist die zweite Diskette nun nicht mehr geschützt.

Es klingt unglaublich, aber so kann ein Kopierschutz der Diskette auch in Basic entfernt werden!

Voraussetzung ist aber, daß auf der geschützten Diskette noch mindestens 2 Sektoren frei sind, in die das neu eröffnete File geschrieben werden kann. Der Filename darf nicht bereits bestehen, da sonst vorhandene FDS verfälscht werden.

Hier eine kurze Anleitung für DSK1:

Auf beiden Disketten wird ein File, z.B. DSK1.PLACEBO im DIS/VAR 80 Format oder irgendeinem anderen eröffnet. Dieser Filename darf auf der zweiten, zu manipulierenden Diskette noch nicht existieren. Das File wird gleich nach der Eröffnung wieder geschlossen. Nun wird die erste, ungeschützte Diskette eingelegt, das File im Append-Mode wieder eröffnet und die geschützte Diskette eingelegt. Nun wird ein Datensatz geschrieben und das File geschlossen. Nun ist die eingelegte Diskette nicht mehr geschützt.

Es gibt jedoch mögliche Fehlerquellen. Neben der Laufwerknummer ist auch die Sektornummer des FDS eine Konstante, der FDS wird also an die Stelle auf der zweiten Diskette geschrieben, an der er sich auf der ersten befand. Es ist aber nicht gesagt, daß dies in beiden Fällen der gleiche Sektor ist. In diesem Fall wäre ein File der zweiten Diskette zerstört.

Hier hilft der umgekehrte Weg. Das Pseudo-File wird auf der geschützten Diskette eröffnet und auf der ungeschützten geschlossen. Dadurch stimmen die Sektornummern überein. Jetzt kann das o.a. Verfahren ohne Gefahr angewandt werden.

Maschinenprogramme für Extended-Basic (DIS/FIX 80) können von geschützten Disketten mittels des Editors zum E/A-Modul kopiert werden, oder ganz einfach mit einem Basic-Programm. Das File wird mit dem Editor geladen und danach als DIS/FIX 80 File wieder abgespeichert. Da Maschinenprogramme für X-B keine Symbole außerhalb des normalen ASCII-Zeichensatzes enthalten dürfen, werden diese Control-Characters auch nicht gelöscht. Ein Basic-Programm zum Kopieren eröffnet beide Dateien normal, liest einen Eintrag aus der geschützten Datei und schreibt ihn auf eine ungeschützte - ganz einfach!

Abriss der Funktionen eines Disketten-Managers

In diesem Kapitel soll weniger eine Anleitung zur Programmierung eines Disk-Managers gegeben werden, das würde den Rahmen in jedem Falle sprengen. Vielmehr sollen die grundlegenden Probleme und logischen Abläufe verdeutlicht werden, die ein solches Programm beinhaltet.

Initialisieren von Disketten

Bereits in den Einleitungskapiteln wurde eine Anleitung zu einem Programm gegeben, welches Disketten formatiert. Nach dem Abschluß dieser Routine sind aber die Sektoren 0 und 1 noch nicht definiert, da vorerst alle Sektoren mit dem Wert >E5 gefüllt sind. Nach diesem mehr physikalischen Vorgang des Formatierens muß die Diskette auch softwareseitig nutzbar gemacht, also initialisiert werden.

Nach dem Formatieren muß also geprüft werden, ob auch wirklich in allen Sektoren der Anfangswert >E5 steht. Ist dies bei einem Sektor nicht der Fall oder treten beim Lesen eines bestimmten Sektors Fehler anderer Art auf, dann muß dieser Sektor, um Datenverlust beim späteren Gebrauch zu vermeiden, in der Sektor-Bitmap als belegt gemeldet werden. Es muß also nach dem Formatieren ein 'Bad Block Scan' durchgeführt werden, während dem im RAM ein Bild des späteren Sektor 0 aufgebaut wird. Die Bits der Sektor-Bitmap, die bei der gewählten Formatierung nicht belegbar sind (die Bitmap ist ja immer etwas zu lang), müssen in jedem Fall als belegt gemeldet werden, sonst wird beim Schreiben versucht, über den Diskettenrand hinaus zu arbeiten. Natürlich müssen auch die Sektoren 0 und 1 belegt gemeldet werden.

Neben dem Isolieren defekter Sektoren (Sektor 0 und 1 müssen o.K. sein!) müssen die Diskettenparameter geschrieben werden (Siehe Kapitel 7). Besonders wichtig ist hier die Textsequenz 'DSK '(Leerzeichen beachten!), die als Kennung einer formatierten Diskette benötigt wird. Ist der Sektor 0 so normiert, dann kann Sektor 1 mit Nullen gefüllt werden.

Erst jetzt wird die Diskette vom DOS akzeptiert.

Will man nun eine Diskette komplett löschen, dann war das bisher nur durch das erneute Initialisieren möglich. Es reicht aber auch, die Sektoren 0 und 1 auf ihren 'Urzustand' zurückzusetzen. Dies birgt jedoch die Gefahr, daß evtl. defekte Sektoren nun wieder zugänglich sind. Ein Sektor-Scan mit Lesen und Schreiben würde den Zeitvorteil zunichte machen.

Maßgeblich für die Ladezeit eines Files ist das sogenannte 'Sektor Interlace'. Je nachdem wie man es wählt, lassen sich die Ladezeiten bei allen Filearten, deren Sektoren nacheinander (am Stück) gelesen werden bis etwa um den Faktor 2 vergrößern oder verringern. Diese Wahl hat man auf normalem Wege nur bei dem Diskmanager zum CorComp 9900 Controller. Beim einfachen Formatieren über die Subroutine >11 wird ein festgelegtes Interlace verwendet. Schaut man sich nach dem Formatieren den Pufferbereich im VDP-RAM an, sieht man direkt, welche Daten die zuletzt geschriebene Spur enthält. Bei Kenntnis des Spuraufbaus kann daraus auf das Interlace geschlossen werden. Näheres findet sich in dem entsprechenden ROM-Listing.

Das Standard-Interlace ist im Allgemeinen ein recht guter Kompromiß für alle Filearten und Loader, kann aber im Einzelfall verbesserungsfähig sein. Es ist jedoch nur bei genauer Kenntnis der Diskcontroller-Hardware möglich, hier einzugreifen.

File-Operationen

File-Operationen beziehen sich auf die logische Struktur 'File', wie bereits in den einführenden Kapiteln angedeutet. Aufgrund der komplexen Struktur, welche insbesondere lange und fragmentierte Files aufweisen können, wird die Behandlung sinnvollerweise nicht auf Sektorebene durchgeführt (obwohl dies natürlich genau das ist, was zum täglichen Brot einer DSR gehört).

Filenamen ändern

Zuerst wird der neue Filename in den entsprechenden File-Directory Sektor (FDS) geschrieben. Danach muß in Sektor 1 die Nummer des FDS so verschoben werden, daß alle Filenamen wieder in alphabetisch aufsteigender Form vorliegen. Beim Filenamen sind prinzipiell alle ASCII-Symbole (7 Bit) erlaubt, es muß aber Verträglichkeit mit der verwendeten Software (Basic o.ä.) bestehen. Weitere Änderungen sind nicht erforderlich.

Einfacher ist hier die Anwendung der Level-1-Routine >13. Die Zeiger auf den alten und den neuen Namen werden übergeben und die Routine aufgerufen. Die Umbenennung des FDS und die Sortierung von Sektor 1 werden von der Routine übernommen. Näheres enthält ein Beispielprogramm.

Files löschen

Ein File wird gelöscht, indem die Nummer seines FDS aus Sektor 1 entfernt wird und alle nachfolgenden Einträge um 2 Byte nach unten verschoben werden. Auch muß anhand der Cluster-Daten im jeweiligen FDS die Sektor-Bitmap in Sektor 0 von einigen gesetzten Bits befreit werden. Unbedingt notwendig ist, daß das zu löschende File vorher geschlossen wurde, da sonst die Cluster-Daten nicht unbedingt stimmen. Das ist alles, was beim Löschen eines Files geschieht. Alle Datensektoren und der FDS existieren immer noch unverändert. Dadurch ist es möglich, ein gelöschtes File komplett wieder herzustellen, wenn nur sein Name bekannt ist und inzwischen nicht auf die Diskette geschrieben wurde!

Zum Löschen eines Files steht die Level-2-Option 'DELETE' zur Verfügung.

Files kopieren

Hier gibt es prinzipiell zwei mögliche Verfahren. Das einfachste (programmiertechnisch gesehen) ist die Eröffnung des Files im INPUT- Modus, die Eröffnung eines Files im OUTPUT-Modus auf der Zieldiskette und der satzweise Transfer der Daten. Gegen dieses Verfahren ist bis auf die niedrige Geschwindigkeit nichts einzuwenden. Es scheitert aber spätestens bei überlangen PROGRAM-Files. Diese können nur sektororientiert kopiert werden. Wenn das aber bei PROGRAM-Files nötig wird, dann kann auch der ganze Rest der Kopiererei sektororientiert geschehen, da dann kein Unterschied mehr zwischen den Files existiert. Zuerst wird der FDS kopiert, und zwar in einen Puffer im RAM. Nun werden die zum File gehörigen Sektoren anhand der Cluster-Tafel bis zum Ende (des Files oder des Datenpuffers) gelesen und auf der Zieldiskette geschrieben (davor einen Sektor für den FDS reservieren!). Während des Transfers wird ein geänderter FDS erzeugt, der eine eventuell notwendige Aufspaltung des Files auf der Zieldiskette protokolliert. Nach dem Transfer wird dieser FDS geschrieben und die alphabetisch korrekte Eintragung in Sektor 1 vorgenommen. Die Sektor-Bitmap wird aktualisiert und der Kopiervorgang ist beendet.

Auch hier hilft eine Level-1-Routine, das komplette Erstellen eines solchen Programms zu umgehen. Eigentlich sind es zwei Routinen, die vefügbar sind. Die erste liest entweder Teile des FDS oder eine bestimmte Anzahl von Datensektoren. Die zweite schreibt entweder den FDS und reserviert gleichzeitig die notwendigen Sektoren, oder schreibt eine bestimmte Anzahl von Datensektoren in das Copy-File. Die Namen der Routinen sind >14 und >15. Auch hierzu sind Beispielprogramme vorhanden.

Gelöschte Files wiederherstellen

Zu einem ordnungsgemäß wiederhergestellten File gehören: richtiger Eintrag der FDS-Nummer in Sektor 1 und angepasste Sektor-Bitmap mit gesetzten Bits für jeden von dem File nun wieder okkupierten Sektor (FDS nicht vergessen!). Zuerst muß der FDS gesucht werden, wozu der ehemalige Filename benötigt wird. Dieser wird auf der Diskette gesucht und es wird geprüft, ob es sich bei dem evtl. gefundenen Filenamen auch um einen FDS handelt (am Besten, Null-Bytes prüfen - Murphy ist überall!!). Beim anschließenden 'Reparieren' der Sektor-Bitmap muß geprüft werden, ob als belegt zu meldende Sektoren noch frei sind, andernfalls wurde das File bereits überschrieben. Wurde der FDS überschrieben, so ist nichts mehr zu retten (im Allgemeinen!), fehlende Datensektoren sind manchmal zu verkraften.

Fileschutz ändern

Hier steht wiederum eine Level-1-Routine zur Verfügung, näheres im Kapitel mit den Beispielprogrammen. Es gibt aber auch den 'direkten' Weg. Zuerst den FDS suchen, dann in den Puffer holen und das File-Status Byte nach Bedarf beeinflussen. Anschließend wird der FDS an seine alte Stelle zurückgeschrieben.

Diskettenkatalog

Der Aufbau eines Diskettenkataloges ist auf 2 Arten möglich. Zum einen kann die Standard Directory Datei 'DSKX.' im INPUT-Mode eröffnet werden (das X ist selbstverständlich durch die entsprechende Laufwerkskennung zu ersetzen), deren Einträge satzweise gelesen und angezeigt werden. Diese Methode erinnert aber stark an Basic, und ist daher wohl dem versierten Assembler-Programmierer suspekt. Eine größere Einschränkung bedeutet aber wohl die eigentlich vollkommen unverständliche Eigenart dieser Routine des DOS, daß bei jedem Eintrag, der gelesen werden soll, vor dem eigentlichen FDS immer erst Sektor 1 gelesen wird. Das macht diese Art des Inhaltsverzeichnisses sehr langsam. Aber es geht selbstverständlich auch anders.

Diskettendaten aus Sektor 0

Sektor 0 wird gelesen, der Name angezeigt. Sodann wird die Gesamtzahl der Sektoren ermittelt und anhand der Sektor-Bitmap so lange dekrementiert (jedes nicht gesetzte Bit reduziert die Anzahl der belegten Sektoren) bis das Ende des Sektor 0 erreicht ist. Nun existieren zwei Zähler, einer mit der Anzahl der belegten

Sektoren und derjenige mit der Gesamtzahl aller Sektoren. Diese werden in Dezimalzahlen umgewandelt und angezeigt.

Der Vorteil dieses Verfahrens liegt darin, daß das Ende der Bitmap nicht bekannt sein muß, da nur nicht belegte Sektoren zählen, das Bitmap-Ende aber entsprechend der Vorschrift mit >FF gefüllt ist.

Filedaten anzeigen

Hier wird eine Kopie von Sektor 1 im RAM angelegt, mit dessen Hilfe nacheinander alle eingetragenen Sektoren (FDS) gelesen werden. Deren Filenamen werden angezeigt, die Anzahl der belegten Sektoren um eins inkrementiert und angezeigt und der Filestatus umgewandelt und ebenfalls angezeigt. Bei einer FDS-Nummer von Null endet die Ausgabe.

Disketten kopieren

Für das Kopieren von Disketten bieten sich 3 Möglichkeiten an:

1. Das komplette Kopieren aller Files,
2. Das Kopieren aller belegten Sektoren,
3. Das Kopieren aller Sektoren der Diskette.

Das erste Verfahren ist dann sinnvoll, wenn alle Files auf eine neue Diskette gebracht werden sollen, wo dann jedes File aus nur einem zusammenhängenden Block besteht. Leider besticht dieses Verfahren nicht durch eine hohe Geschwindigkeit, es werden aber keine bereits belegten Sektoren überschrieben, wie dies bei den beiden anderen Verfahren der Fall sein kann.

Das zweite Verfahren kann bei nur teilweise belegten Disketten enorm zeitsparend sein, vorausgesetzt die Sektor-Bitmap wurde nicht manipuliert. Bei diesem Verfahren wird nämlich die Sektor-Bitmap in einen Puffer im RAM verlegt und anhand der Einträge geprüft, welche Sektoren kopiert werden müssen.

Es ist übrigens nicht, wie einige nun vielleicht denken werden, notwendig, die Nummern der gelesenen Sektoren zu protokollieren um diese später wieder an den richtigen Platz zu bringen. Diese Information steckt ja in der Sektor-Bitmap. Es reicht also, die Sektoren anhand der Bitmap zu lesen, bis der Puffer voll oder die Bitmap zu Ende ist. Wird beim Schreiben gleichermaßen verfahren, dann ist die richtige Reihenfolge automatisch sichergestellt. Da aber, wie bereits angesprochen, der Zustand der Sektor-Bitmap beim Lesen von Files irrelevant ist, läßt sich ein solches, die Bitmap benutzendes Programm aber risikolos austricksen.

Dieses Austricksen ist bei einem Kopierprogramm des dritten Typs nicht möglich. Hier wird die Diskette Sektor für Sektor kopiert, ohne auf deren Belegung zu achten. Dazu muß aber die Anzahl der Sektoren auf der Diskette bekannt sein. Diese läßt sich aus Sektor 0 lesen oder aber vom Bediener abfragen, was sicherer ist. Gültige Angaben sind hier:

- 360 Single Sided, Single Density
- 720 Double Sided, Single Density
- 640 Single Sided, Double Density (Myarc)
- 720 Single Sided, Double Density
- 1280 Double Sided, Double Density (Myarc)
- 1440 Double Sided, Double Density

Alle Zahlen dezimal!

Diskettentests

Diskettentests dienen im Allgemeinen dazu, defekte Sektoren aufzuspüren, sei es beim erstmaligen Einsatz der Diskette, also nach dem Formatieren oder beim Auftreten von Lesefehlern im Betrieb. Bei Fehlern die während des Einsatzes beschriebener Disketten auftreten, kommen nur Tests in Frage, welche die Daten unverändert lassen. Zum eingehenden Prüfen neuer Disketten können Schreib/Lesetests angewandt werden.

Logische Tests

Hier wird fileorientiert vorgegangen. Der FDS eines Files wird gelesen, anschließend wird anhand dessen Daten geprüft, ob sich jeder vom File belegte Sektor lesen lässt. Hierbei ist zu beachten, daß ein Lesefehler bei einem Sektorzugriff nicht zwangsläufig bedeutet, daß die Daten verloren sind. Eine Fehlermeldung wird

nämlich auch dann gegeben, wenn das Adressfeld in Ordnung ist und lediglich die CRC-Prüfung der Daten negativ war. Die Fehlercodes >22,>23 bedeuten, daß Daten vom Controller übergeben wurden, die zurückgeschrieben werden können, das Adressfeld also in Ordnung ist. Ein Fehlercode >11 deutet auf ein defektes Adressfeld hin, weshalb der Sektor nicht gefunden wurde und auch nicht zurückgeschrieben werden kann. Tritt dies bei einem FDS auf, dann ist das File in den meisten Fällen verloren. Ist irgendein Datensektor defekt, der nicht zurückgeschrieben werden kann, dann kann dieser im FDS 'ausgeclustert' werden, wodurch er keine Lesefehler mehr produziert. Ein fehlender Datensektor ist leichter zu verkraften als ein total unbrauchbares File.

Tests ohne Datenverlust

Tests dieser Art sind immer dann angebracht, wenn der fehlerhafte Datenträger bereits wichtige Daten enthält, die nicht gelöscht werden können. Dabei kann u.a. eine ggfs. vorhandene Laufwerksabhängigkeit eines Fehlers erkannt werden, was die Wiederherstellung der Daten erleichtern kann.

Oftmals ist es so, daß ein Fehler aufgrund einer mangelhaften Zentrierung der Diskette im beschreibenden oder lesenden Laufwerk auftritt. Dann sinkt der Lesepegel zu stark ab, was einen Datenverlust bewirkt. Ein erneutes Einlegen ins gleiche oder ein anderes Laufwerk kann hier schon hilfreich sein.

Physikalische Nur-Lese-Tests

Hierbei werden alle Sektoren der Diskette gelesen. Tritt ein Lesefehler auf, dann sollte versucht werden, den Sektorinhalt, auch wenn er falsch ist, wieder zurückzuschreiben. Bei einem defekten Adressfeld ist das nicht mehr möglich. Ist das Adressfeld aber in Ordnung, dann werden durch das Rückschreiben die Prüfsummenbytes wieder aktualisiert, so daß der nächste Zugriff auf diesen Sektor keinen Lesefehler mehr produziert. Soll überhaupt nicht geschrieben werden, so kann dieser Test nur anzeigen, wo was nicht i.O. ist.

Schreib-Lese-Tests

Jeder Sektor wird gelesen und wieder an die selbe Stelle geschrieben. Tritt beim Lesen oder Schreiben ein Fehler auf, dann wird der Vorgang für diesen Sektor wiederholt bzw. nach n Versuchen ein Fehler gemeldet.

Physikalische Mediumsprüfung

Bei dieser Prüfung können Disketten auf Bits und Bytes geprüft werden. Zuerst wird formatiert, dann wird jeder Sektor auf >E5 geprüft. Anschließend wird jeder Sektor gezielt geschrieben und gelesen. Besonders interessant ist eine Prüfung auf Seitenübersprechen. Hier wird bei doppelseitigen Disketten auf einen bestimmten Sektor einer Seite ein Muster aus lauter >00 geschrieben; auf den direkt gegenüberliegenden werden >A5 geschrieben (mehrfach schreiben!). Nun wird mehrere Male der Sektor mit dem Nullmuster gelesen, ob kein Bit 'umgefallen' ist. Wenn doch, dann kopieren Daten auf die andere Seite über und die Diskette sollte nur einseitig benutzt werden.

Welche Sektornummer der gegenüberliegende Sektor hat, kann einfach aus der Formatierungsart bei Kenntnis des Sektor-Interlace bestimmt werden. Im Zweifelsfall werden alle Sektoren einer Spur mit dem gleichen Muster beschrieben und alle geprüft.

Auf die gleiche Weise kann ein Spurübersprechen erfasst werden. Hier werden die Nullen auf den Sektor X geschrieben, die >A5 auf den Sektor X plus Y, wobei Y die Anzahl der Sektoren pro Spur ist. Dieser Test ist besonders bei 80 Spur Laufwerken interessant.

Wiederherstellung defekter Adreßfelder

Die Adreßfelder der Sektoren einer Spur werden beim Formatieren geschrieben und dann immer nur gelesen. Ist also so ein Adressfeld defekt, kann es nur durch formatieren der betreffenden Spur wiederhergestellt werden. Zuerst werden alle lesbaren Sektoren im RAM zwischengespeichert. Dann wird die Spur neu formatiert und die Sektoren werden, bis auf die ehemals defekten, wieder geschrieben. Soweit die 'brutale' Methode.

Besser ist es, die gesamte Spur zu lesen, die Sektordaten zwischenzuspeichern, die Spur in ein formatierungsfähiges Format zu bringen (Datenfeld mit >E5 füllen, CRC-Start/Stop-Bytes setzen etc.) das betreffende Adressfeld zu reparieren und die Spur nach dem Rückschreiben wieder mit den Sektordaten zu füllen.

Wie zu sehen ist, ist das nur für den weit fortgeschrittenen Anwender möglich, der die Hardware genau kennt. Dann ist aber schon einiges nötig, um irreparable Disketten zu produzieren (Magnetanschläge o.ä.).

Erzeugen defekter Sektoren

Dies ist eine relativ einfache Methode, eine Diskette mit einem 'Fingerabdruck' zu versehen. Man erzeugt beim Formatieren (natürlich mittels einer eigenen Formatierungssoftware) defekte Sektoren, macht im Programm Sektorzugriffe auf diese Sektoren (mit den Standard Level-1-Routinen) und prüft das Fehlerbyte. Danach kann eine etwaige Raubkopie erkannt werden.

Wie kann man das nun anstellen?

Erzeugen eines LOST-DATA Zustandes.

Hier wird ein Datenfeld mit der falschen Längenkennung erzeugt. Das kann z.B. ein mit 512 Byte angegebenes Feld sein, in das aber nur 256 Byte geschrieben werden. Die Daten werden richtig geschrieben und gelesen, es wird aber immer ein Fehler gemeldet.

Erzeugen nicht normierter Spuren.

Das ist recht einfach. Es werden die in den Datenblättern zum Controller-IC angegebenen Mindestparameter unterschritten, so beispielsweise der ID-GAP oder es werden alle Adreßfelder mit einer falschen Spurnummer versehen - die Möglichkeiten sind hier so vielfältig, daß der Platz nicht reicht. Das Ergebnis ist aber immer, daß die ganze Spur unlesbar wird.

Dies sind nur Beispiele, dem versierten Programmierer werden noch mehr bösartige Ideen kommen, einem Kopieren vorzubeugen. Dieses Kapitel befindet sich daher auch nur der Vollständigkeit wegen unter dem Punkt 'Diskmanager-Funktionen'.

Nutzung der Dienstprogramme in der Controller-DSR

Für den 'normalen' Betrieb benötigt die Disk-Controller-DSR diverse Routinen zur Hardware-Ansteuerung, Sektormanipulation und Bearbeitung von Daten auf File-Ebene. Daraus ergibt sich eine funktionelle Hierarchie, welche innerhalb von Diskinfo mit folgenden Bezeichnungen belegt wird:

- Level-0-Routinen - DSR-interne Ansteuerung des Floppy-Disk-Controller-Formatter-Chips (FDC)
- Level-1-Routinen - Extern verfügbare, jedoch auch intern genutzte Routinen zur Bearbeitung einzelner Sektoren
- Level-2-Routinen - Externe Routinen zur Behandlung kompletter Files

Die Level-0-Routinen sind extern nicht verfügbar, zumindest nicht über normierte Funktionsaufrufe wie DSR- oder SBRLNK. Diese Routinen arbeiten 'hart an der Hardware' und erfordern entsprechende Sachkenntnis bei der Anwendung. Da sie innerhalb der DSR als Teile der Level-1-Routinen genutzt werden, sind sie i.d.R. nur durch Nachbildung innerhalb eigener Programme nutzbar. Beispiele zur Programmierung finden sich in den Kapiteln 16 und 17 sowie in den ROM-Listings.

Die Level-1-Routinen werden über den standardisierten Aufruf DSRLNK mittels des Offset >A aktiviert und bearbeiten hauptsächlich Daten auf Sektor-Ebene (Ausnahme CALL FILES, >16). Selbst die Routinen >12 und >13, die auf den ersten Blick wie File-Operationen aussehen mögen, bearbeiten nur bestimmte Bytes des jeweiligen FDS (>13 noch dazu Sektor 1). Das Programmierinterface beschränkt sich auf einen rudimentären PAB (2 Bytes) sowie einige reservierte Adressen im PAD- und VDP-RAM.

Die Level-2-Routinen schließlich stellen die höchste Ebene der Funktionalität aller von der Disk-DSR zur Verfügung gestellter Programme dar. Entsprechend werden diese Routinen von der DSR selbst nicht genutzt, können also nur aus einer Anwendersoftware heraus mittels eines recht komfortablen Programmierinterface' genutzt werden. Die entsprechenden Beispielprogramme des Editor/Assembler-Handbuches sollen hier nicht wiederholt werden - vielmehr werden die Funktionen beschrieben, welche das E/A-Handbuch verschweigt.

Anhand des oben Gesagten kann vereinfachend festgehalten werden:

Die Level-0-Routinen dienen der Hardwaresteuerung, die Level-1-Routinen arbeiten sektororientiert und die Level-2-Routinen schließlich bearbeiten ganze Files bzw. Ausschnitte daraus.

Beispielprogramme zu den Level-1 Routinen

Vorbemerkungen

Obwohl die verwendeten Parameter für alle Diskcontroller identisch sind, so existieren doch einige geringfügige Unterschiede zwischen den Controllern verschiedener Hersteller. Sollte also ein Beispielprogramm Ärger machen, dann sollte zuerst geprüft werden, ob die hier gemachten Voraussetzungen beim verwendeten Controller zutreffen.

Der Pointer an >8356, der sog. Namelength-Pointer zum DSRLNK, wird bei den Level-1-Routinen genauso verwendet wie beim normalen DSRLNK - er zeigt immer auf die Länge des Routinennamens.

Die vorgestellten Programmbeispiele sind aber, wie der Name sagt, nur Beispiele und keine fertigen, jedem Anwendungsfall gewachsenen Universalroutinen. Sie sollen zeigen, wie man mit den DOS-Routinen umzugehen hat und die Grundlage weiterer, eigener Experimente sein. Um hier Anfangsfrustrationen vorzubeugen: alle Beispielprogramme sind geprüft und bei korrektem Abtippen lauffähig!

Subroutine >10, Sektor lesen/schreiben

Diese Routine wurde zwar bereits weiter oben angesprochen, soll aber zusammenfassend hier nochmals erwähnt werden. Die Routine benötigt einen exakt 256 Bytes großen Puffer irgendwo im VDP und verlangt folgende Pointer:

>834C (B) Laufwerknummer, >01 bis >04

>834D (B) I/O-Code, >00 - schreiben, sonst - lesen

>834E (W) Adresse das VDP-Datenpuffers, 256 Bytes

>8350 (W) Sektornummer

In >8350 wird das Fehlerbyte übergeben, siehe die entsprechende Liste in Kapitel 11.

Hier nun das Programm, das mit dem Start über READ den Sektor 150 in den Puffer im VDP-RAM ab >1100 liest und über WRITE die Daten des Puffers in den Sektor 150 schreibt:

Beispielprogramm SBR >10

```
* (c) 1986 Ch. Winter
*
       DEF   READ,WRITE
       REF   DSRLNK,VMBW
*
PABAD  EQU   >1000     Adresse des PABs im VDP-RAM
BUFFER EQU   >1100     Adresse des Datenpuffers
*
LW     BYTE  >01       Laufwerk 1
       EVEN
*
NAME   DATA  >0110     Subroutinenname
SEKTOR DATA  150       Sektornummer, beliebig
MYWS   BSS   32        Eigener Workspace
*
READ   LWPI  MYWS      Eigene Register laden
       SETO  R0        R0 = >FFFF, lesen indizieren
RW     MOVB  R0,@>834D           I/O-Code
       MOVB  @LW,@>834C          Laufwerknummer
       MOV   @SEKTOR,@>8350      Sektornummer
       LI    R0,BUFFER
       MOV   R0,@>834E      Adresse des Puffers angeben
       LI    R0,PABAD
       MOV   R0,@>8356      Zeiger auf das Namelength-Byte
       LI    R1,NAME        Position des SBR-Namens
       LI    R2,2           2 Byte verschieben
       BLWP  @VMBW          PAB ins VDP-RAM schaffen
       BLWP  @DSRLNK        Routine aufrufen
       DATA  >A             Routinenindex
       MOVB  @>8350,@>8350  Fehlerbyte prüfen
       JEQ   OK             kein Fehler
ERROR  ****  ****          hier Fehlerbehandlung einbauen
```

```
OK     ****  ****          hier Programm weiterführen
*
WRITE  LWPI  MYWS
       CLR   R0       Schreiben indizieren
       JMP   RW       alle weiteren Aktionen sind gleich!
*
       END            Ende des Beispielprogramms*
```

Die Sektor-I/O-Routinen von CorComp und Myarc weisen Besonderheiten auf: CorComp erlaubt es, den Datenpuffer nicht im VDP- sondern alternativ im CPU-RAM zu halten, was in einigen Fällen das Speicher-Herumgeschiebe von und nach VDP-RAM entbehrlich macht. Aktiviert wird diese Funktion durch Setzen des Bit 5 im System-RAM an Adresse >400B (mit >20 verODERn) sowohl für Lese- wie auch Schreib-Operationen. Zusätzlich kann die Verify-Funktion beim Schreiben von Sektoren deaktiviert werden, was jedoch außer der Gefahr des Daten-Harakiri keinen weiteren Sinn ergibt - Tempo gegen Datensicherheit aufzuwiegen (0,2 Sekunden Gewinn pro Sektor) ist generell nicht akzeptabel.

Auch hier erlaubt sich Myarc wieder eine Extrawurst, die sich keiner so richtig erklären kann: Beim 'Proofreading', dem Lesen des Sektors unmittelbar nachdem er geschrieben wurde, wird nicht etwa Byte für Byte mit dem Pufferinhalt verglichen, sondern der Sektorinhalt ins ROM-0 befördert. Wieso man diesen destruktivsten (und dümmsten) aller Wege beschritt, statt in den genauso schnellen Registerbereich im PAD zu schreiben, ist unklar. Klar ist, daß man sich so unter anderem das inkompatible DSR-Konzept des GENEVE einhandelte (alle Design-Fehler des Myarc-Controllers bedingen die Design-Fehler des GENEVE) und daß jeder, der seine Konsolen-ROMs durch RAMs ersetzt hat, von diesem Diskcontroller seine RAMs gelöscht bekommt.

Subroutine >11, Diskette formatieren

Auch diese Routine wurde bereits beschrieben. Die Pointer sind wie folgt zu benutzen:

>834C (B) Laufwerknummer

>834D (B) Spuranzahl (Hex)

>834E (W) Datenpuffer für den Spuraufbau, bei DD werden etwa 6,5 KByte benötigt

>8350 (B) Dichte, >01 - Single, >02 - Double Density

>8351 (B) Seiten, >01 - Single, >02 - Double Sided

Nach dem Formatieren werden folgende Werte übergeben:

>834A (W) Gesamtzahl aller Sektoren

>834D (B) Anzahl der Sektoren pro Spur

>8350 (B) Fehlercode

Nun das Beispielprogramm. Es formatiert die Diskette in Laufwerk 1 im Format Single Sided, Single Density:

Beispielprogramm SBR >11

```
* (c) 1986 Ch. Winter
*
       DEF  INIT
       REF  DSRLNK,VMBW
*
PABAD  EQU  >1000       Adresse des PABs im VDP-RAM
BUFFER DATA >1100       Adresse des Datenpuffers
*
LW     BYTE >01         Laufwerk 1
TRACKS BYTE 40          40 Spuren
*
NAME   DATA >0111       Subroutinenname
FCODE  DATA >0101       Formatierungscode, SS, SD
MYWS   BSS  32          Eigener Workspace
*
INIT   LWPI MYWS
       MOVB @LW,@>834C        Laufwerk
       MOVB @TRACKS,@>834D    Spuranzahl
       MOV  @BUFFER,@>834E    Spurpuffer
       MOV  @FCODE,@>8350     Formatierungscode
```

```
       LI   R0,PABAD
       MOV  R0,@>8356         Zeiger auf das Namelength-Byte
       LI   R1,NAME           Position des SBR-Namens
       LI   R2,2              2 Byte verschieben
       BLWP @VMBW             PAB ins VDP-RAM schaffen
       BLWP @DSRLNK           Routine aufrufen
       DATA >A                Routinenindex
       MOVB @>8350,@>8350     Fehlerbyte prüfen
       JEQ  OK                kein Fehler
ERROR  **** ****       hier Fehlerbehandlung einbauen
OK     **** ****       hier Programm weiterführen
*
       END             Ende des Beispielprogramms
*
```

Auch bezüglich der Formatierroutine gibt es Besonderheiten bei den CorComp- und Myarc-Produkten: So kann bei CorComp die Sektor-Interleave-Liste durch Setzen eines Steuerbits im System-RAM von der Default-Tabelle im ROM auf eine anwenderdefinierte Tabelle im PAD-RAM umgeleitet werden. Da jedoch die Standard-Interleave-Werte einen Kompromiß zwischen optimaler Datenrate und breiter Austauschbarkeit der Disketten darstellen (langsame Systeme lesen z.B. DD-Disks mit IL=3 nur mühsam), sollte diese Option auf wenige Spezialanwendungen beschränkt bleiben.

Myarc benötigt einen kleineren VDP-Puffer, da der Pre-Index-GAP (der sog. Spur-Trailer, also der Bereich nach dem letzten Sektor) 'von Hand' geschrieben wird. Auch hier bleibt die Frage nach dem Warum offen.

Subroutine >12, Fileschutz ändern

Diese Routine ermöglicht es, den Schreibschutz irgendeines Files zu setzen oder zu löschen. Es muß lediglich der Filename und das Laufwerk angegeben werden. Folgende Pointer werden verwendet:

>834C (B) Laufwerknummer

>834D (B) Schutzcode, >00 - nicht schützen, >FF - schützen

>834E (W) Zeiger auf den Filenamen im VDP-RAM

Der Filename muß 10 Byte lang sein, ggfs. muß ein kürzerer Name mit Leerzeichen aufgefüllt werden. Die Fehlercodes in >8350 entsprechen nun, da eine Fileoperation vorliegt, den Codes in der Tabelle der File-Errors! Das Programm setzt beim Einsprung über PROT den Fileschutz von 'FILENAME' und entfernt ihm beim Einsprung über UNPROT.

Nun das Beispielprogramm:

Beispielprogramm SBR >12

```
* (c) 1986 Ch. Winter
*
       DEF PROT,UNPROT
       REF DSRLNK,VMBW
*
PABAD  EQU >1000 Adresse des PABs im VDP-RAM
*
LW     BYTE >01 Laufwerk 1
       EVEN
*
NAME   TEXT 'FILENAME'     Filename (beliebig), 10 Bytes lang!
       DATA >0112          Subroutinenname
MYWS   BSS 32              Eigener Workspace
*
PROT   LWPI MYWS
       SETO R0             Fileschutz setzen
       JMP  PU             weitermachen
UNPROT LWPI MYWS
       CLR  R0             Fileschutz entfernen
PU     MOVB @LW,@>834C     Laufwerk
       MOVB R0,@>834D      Schutzcode angeben
       LI   R0,PABAD
       MOV  R0,@>834E      Zeiger auf Filenamen
       LI   R1,NAME        Position des Namens
       LI   R2,12          12 Bytes verschieben
       BLWP @VMBW          PAB ins VDP-RAM schaffen
```

```
       LI   R0,PABAD+10    Zeiger auf SBR-Namelength
       MOV  R0,@>8356      Namelength Pointer
       BLWP @DSRLNK        Routine aufrufen
       DATA >A             Routinenindex
       MOVB @>8350,@>8350  Fehlerbyte prüfen
       JEQ  OK             kein Fehler
ERROR **** ****            hier Fehlerbehandlung einbauen
OK  **** ****              hier Programm weiterführen
*
```

Subroutine >13, Filenamen ändern

Mit dieser Routine kann der Name eines bestehenden Files geändert werden. Der alte und der neue Name müssen im VDP-RAM übergeben werden. Beide müssen jeweils 10 Byte lang sein, ggfs. mit Leerzeichen auffüllen!

Die Pointer:

>834C (B) Laufwerknummer

>834E (W) Zeiger auf den neuen Filenamen

>8350 (W) Zeiger auf den alten Filenamen

Die Übergabe des File-Error Bytes geschieht wie gehabt in >8350 (B). Es kann nur ein bestehendes File umbenannt werden, das nicht schreibgeschützt sein darf!

Beispielprogramm SBR >13

```
*  (c)1986 Ch. Winter
*

       DEF CHGNAM
       REF DSRLNK,VMBW
*
PABAD  EQU >1000          Adresse des PABs im VDP-RAM
*
LW     BYTE >01           Laufwerk 1
       EVEN
*
NAME   TEXT 'NAME-ALT'    Filename (alt), 10 Bytes lang!
       TEXT 'NAME-NEU'    Filename (neu), 10 Bytes lang
       DATA >0113         Subroutinenname
MYWS   BSS  32            Eigener Workspace
*
CHGNAM LWPI MYWS
       MOVB @LW,@>834C    Laufwerk
       LI   R0,PABAD
       LI   R1,NAME       Position des Namens
       LI   R2,22         22 Bytes verschieben
       BLWP @VMBW         PAB ins VDP-RAM schaffen
       MOV  R0,@>8350     Zeiger auf den alten Namen
       LI   R0,PABAD+10   Zeiger auf den neuen Namen
       MOV  R0,@>834E
       LI   R0,PABAD+20   Zeiger auf Namenlänge
       MOV  R0,@>8356
       BLWP @DSRLNK       Routine aufrufen
       DATA >A            Routinenindex
       MOVB @>8350,@>8350 Fehlerbyte prüfen
       JEQ  OK
ERROR **** ****           hier Fehlerbehandlung einbauen
OK  **** ****             hier Programm weiterführen
*
       END                Ende des Beispielprogramms
```

Subroutine >14, File-Copy Input

Hierbei handelt es sich um eine funktionell sehr umfangreiche Routine, zu der die Routine >15 gehört. Beide Routinen ermöglichen es, ein komplettes File, wie viele Sektoren es auch umfassen mag, stückweise zu kopieren. Dementsprechend sind die über Pointer zwischen den beiden Routinen übergebenen Parameter von sehr komplexer Struktur.

Die Pointer:

- >834C (B) Laufwerknummer
- >834D (B) Sektoranzahl, Filestatus wenn >00!
- >834E (W) Zeiger auf den Filenamen (10-stelliger String)
- >8350 (B) Offset des Parameterblocks im PAD. Dieses Byte ist das niederwertige Byte eines Zeigers in das PAD-RAM, wo die File-Parameter stehen. Das höchstwertige Byte ist demnach >83.

Aufbau des Parameterblocks im PAD-RAM:

Byte	**Input**	**Output**
Bytes 0,1	Datenpufferadresse	
Byte 2,3	Sektoroffset bei dem begonnen werden soll, 0 entspricht dabei dem Anfang des Files.	Anzahl vom File belegter Sektoren
Byte 4		File-Status aus dem FDS
Byte 5		Datensätze pro Sektor
Byte 6		EOF-Offset im letzten Sektor
Byte 7		Satzlänge
Bytes 8,9		Anzahl der Einträge/Records (geswapt!)

Der Output in den Bytes 2 bis 9 geschieht nur, wenn die Anzahl der zu lesenden/kopierenden Sektoren >00 war. Damit kann das steuernde Programm den Status eines Files lesen, bevor mit dem Kopieren begonnen wird.

Nach Ausführung der Routine befindet sich in >834C (W) die Anzahl der tatsächlich gelesenen Sektoren, falls mehr als vorhanden spezifiziert wurden, ansonsten steht hier der in >834D angegebene Wert.

Das Kopieren eines Files geschieht dann in einer Art Schneeballverfahren, indem die Routinen >14 und >15 abwechselnd aufgerufen werden, bis keine Sektoren mehr zu kopieren sind. Es ist einfach möglich, immer die Anzahl von Sektoren, die kopiert werden sollen, anzugeben, die der VDP-Puffer maximal fassen kann. Ohne Rücksicht auf die tatsächliche Filelänge wird nun so lange kopiert, bis die Rückgabe in >834C (W) null ist. Natürlich muß der Startsektor eines Durchganges immer mitlaufen, also entsprechend der Puffergröße inkrementiert werden! Die Routinen prüfen selbst, wann das Ende des Files erreicht ist.

Die gelesenen Daten werden ab der im ersten Wort des Parameterblocks angegebenen VDP-Adresse in 256 Byte Blöcken, einer pro Sektor, aufsteigend abgelegt.

Die DOS-Routine >14 kann, wie o.a., nicht getrennt von der Routine >15 besprochen werden. Vor dem zusammenfassenden Beispielprogramm soll daher auf SBR >15 eingegangen werden.

Subroutine >15, File-Copy Output

Wegen der engen Beziehung zu SBR >14 sind die Pointer und deren Verwendung identisch, jedoch mit dem Unterschied, daß der File-Status ein neues File erzeugt, zumindest dessen Directory-Eintrag. Hierfür werden die Daten des PAD-Parameter Blocks verwendet.

Ebenso werden bei der Eröffnung des Zielfiles alle für die Länge des Files notwendigen Sektoren in der Sektor-Bitmap reserviert. Führt man nun nur einen Status-Transfer von der Quelle zum Ziel aus, so existiert ein ordnungsgemäßes File, dessen Datensektoren aber nicht die richtigen Daten enthalten.

Die Anzahl zu kopierender Sektoren wird nicht geprüft, hier sollte also der Rückgabewert der Routine >14 nicht verändert werden.

Sollte es passieren, daß das Zielfile den Namen eines bestehenden Files auf der Diskette hat, so wird die Routine mit einem File-Error abgeschlossen, der in >8350 zurückgegeben wird. Die Namen des Quellfiles und des Zielfiles (Source und Copy) müssen nicht übereinstimmen!

Bei jedem Durchlauf der Sequenz SBR >14 - SBR >15 wird immer wieder der FDS des entsprechenden Files gelesen. Das ist zwar lästig und wenig sinnvoll, aber nicht zu ändern, da Firmware. Es erfordert aber, daß die Filenamen immer übergeben werden und während des Kopierens die dem Quell- und dem Zielfile zugeordneten Namen immer gleich bleiben, bis das gesamte File kopiert ist!

Zum Beispielprogramm: Es ist diesmal etwas kompakter programmiert, um ein paar kleine Tricks zur Reduzierung des belegten Speicherplatzes zu zeigen. Das Programm kopiert in nur einem Durchlauf und maximal 32 Sektoren. Sollen längere File kopiert werden, so ist zu Anfang nach dem Status des Quellfiles dessen Sektoranzahl zu lesen und in jedem Durchlauf zu dekrementieren, wobei natürlich auch der Startsektor des jeweiligen Kopiervorgangs erhöht werden muß. Ende ist dann, wenn die Sektoranzahl 0 ist.

Der Kopiervorgang beginnt beim Sektoroffset 0 und endet mit dem letzten vom File belegten Sektor. Die Rückgabe der Anzahl der File-Sektoren (Status) wie im vorigen Absatz beschrieben, schließt den FDS nicht mit ein! Dieser wird bei der Übergabe der File-Parameter durch den Status-Aufruf erzeugt. Im Beispielprogramm wird dieser Wert direkt übergeben bzw. nicht geändert zwischen dem Aufruf der beiden Routinen hintereinander.

Es wurde auf jegliche Fehlerbehandlung verzichtet, diese muß bei Bedarf nach jedem Routinenaufruf durchgeführt werden. Da es sich hierbei um eine File-Operation handelt, werden die entsprechenden File-Error Codes in >8350 geliefert.

Es können nun während des Kopiervorgangs an zwei verschiedenen Stellen Fehler auftreten, die unterschiedlich behandelt werden müssen. Zuerst kann es sein, daß bei der Erzeugung des FDS des Zielfiles nicht alle notwendigen Sektoren reserviert werden können, daß also die Diskette bereits zu voll ist. In diesem Fall wird in >8350 der entsprechende Fehlercode >04 geliefert und es werden alle bisher für dieses File reservierten Sektoren gelöscht- die Zieldiskette bleibt unangetastet. Tritt beim Schreiben eines Datensektors ein Fehler auf, so bleibt der FDS unberührt. In diesem Fall muß das File anschließend gelöscht werden, um den Platz auf der Diskette frei zu machen.

Beispielprogramm Subroutinen >14 und >15

```
*
* (c) 1986 Ch. Winter

       DEF START
       REF DSRLNK,VMBW

PABAD  EQU >F00           PAB-Pufferadresse

LWA    DATA >0100         LW A Status
LWB    DATA >0200         LW B Status
LWA2   DATA >0120         LW A kopieren
PUFFER DATA >1000         Datenpuffer

FILE   TEXT 'FILENAME'    Filename
SUB    DATA >0114         SBR-Name, eigenes Label für Änderungen
MYWS   BSS 32             Workspace

START  LWPI MYWS
       MOV @LWA,@>834C    LW und Status
       BL @SBR2           Filestatus lesen
       INC @SUB           auf >0115
       MOV @LWB,@>834C    LW und Status
       BL @SBR2           Filestatus erzeugen
       DEC @SUB           auf >0114
       MOV @LWA2,@>834C
       BL @SBR1
       INC @SUB           auf >0115
       MOVB @LWB,@>834C   Sektoranzahl stehenlassen wenn unter 256!!
       BL @SBR1

       BLWP @0            Programmende

SBR1   CLR @>8302         Startsektor=0, nur ein Lauf!!
SBR2   LI R0,PABAD
       LI R1,FILE         Filename
       LI R2,12
       BLWP @VMBW
       CLR @>8350          Parameterblock ab >8300
       MOV @PUFFER,@>8300  Datenpufferadresse
       MOV R0,@>834E       Filenamenzeiger
       LI R0,PABAD+10
       MOV R0,@>8356       POINT
       BLWP @DSRLNK
       DATA >A
       RT

       END
```

Zum Beispielprogramm:

Die Daten LWA, LWB und LWA2 geben direkt die Laufwerknummer und den Operationsmodus an. Im ersten Aufruf von SBR2 wird also der Status des Files 'FILENAME' auf der Diskette in Laufwerk 1 ermittelt.

Dieses File muß existieren, andernfalls tritt eine Fehlerbedingung auf.

Direkt danach wird nach Änderung des Subroutinennamens die Routine >15 im Status-Mode aufgerufen, die mit den Daten im PAD-Block einen neuen FDS (File-Directory-Sektor) erzeugt und die Sektor-Bitmap aktualisiert.

Sodann wird wieder der Name repariert und SBR1 aufgerufen, die im Gegensatz zu SBR2 den Startsektor zu null setzt. Dabei ist LWA2 im Low-Byte die Maximalzahl zu lesender Sektoren enthalten.

Der tatsächliche Wert wird in >834C (W) zurückgegeben. Da dieser nur im Low-Byte stehen kann (kleiner gleich 32), kann die Ziellaufwerknummer ins High-Byte. Da der VDP-Puffer maximal etwa 15 kByte fassen kann, wird dieser Wert im Low-Byte auch nie größer als 60 werden!

Das Beispielprogramm kopiert also das File 'FILENAME' von Laufwerk 1 auf Laufwerk 2 mit demselben Filenamen. Die festgelegte Maximallänge von 32 Datensektoren sei hier nochmals erwähnt.

Damit ist die Beschreibung der File-Routinen beendet.

Subroutine >16, Call Files

Auch aus einem Assemblerprogramm heraus ist es möglich, dieses Kommando auszuführen. Je nach Diskcontroller sind hier sogar bis zu 16 gleichzeitig offene Files erlaubt (Atronic V1.0 und 1.1), ansonsten gilt das im Basic festgelegte Limit von maximal 9 gleichzeitig offenen Files.

Der Aufruf dieses Programms ist so trivial, daß hier kein Beispielprogramm aufgeführt wird. Es muß jedoch sichergestellt sein, daß zum Zeitpunkt, da diese Funktion aufgerufen wird, keine Datei mehr offen ist, da es sonst passieren kann, daß der entsprechende Puffer gelöscht wird (nur bei einer Reduzierung der Pufferzone).

Die Anzahl der Files (nicht 0) wird in >834C (B) übergeben, der Aufruf erfolgt mittels des Standard SBR-PABs wie er in den vorangegangenen Beispielen verwendet wurde. Die Zahl im FAC muß in hexadezimaler Notation angegeben werden, nicht etwa in ASCII. Wurde eine falsche Zahl angegeben, so erfolgt die Fehlermeldung in >8350 (>FFFF bei Fehler, >0000 wenn o.K.).

Weitergehende Funktionen

Diskettenkatalog im Standardverfahren

An einem solchen Programm wird wohl niemand verzweifeln, es sei denn er (sie) wüßte nicht, welche Daten ein Record der Datei 'DSKX.' beinhaltet. Da die Möglichkeiten, einen Diskettenkatalog darzustellen extrem vielfältig sind, soll hier nur das Prinzip gezeigt werden.

Zuerst wird die Datei im Format 'INPUT, INTERNAL, FIXED 38' normal eröffnet, wozu ein Standard-PAB verwendet wird. Zulässige Namen sind hier, je nach Controller 'DSK1.' bis 'DSK4.'.

Sodann werden die Datensätze in den Pufferbereich gelesen. Die gelieferten Daten gliedern sich wie folgt:

1. Byte:	Namenlänge.
2. Byte und folgende:	Disk- bzw. Filename.
dann	>08 Anzahl Bytes der folgenden Zahl.
dann	8 Byte im RADIX 100 Format.
	>08 nächste Zahl.
	8 Byte der nächsten Zahl.
	>08 letzte Zahl.
	8 Bytes der letzten Zahl.

Wie man sieht, ist der Output keineswegs immer FIXED 38, sondern variiert in der Länge je nach File- bzw. Diskettenname.

Im ersten Record werden die Diskettendaten übergeben. Der String zu Anfang enthält also den Diskettennamen. Die erste Zahl enthält eine Null im RADIX 100 Format, die zweite die Gesamtzahl aller Sektoren der Diskette und die letzte Zahl die Anzahl freier Sektoren.

In jedem weiteren Record werden Filedaten übergeben. Zuerst der Filename, dann der Filetyp, die Anzahl vom File belegter Sektoren inclusive seines FDS und zum Schluß die Datensatzlänge.

Es stellt sich nun als einziges Problem die Umwandlung einer RADIX 100 Zahl in eine Dezimalzahl. Vereinfacht wird dies dadurch, daß nur ganze Zahlen zwischen -9999 und 9999 übergeben werden.

Eine Zahl im Radix-100 Format besteht im TI aus 8 Bytes, wobei das Längenbyte nicht gezählt wird. Das erste Byte stellt den Exponenten dar, also die Potenz, in die 100 erhoben wird. Jedes folgende Byte gibt 2 Stellen einer Dezimalzahl an. Das bedeutet, daß der Wert eines jeden Bytes nur von >0 bis >63 betragen kann, entsprechend 0 bis 99 im Dezimalsystem. Die Bytes müssen also in Dezimalzahlen umgewandelt werden. Das erste, dem Exponenten folgende Byte gibt den Wert der Vorkommastellen an, alle 6 folgenden die Nachkommastellen.

Besondere Aufmerksamkeit verdient der Exponent. Dieser nimmt jedoch in diesem speziellen Fall nur den Wert 0 oder 1 an. Zur eindeutigen Erkennung negativer Mantissen wurde jedoch die Mitte des Zahlenkreises für den Exponenten auf >40 gelegt. Das bedeutet, daß ein Exponent von tatsächlich 0 als >40 erscheint, einer von 1 als >41 übergeben wird.

Negative Mantissen werden daran erkannt, daß das erste Wort der Radix 100 Zahl negativ ist (High-Bit gesetzt, siehe weiter unten). Das bedeutet aber, daß der Zahlenbereich für den Exponenten von >00 bis >7F festgelegt ist. Dementsprechend sind Zahlen nur im Bereich +/-100 E 63 darstellbar.

Vom Exponenten muß also >40 oder, um direkt einen ASCII-Wert zu erhalten, >10 abgezogen werden. Eine Anzeige des Exponenten ist aber nicht immer die sinnvollste, zumal im Falle der Datei 'DSKX.' die einzige aus dem Exponenten ableitbare Information die ist, ob die Zahl 4- oder nur 2-stellig ist.

Ist der Exponent Null (>40), so folgt dem Exponentenbyte nur ein einziges weiteres Byte, welches Zehner und Einer darstellt. Ein Exponent von Eins zeigt an, daß im folgenden Byte die Tausender und Hunderter stehen, auf das die Zehner und Einer folgen.

Liegen negative Zahlen vor, wie es beim Filetyp passieren kann, so ist das erste Wort der Radix 100 Zahl, also Exponent und Vorkommastellen, negativ (erstes Bit gesetzt). Dieses Wort liegt dann in der Zweierkomplement-Form vor und muß mit NEG in eine positive Zahl umgewandelt werden. Nun stimmt der Exponent wieder (0 oder 1) und vor der Anzeige der Zahl muß lediglich ein '-' ausgegeben werden.

Nach dieser Theorie nun einige Beispiele:

Freie Sektoren: 1438 (dezimal)

Radix 100 Zahl:

>08 - Längenbyte

>41 - Exponent 1, also Mantisse mal 100 hoch 1

>0E - dezimal gleich 14

>26 - dezimal gleich 38

5 mal >00 - Leerpositionen

Filetyp: -5, geschütztes PROGRAM-File

Radix 100 Zahl:

>08 Längenbyte

>BFFB 1. Wort negativ, Negation liefert

>40 Exponent 0

>05 dezimal gleich 5

6 mal >00 Leerpositionen

Weitere Beispiele finden sich im Manual zum Editor/Assembler Modul.

Für die Zahlenanzeige ist also eine Routine notwendig, welche Hexzahlen in vierstellige Dezimalzahlen umwandelt, am besten gleich als ASCII-String. Hier sei abschließend auf die XMLLNK/CFI-Routine verwiesen.

Liste der Fehlercodes bei Diskettenoperationen

Die Übergabe der Fehlercodes erfolgt nach Ausführung einer DOS-Routine entweder im Status-Byte des PAB's oder in >8350 (B) wenn eine Routine aufgerufen wurde, die keinen regulären PAB benötigt.

Fehlercodes im PAB sind immer File-Error Codes, Fehlerbytes in >8350 (B) können File-Error Codes oder Hardware-Error Codes sein.

Liste der Fehlercodes bei Sektor-I/O oder Formatieren

- >00 kein Fehler
- >06 Hardwarefehler, Laufwerk oder Controller nicht i.O.
- >07 Softwarefehler, Laufwerk- oder Sektornummer falsch
- >11 Spur nicht gefunden oder alle Schreib/Leseversuche sind fehlgeschlagen
- >21 Lesefehler, Record-Not-Found, RNF
- >22 Lesefehler, Prüfsummenfehler im Sektor, CRC
- >23 Lesefehler, Lost-Data Zustand, sollte nicht vorkommen, da dies signalisiert, daß die CPU mit dem FDC nicht Schritt halten konnte.
- >28 Verify-Fehler, ein Sektor konnte mehrfach nach dem Lesen nicht verifiziert werden.
- >31 Schreibfehler, Record-Not-Found, RNF
- >33 Schreibfehler, Lost-Data, siehe Code >23
- >34 Diskette ist schreibgeschützt

Die Fehler >11 bis >33 werden erst ausgegeben, nachdem mehrere Wiederholungen fehlschlugen. Das DOS läßt im Allgemeinen bis zu 10 Versuche (Retries) zu, bevor mit einer dieser Fehlermeldungen abgebrochen wird. Beim Fehler >22 befinden sich jedoch alle Daten des betreffenden Sektors im spezifizierten Puffer. War also lediglich die Prüfsumme 'umgefallen', so kann der Sektor durch einfaches Rückschreiben wieder repariert werden!

Liste der File-Error Codes

- >00 Kein Fehler oder falscher Gerätename. Hier muß zuvor das Equal-Bit direkt nach der Rückkehr von DSRLNK geprüft werden, ob der Fehlercode gültig ist!
- >01 Schreibgeschützt
- >02 Fehlerhafte Parameter bei der Eröffnung eines Files
- >03 Illegale Operation
- >04 Kein Platz für einen File-Puffer im VDP-RAM gefunden oder Diskette ist voll
- >05 Versuch, über EOF hinaus zu lesen
- >06 Allgemeiner Hardware-Fehler
- >07 File-Error, alle Fehler, die die vorangegangenen Codes nicht überdecken

Hardwareinformationen

Pinbelegung des Platinensteckers am Laufwerk

Die Laufwerke werden über einen 34-poligen Platinenstecker und ein 34-poliges Flachbandkabel mit dem Controller verbunden, das maximal 3 Meter lang sein darf. Die Kontakte mit den ungeraden Nummern befinden sich auf der Unterseite der Platine und führen alle Massepotential. Somit ist durch die Verwendung von Flachkabel und der entsprechenden Stecker sichergestellt, daß zwischen zwei Signalleitungen immer Masse liegt.

Die Kontakte mit geraden Nummer liegen folgerichtig an der Oberseite und führen Signalspannungen.

Die Belegung der Kontakte ist genormt, wenn auch die Stecker z.B. bei 3,5 Zoll Laufwerken anders aussehen! Man spricht hier von einem sog. Shugart-Bus, was die Kompatibilität durch Anlehnung an einen gemeinsamen Standard andeuten soll.

Im Folgenden nun die Bezeichnung der einzelnen Kontakte (Pins) und deren Funktion:

PIN	Out/In	Beschreibung der Funktion
2	O I	Optional, frei verfügbar
4	I	IN USE
6	I	DS3, Anwahl für Laufwerk 4
8	O	INDEX, Ausgang des Index-Sensors, 5 Hz nominal!
10	I	DS0, Anwahl für Laufwerk 1
12	I	DS1, Anwahl für Laufwerk 2
14	I	DS2, Anwahl für Laufwerk 3
16	I	MOTOR ON, direktes Anschalten des Motors, alle LW!
18	I	DIR, Richtung, in welcher der Kopf bewegt werden soll
20	I	STEP, Schrittimpulse an den Stepper Motor
22	I	WRITE DATA, serieller Datenstrom vom Controller
24	I	WRITE GATE, umschalten auf Schreibbetrieb
26	O	TRK 00, aktiv wenn der Kopf über Spur 0 steht
28	O	WRTPT, Write Protect, Schreibschutz aktiviert
30	O	READ DATA, serieller Datenstrom zum Controller
32	I	SIDE SELECT, Anwahl der Ober- bzw. Unterseite
34	O	READY, Laufwerk bereit, Drehzahl o.K.

Wellenwiderstand und Anpassung

Alle Laufwerke sind parallel über ein 34-poliges Flachbandkabel miteinander verbunden. Da die höchste zu übertragende Frequenz auf dem Kabel etwa bei 250 KHz liegt, muß der Wellenwiderstand an den Kabelenden mit dem des Kabels übereinstimmen. Der Wellenwiderstand von Flachkabeln kann in der Praxis mit etwa 100 Ohm angesetzt werden.

Die verwendeten Schaltglieder in TTL-Technik, für Ausgänge vorzugsweise 7438 und für Eingänge 74LS04 oder 74LS14, weisen jedoch typische Eingangswiderstände von ca. 1 kOhm auf.

Um nun bei der hohen Datenfrequenz ein Signalüber- oder allgemein -nachschwingen zu vermeiden, muß das Ende des Anschlußkabels, also am letzten Laufwerk, mit sogenannten 'Terminator'-Widerständen gegen +5V abgeschlossen werden. Diese sollten als DIP-Widerstände etwa 150 Ohm aufweisen. Es ist nur ein 'Terminator-Chip' notwendig, mehrere davon belasten die Treiber über Gebühr!

Besonders wichtig sind die korrekten Werte der Abschlußwiderstände bei modernen Laufwerken, deren Eingangsstufen mit CMOS-ICs bestückt sind, um deren hohe Eingangsimpedanzen zu kompensieren.

Signaltiming

Dieser besonders kritische Punkt ist in den jeweiligen Datenblättern zum Laufwerk beschrieben. Allgemein ist nur folgendes zu sagen, sofern das richtige Timing nicht einem Floppy-Disk Controller-Formatter überlassen wird:

- Vor einem Lesezugriff muß der Kopf über der richtigen Spur stehen, die richtige Seite der Diskette und das richtige Laufwerk angewählt und der Motor auf Nenndrehzahl sein. Nach dem Steppen muß dem

Kopf eine Beruhigungszeit von typisch 30 msec gewährt werden, um Trägheitseffekte abklingen zu lassen.

- Beim Schreiben muß zudem vor Beginn des Datentransfers der Schreibschutz geprüft und das Write-Gate aktiviert werden.
- Vor einem Step-Impuls muß die Richtungsleitung definiert und stabil sein, die Pulsdauer und Frequenz der Step-Impulse richtet sich nach dem verwendeten Laufwerk, wobei die Frequenz vom Anwender (Programmierer) festgelegt wird.

Näheres finden Sie in den Kapiteln 16 und 17.

Signalbeschreibung

Die Signalbeschreibung findet aus der Sicht des Laufwerkes statt. Alle Leitungen sind Low-Aktiv, was bedeutet, daß ihr logischer Zustand 'WAHR' ist wenn ein TTL-Low Signal anliegt.

Die Signale im Einzelnen:

Pins 6,10,12,14 DS0 bis DS3 - DRIVE SELECT

Mit diesen Leitungen wird eines der 4 möglichen Laufwerke direkt aktiviert. Es darf immer nur eine dieser Leitungen aktiv, also Low sein! Je nach Codierung des Laufwerks startet unmittelbar nach dieser Anwahl der Motor und wird der Kopf geladen, sofern ein Head-Load vorgesehen ist. Ggfs. muß der Motor über MOTOR ON vor dem Zugriff gestartet werden, um die eine Sekunde für das Hochlaufen auf die Solldrehzahl zu gewährleisten. Hätte man hier eine BCD-Codierte Anwahl vorgesehen, so könnten 15 Laufwerke angeschlossen werden.

Pin 8 INDEX - INDEX PULSE

Zur Regelung der Drehzahl und der Erkennung des Startbereiches einer Spur befindet sich das Indexloch in der Diskette. Damit wird der Strahlengang einer Lichtschranke periodisch geschlossen. Aufgrund der Umdrehungsgeschwindigkeit bei Minidisketten liegt an diesem Kontakt ein TTL-Signal mit einer Frequenz von 5 Hz an.

Pin 16 - MOTOR ON/MOTOR START

Diese Leitung ermöglicht den Start der Antriebsmotoren aller Laufwerke. Da jedesmal auf das Erreichen der Solldrehzahl gewartet werden muß, kann mit dieser Leitung bereits in Erwartung eines Diskettenzugriffs die Motordrehzahl stabilisiert werden.

Pin 18 - DIR/STEP-DIRECTION

Ist diese Leitung aktiv, also Low, so bewegt sich der Kopf nach einem Step-Impuls nach außen zum Rand der Diskette. Andernfalls wird auf die Diskettenmitte hin gefahren. Dieses Signal muß eine bestimmte Zeit vor einem Step-Impuls stabil sein.

Pin 20 - STEP/STEP-PULSE

Pro Impuls, dessen Dauer und Frequenz laufwerkspezifisch sind, bewegt sich der Schreib-Lesekopf um den radialen Abstand einer Spur weiter. Richtung durch DIR.

Pin 22 - WRITE DATA

Diese Leitung führt den seriellen Datenstrom vom Controller zum Laufwerk. Die Frequenz hängt vom Aufzeichnungsformat ab. Im Laufwerk selbst werden diese bereits codierten Rechtecksignale auf der Diskettenoberfläche aufmagnetisiert.

Pin 24 - WRITE GATE

Hiermit wird von Lesen (Inaktiv, High) auf Schreiben (Aktiv, Low) umgeschaltet. Nach Änderung dieses Pegels muß der Elektronik des Laufwerkes Zeit gegeben werden, alle Einschwingvorgänge zu beenden. Ist die Diskette schreibgeschützt, so ist bei einigen Modellen (nicht allen!) die Schreibelektronik abgeschaltet. Vorher muß also WRTPT geprüft werden.

Pin 26 - TRK 00/TRACK ZERO

Ist diese Leitung aktiv, so steht der Kopf über Spur Null, also ganz am äußeren Rand der Diskette. Außer dieser Rückmeldung ist zu keiner Zeit vom Laufwerk zu erfahren, wo der Kopf steht!

Die erste Aufgabe beim erstmaligen Ansprechen eines Laufwerkes ist demnach: DIR auf 'nach außen' setzen und so lange Step-Impulse geben, bis das TRK 00 Signal aktiv ist (hierzu gibt es den FDC-Befehl RESTORE, siehe dort). Passiert nach mehr als 80 (40) Impulsen nichts, dann ist das Laufwerk defekt.

Wurde TRK 00 aktiv, so muß der Rechner nach jedem Step-Impuls ein eigenes Register mitlaufen lassen, um immer zu wissen, wo der Kopf gerade steht. Ein FD-Controller kann dies mit einem 'Verify on track' anhand der Formatierung kontrollieren und bei Bedarf selbsttätig ein Spurregister auf den aktuellen Stand bringen.

Pin 28 - WRTPT/WRITE PROTECT

Hier wird angezeigt, ob eine Diskette schreibgeschützt ist oder nicht (aktiv oder inaktiv). Bei einer schreibgeschützten Diskette, also einer Diskette, bei der die entsprechende Kerbe zugeklebt wurde, ist kein Schreiben möglich, da die Schreibelektronik (des FDC und evtl. des Laufwerks, s.o.) blockiert ist.

Pin 30 - READ DATA

Analog zu Pin 22 liefert hier das Laufwerk die aus den Flußwechseln 'demodulierten' Daten als TTL-Signale zum Controller.

Pin 32 - SIDE SELECT

Wenn aktiv, dann wird mit dieser Leitung die Unterseite der Diskette angewählt. Bei einseitigen Laufwerken ist dieser Pin N.C., ein Pegelwechsel an dieser Leitung bewirkt bei rein einseitigen Laufwerken nichts, bei solchen mit fehlendem zweiten Kopf (optional) kann es sein, daß bei falscher Seitenwahl keine Daten übertragen werden können.

Pin 34 - READY/DRIVE READY

Hiermit wird angegeben, ob das Laufwerk bereit ist. Diese Funktion ist optional und nicht bei jedem Laufwerk bzw. Controller anzutreffen. Wenn doch, dann wird im Allgemeinen nur angezeigt, daß die Drehzahl erreicht wurde.

Pins 2,4 - SPARE-PINs

Ihre Funktionen sind ebenfalls optional und nicht immer verfügbar. Meist werden diese Signale nur bei Laufwerken für PCs eingesetzt, z.B. für ein Media-Change (Diskettenwechsel), was bei modernen DSRs für den TI auch recht nützlich wäre.

Ein Vielzahl der mit der korrekten Ansteuerung von Laufwerken verbundenen Aufgaben wird vom Floppy-Disk-Controller-Chip übernommen, so daß der Programmierer nicht direkt auf der Hardwareebene arbeiten muß.

Prozessor-Interface

Das verbindende Element zwischen CPU und Floppy-Disk-Laufwerk stellt ein hochintegrierter Baustein mit Namen

Floppy-Disk-Controller/Formatter

dar, auch in Kurzform 'FDC' genannt. Wie er sich softwareseitig der CPU präsentiert und im Detail angesprochen wird, erfahren Sie in den Kapiteln 14 bis 16. Die Hardware stellt dieses Kapitel vor.

Vom FDC übernommene Aufgaben

Nicht alle Leitungen, von denen bisher die Rede war, sind der CPU direkt zugänglich; einige davon sind nur über den FDC beeinfluß- bzw. lesbar. Direkt damit hängt es zusammen, wenn der FDC der CPU einige Aufgaben abnimmt bzw. es der CPU unmöglich ist, manche Steuerfunktionen ohne Mithilfe des FDC auszuführen.

Kopfpositionierung

Alle notwendigen Bewegungen des Schreib/Lesekopfes werden vom FDC ausgelöst. Dabei sorgt der FDC für die Einhaltung der Wartezeiten zwischen Schalten des DIR-Pegels und Ausgabe der Step-Impulse. Die Leitungen DIR und STEP sind von der CPU über den FDC nur indirekt beeinflußbar.

Datentransfer

Für die CPU erfolgt der Datentransfer vom und zum Laufwerk parallel, jeweils ein Byte (8 Bit) auf einmal. Der FDC liefert und empfängt die Laufwerk-Daten jedoch seriell, noch dazu codiert. Das von der CPU gelieferte Byte wird in ein, der CPU unzugängliches, Schieberegister übernommen (DATA-SHIFT-REGISTER), aus dem die Daten seriell herausgeschoben und zum Laufwerk übertragen werden. Beim Lesen von Diskette werden die seriellen Daten decodiert und im gleichen Schieberegister gesammelt. Ist ein Byte komplett, so wird es in das der CPU zugängliche Datenregister kopiert.

Synchronisation des Datentransfers

Der CPU muß beim Datentransfer, ganz gleich in welche Richtung, mitgeteilt werden, wann das Datenregister neu geladen bzw. wann es gelesen werden kann. Hierzu verfügt der FDC über 3 prinzipiell verschiedeneMöglichkeiten.

1. Er kann der CPU mittels eines Bits im Statusregister anzeigen, wann das Datenregister bereit für einen neuen Zugriff ist (DRQ-Bit). Dies wird im POLLING-MODE ausgewertet. Eine Variante testet per CRU laufend den Pegel am DRQ-Pin (Myarc).
2. Er legt den logischen Zustand dieses Bits an einen Anschluß gleichen Namens (DRQ-Pin) und stoppt den Prozessor nach einen FDC-Registerzugriff solange, bis ein Byte gewandelt wurde. Dies ist der WAIT-STATE-MODE.
3. Er gibt über den DRQ-Pin und entsprechende Hardware die Anforderung eines direkten Speicherzugriffs aus. Dies ist der DMA-MODE.

Es sollte angemerkt werden, daß es natürlich nicht reicht, nur DRQ zu testen, wie auch immer das geschehen mag. Natürlich muß auch INTREQ einbezogen werden, da schließlich auch Schreib-/Lesefehler auftreten können, die nicht über DRQ, wohl aber über INTREQ gemeldet werden!

Der Polling-Mode ist nur bei solchen Prozessoren möglich, die innerhalb der sehr kurzen Wandlungszeit eines Bytes mehrere Maschinenbefehle abarbeiten können. Dies ist beim TI aufgrund verschiedener geschwindigkeitsmindernder Hardware-Schaltungen nicht möglich (Myarc probiert es mit mittlerem Erfolg in der CRU-Polling-Variante doch, siehe ROM-Listing im Kapitel ROM-Listing MYARC DDCC-1ab Seite 122).

Beim Status-Register-Polling wirkt sich eine Eigenart der 1773/72/71 aus, die nicht leicht zu beherrschen ist.

Nachdem ein Befehl in das Kommandoregister geschrieben wurde, muß abhängig von der eingeschalteten Dichte eine unterschiedliche Zeit verstreichen, innerhalb der kein Statusregisterzugriff erfolgen darf.

Wird diese Regel mißachtet, so werden einige Bits im Statusregister 'wackelig', so daß es z.B. Checksum- oder RNF-Fehler gibt, die an sich keine sind, aber wer kann das im konkreten Fall entscheiden.

Bei den Controllern von Atronic, CorComp und TI wird das Wait-State-Verfahren angewendet. Dabei ist es nach Freigabe der entsprechenden Logik möglich, daß der FDC die CPU dann in Wartezyklen zwingt, wenn diese auf das noch nicht bereite Datenregister zugreift. Eine Synchronisation wird dann derart erreicht, daß

von Seiten des steuernden Programms gewährleistet ist, daß die CPU nach Abholung bzw. Anlieferung des vorangegangenen Bytes immer wieder vor Ablauf einer Byte-Wandlungszeit auf das Datenregister zugreift. Die Wait-State-Logik wird immer unmittelbar vor Beginn eines Datentransfers aktiviert und sofort nach Bearbeitung des letzten Bytes wieder abgeschaltet.

Das Polling-Verfahren wird trotz aller Risiken beim Myarc-Controller angewandt. Hier werden einige Parameter zur Befehlsart übergeben, die Bearbeitung gestartet und mittels Abfrage von CRU-Bit 0 auf das Ende der Operation gewartet, wobei in der Zwischenzeit die Daten in der angegebenen Richtung übergeben werden. Durch die CRU-Abfrage wird eine höhere Abfragefrequenz erreicht, als dies bei einer Bitmaskenabfrage möglich wäre. Nachteilig ist trotzdem die hohe Anzahl asynchroner Prozessorzyklen im Gegensatz zu einer Synchron-Methode über Hardware-Wait-States!

Datentransferrichtung

Wie bekannt sein wird, findet der Datentransfer zwischen CPU, FDC und Laufwerk generell in 2 verschiedenen Richtungen statt. Es werden entweder Daten geschrieben oder Daten gelesen. Dabei müssen die Laufwerke verschieden angesteuert werden.

Beim Lesen von Daten muß der Schreibverstärker der Laufwerkelektronik ausgeschaltet werden und es müssen die Impulse der Leitung READ-DATA gesammelt, decodiert und in Parallel-Form gewandelt werden. All dies erledigt der FDC.

Prinzipiell genauso ist der Vorgang beim Schreiben auf Diskette. Hier muß die Schreibelektronik aktiviert werden (WRITE GATE) und der Schreibschutz geprüft werden. Bevor ein Schreibbefehl ausgeführt wird, wird der Pegel der Leitung WRTPT vom FDC geprüft. Ist der Schreibschutz aktiviert, dann wird jeder Schreibbefehl abgebrochen. Wird WRTPT während eines laufenden Schreibbefehls aktiv, so wird nicht abgebrochen, sondern der Sektor/die Spur fertig geschrieben.

Statusmeldungen des Laufwerks

Folgende Signalleitungen des Laufwerks kann die CPU über das Status-Register des FDC abfragen:

READY, TRK 00, WRTPT, MOTOR ON, HEAD LOADED, INDEX.

Dabei ist zu beachten, daß die Signale bereits mit positiver Logik vorliegen, also vom Laufwerk kommend bereits invertiert wurden. Nicht alle FDC-Typen lassen die Abfrage aller aufgeführten Signale zu. Dies wird bei der Besprechung der Status-Register-Zustände in Kapitel 16 genau beschrieben.

Von der CPU auszuführende Arbeiten

Der FDC arbeitet immer nur auf dem Laufwerk, der Spur und der Seite der Diskette, wie es die CPU vorgab. Das bedeutet, daß die CPU neben den eigentlichen Aufgaben beim Datentransfer und der Erteilung von Befehlen an den FDC noch dafür sorgen muß, daß immer das korrekte Laufwerk zugeschaltet ist, der Schreib/Lesekopf korrekt positioniert ist, der passende Kopf (bei zweiseitigen Laufwerken) geschaltet ist und die korrekte Speicherdichte gewählt wurde. Erst nach Abschluß dieser Einstellungen kann mit einem Befehl an den FDC mit einem Datentransfer begonnen werden.

Die CPU kann dabei direkt die folgenden Leitungen beeinflussen:

DRIVE SELECT 0 bis 3, MOTOR ON, SIDE SELECT, HEAD LOAD (IN USE)

Zur programmtechnischen Manipulation dieser Leitungen sei hier auf Kapitel 16ff verwiesen.

Die verschiedenen FDC-Typen in TI-Diskcontrollern

Dieser Abschnitt soll einige Informationen zu den verschiedenen FDCs geben und Unterschiede in deren Interfacing zu CPU und Laufwerk zeigen.

In allen TI-Diskcontrollern, ob nun dem Original von TI, den CorComp-Typen, Atronic-Controllern (Box oder Stand-Alone), CPS99 oder Myarc, werden FDCs der Firma Western Digital eingesetzt. Folgende Typen kommen zum Einsatz:

Typ-Bezeichnung	Einsatz in	Besonderheiten
WD 1771-01	TI-Controller	nur einfache Dichte, eigenes Schrittmotor-Interface, eigene Head-Load Steuerung, eingebauter Daten-Separator, Schreibstromreduktionssignal, 2 Mhz Takt, invertierter Datenbus
WD 2793-02	CorComp (alt)	SD und DD möglich, eigene Head-Load Steuerung, PLL als Daten-Separator, Schreibstromreduktionssignal, 1 oder 2 MHz Takt, extern einstellbare Schreib-Vorkompensation
WD 1772	Myarc	SD und DD möglich, Gehäuse nur 28 polig, digitaler Daten-Separator, Schreib-Vorkompensation per Software schaltbar, Hohe Step-Raten, 8 Mhz Takt
WD 1773	CorComp (neu)	SD und DD möglich, Gehäuse Atronic (alle nur 28 polig, digitaler Typen) Daten-Separator, Schreib-Vorkompensation per Hardware schaltbar, softwarekompatibel zu 2793, 8 MHz Takt

Alle Unterschiede zwischen den Controller-Chips betreffen nur die Laufwerkseite. Auf der CPU-Seite sind die Unterschiede gering. Diese Leitungen auf CPU-Seite sind:

/CS	Chip-Select Decodiert die 5 Register des Chips im Adreßraum der CPU
R/W	Read/Write Schreiben bzw. Lesen der Register
A0	Address Bit 0 Adressleitung für Registerauswahl
A1	Address Bit 1 dto.
DAL	0 - 7 Data Lines Datenleitungen für Datenaustausch mit CPU
/MR	Master Reset Hardware-Reset, softwareseitig über einen Interrupt realisierbar
/DDEN	Double-Density- Speicherdichte umschalten Enable
DRQ	Data Request Zeigt an, daß auf das Datenregister zugegriffen werden muß/kann
INTRQ	Interrupt Request Gibt nach Befehlsausführung diese an und zeigt an, daß Daten im Statusregister verfügbar sind
/WE	nur 1771 und 2793 Indiziert einen Schreibzugriff der CPU. Kann direkt an R/W ange schlossen werden
/RE	nur 1771 und 2793 Indiziert einen Lesezugriff der CPU. Kann über einen Inverter an R/W angeschlossen werden

Die FDCs der Typen 1772 und 1773 stellen die modernsten Typen dar, die im TI eingesetzt werden. Sie haben nicht mehr das 40-polige Gehäuse und benötigen nicht mehr so viele externe Bauteile. Besonders kritisch ist der WD 2793-02, der die Einstellung von Schreib-Vorkompensation (Write Precompensation) und Leseimpulsbreite über externe Potentiometer erfordert. Diese Serie wurde von CorComp kurz nach Erscheinen der ersten Exemplare, die fast alle nicht korrekt arbeiteten, eingestellt.

Erkennbar sind diese Karten am beigen Gehäuse und der alten CorComp-Adresse in Anaheim. Leider hat aber auch der 1773 ein Problem: Vermutlich aufgrund eines Maskenfehlers, liest er bei FM-codierten Signa-

len die Adreßfelder gerne mit einem CRC-Error, obwohl es keine gibt. Diese Eigenart sollten alle Programme beachten, die mit einem 1773 Adreßfelder in FM lesen müssen!

Aufzeichnungsverfahren

Wollte man sich grundsätzlich zu Datenaufzeichnungsverfahren auslassen, so könnte hier eine neue Ausarbeitung begonnen werden.

Gemeinsam ist allen Verfahren jedoch, daß mit ihrer Hilfe die binärenInformationen des Rechners derart umgewandelt werden, daß sie den derzeit verwendeten analogen Speichermedien optimal angepasst sind. Ziele sind dabei hohe Datensicherheit, hohe Transfergeschwindigkeiten und Portabilität, was im konkreten Fall mit Austauschbarkeit übersetzt werden kann.

Bei der Datenaufzeichnung dominieren 2 Verfahren. Systeme mit Trägerschwingung (Cassettenrecorder, auch Standard-DFÜ) und getaktete Systeme. Beide arbeiten bitseriell.

Die geträgerten Systeme basieren auf einer Grundfrequenz, dem sog. Carrier, und werten Frequenzabweichungen in die eine oder andere Richtung als die Übertragung einer Null oder Eins. Dabei wird im Allgemeinen nach jeder Datenzelle (üblicherweise 1 Byte), synchronisiert. Schlagworte hierzu sind: FSK (Frequency-Shift-Keying, Frequenzumtastung), Manchester-Verfahren und Turbo- bzw Hypertape.

Als Nachteile sind zu nennen: die geringe Transfergeschwindigkeit der klassischen Systeme (Einige Derivate werten keine Momentanfrequenzen mehr aus und zählen Perioden unterschiedlicher Dauer, was eine starke Geschwindigkeitserhöhung bewirkt), sowie die starke Sensibilität gegenüber Bandgeschwindigkeitsschwankungen bzw. Frequenzverwerfungen. Vorteile der Standardmethode mit analoger PLL ist die fast vollständige Unempfindlichkeit gegenüber Pegelschwankungen.

Diese Systeme erreichen ihre physikalischen Grenzen jedoch sehr schnell, da es eine Relation zwischen Trägerfrequenz und maximaler Modulationsfrequenz gibt. Bei den klassischen Verfahren mit Frequenzmodulation ist die maximale Datenrate etwa ein Zehntel, bei den Pulsdauerverfahren etwa ein Drittel der Carrierfrequenz. Eine Erhöhung der maximalen Datenrate durch Erhöhen der Trägerfrequenz scheitert an dem begrenzten Frequenzgang billiger Recorder und den mit zunehmender Annäherung an die Grenzfrequenz steigenden Phasenfehlern.

Die getakteten Systeme trifft man bei Floppy-Disk Systemen an. Hierbei werden die Datenbits des Rechners mit einem festgelegten Takt als Magnetisierungsimpulse zusammen mit den Taktsignalen auf dem Medium abgelegt. Es wird also keine eigentliche Trägerschwingung benutzt, der die Daten aufmoduliert werden. Dadurch wird die Transfergeschwindigkeit bis auf den physikalischen Grenzwert erhöht. Zudem wird quasi bei jedem gelesenen Bit eine Information an die Taktrückgewinnung gegeben, wodurch die Datensicherheit steigt.

Randbedingungen

Unterteilt man die Oberfläche einer Diskette in Sektoren, um die Daten in kleinen Portionen besser bearbeiten zu können, so stellt sich sofort die Frage "Wie findet der Rechner die Sektoren wieder?". Gleich zu Anfang dieses Kapitels soll festgestellt werden, daß diese Aufgabe nicht eigentlich dem Rechner angelastet wird, sondern von dem 'Floppy-Disk-Controller-Formatter Chip'-übernommen wird (FDC). Diesem, mit beschränkter Intelligenz ausgestatteten Chip teilt der Rechner mit, welchen Sektor auf welcher Spur er gerne lesen würde. Der FDC sucht nun innerhalb der Spur, über der der Kopf der Diskettenstation positioniert wurde, nach diesem Sektor.

Wie bereits im Kapitel 4 beschrieben, befinden sich hierzu einige zusätzliche Informationen auf der Diskette, die sog. Adreßfelder. All das wurde also bereits besprochen.

Was kann aber alles passieren, wenn ein Sektor gesucht wird?

In dem Moment, wenn der Schreib/Lese-Kopf sich auf die Diskettenoberfläche senkt, kann ja nicht angenommen werden, daß sich die Diskette an einem ganz bestimmten Ort befindet, ja im Allgemeinen wird der Kopf sicher mitten in einem Datensektor landen. Es müssen sich also zwischen den einzelnen Datensektoren Lücken befinden, anhand derer erkannt wird (vom FDC), wo ein Sektor beginnt bzw. endet. Diese Lücken werden dann auch dazu verwendet, den Taktgenerator der Leseelektronik auf den aufgezeichneten Schreibtakt zu synchronisieren. Zudem werden besondere Bytes auf die Diskette geschrieben, bei denen 2 Taktimpulse in der Mitte des Bytes fehlen. Durch diese 'Law-Violation' (Regelverletzung) erkennt der FDC ein ID- oder Adreßfeld. Damit synchronisiert sich der FDC auf den Start der jeweiligen Datenbytes ein.

Dieses relativ komplizierte Verfahren entstand erst einige Zeit nach der umfassenden Einführung von Disketten (damals aufgrund technologischer Probleme in der Hauptsache 8 Zoll), weshalb man sich anders behalf. Um den Start eines Sektors zu finden, wurden so viele Indexlöcher in die Diskette gestanzt, wie diese Sektoren in einer Spur besaß. Um nun den Sektor am Anfang einer Spur zu kennzeichnen wurde ein zusätzliches

Index-Loch gestanzt, welches nicht in die Reihenfolge passte. Nun wurde ein Hardwarezähler von dem durch die Indexlöcher erzeugten Takt permanent weitergeschaltet und bei Erkennen der Taktverletzung durch das Zusatzloch auf Null gesetzt. Da der Rechner den Zähler auslesen konnte (resp. der FDC), war jederzeit klar, wo sich die Diskette befand. Damit waren zwar die Sektorzahlen festgelegt, die Datendichte konnte aber so um fast ein Drittel erhöht werden. Dieses Verfahren nennt man 'Hard-Sektorierung'; man findet es im Allgemeinen nur bei 8 Zoll Disketten.

Doch nun wieder zu den 'Soft-Sektorierten' Disketten.

Würde man auf die Lücken verzichten, so entstünden neben einer fehlenden Synchronisation auch Probleme beim Schreiben von Sektoren. Da die Umdrehungsgeschwindigkeit des Antriebsmotors nicht immer konstant ist, würde z.B. ein Sektor, der in einer Phase geringer Geschwindigkeit auf kleinem Raum geschrieben wurde, beim Rückschreiben unter höherer Umdrehungsgeschwindigkeit die nachfolgenden Daten zerstören. Die Lücken dienen also auch dem Auffangen von Drehzahlschwankungen.

Wurde nun mittels der Vorgabe vom Rechner der passende Sektor gefunden, so muß dieser immer noch gelesen bzw. geschrieben werden. Auch dabei können Fehler auftreten, entweder durch einen kurzzeitigen Einbruch der Umdrehungszahl oder durch einen Drop-Out, der z.B. durch ein Staubkorn verursacht wurde. Es ist also nötig, auf irgendeine Art zu prüfen, ob die Daten richtig gelesen wurden. Dies geschieht, wie bereits zuvor angesprochen, mithilfe der CRC-Bytes, die jedes Adressfeld und jeden Datensektor kontrollieren. Schreib- und Lesefehler müssen vom DOS einkalkuliert werden, üblich sind 5 bis 10 Versuche die fehlschlagen müssen, ehe endgültig eine Fehlermeldung gegeben wird.

Um einen Sektor zu finden, kann es dann notwendig sein, eine gesamte Spur abzusuchen, wenn nämlich mitten in dem gesuchten Sektor der Kopf aufgesetzt hat - der Erwartungswert liegt jedoch bei einer halben Umdrehung, die beim Zugriff auf einen einzelnen Sektor abgewartet werden muß.

Realisierung

Im Zusammenhang mit der Aufzeichnung der Daten auf Floppy-Disks fällt des öfteren das Wort 'Frequenzmodulation', oft auch als FM abgekürzt. Bei der moderneren Aufzeichnung mit doppelter Dichte taucht dann MFM auf, was soviel wie 'Modifizierte Frequenzmodulation' bedeutet. Gemeinsam ist beiden Verfahren, daß sie nichts mit der tatsächlichen Frequenzmodulation zu tun haben, sondern lediglich der Effekt unterschiedlicher Modulationsfrequenzen bei 0 und 1 diese Namensgebung be wirkte. Wieso es nun dazu kam, soll im folgenden Abschnitt erklärt werden.

Betrachtet man die Übertragungsgeschwindigkeit von 250000 bzw. 500000 Bits pro Sekunde, so wird klar, daß eine klassische FM eine Trägerfrequenz von mindestens 2,5 MHz benötigt, wenn der Modulationsindex unter 10 bleiben soll. Eine so hohe Mittenfrequenz verteuert jedoch die ganze Schreib-Leseelektronik, ganz abgesehen von den extremen Anforderungen an das Diskettenmaterial.

Bei der Aufzeichnung auf der Floppy-Disk werden lediglich rein digital bestimmte Bereiche durchmagnetisiert. Es existieren also nur Flußwechsel mit positiver oder negativer Flanke, wobei diese Flußwechsel so gesteuert werden, daß im Ausgangssignal kein Gleichspannungsanteil auftritt, der bei dieser Modulationsart ohnehin nicht übertragen werden kann.

Daneben ist es notwendig, neben den reinen Daten auch den bei der Aufzeichnung verwendeten Datentakt mit abzuspeichern. Es kann nämlich nicht davon ausgegangen werden, daß der Takt, mit dem geschrieben wurde, auch beim Lesen vorliegt, ganz besonders nicht bei Fremddisketten. Wollte man davon ausgehen, so müssten wegen der hohen Datenrate alle Floppy-Disk-Drive Antriebsmotoren mit einer quarzstabilisierten Regelung ausgestattet sein! Diese nebenbei auch teure Lösung kann durch die Aufzeichnung des Taktes umgangen werden.

Dabei synchronisiert die PLL des FDC auf die aufgezeichnete Datenrate und stellt somit sicher, daß alle Bytes richtig 'rüberkommen'.

Zudem muß noch unterschieden werden zwischen den Daten, die vom FDC kommen und den Daten, die die Elektronik des Laufwerkes dann tatsächlich auf die Diskette bringt.

Aus den Daten, die der Rechner parallel an den FDC liefert, wird ein serieller Datenstrom, der aus Taktsignalen und dazwischengesteckten Datenimpulsen besteht. Bei FM liegt am Ausgang des FDC ein 250 kHz Signal (Frequenz im Mittel!) an, das den Takt repräsentiert. Soll nun eine 1 geschrieben werden, so liegt zwischen zwei Taktimpulsen noch ein Impuls, bei einer 0 jedoch keiner. Die Elektronik im Laufwerk produziert nun bei jedem Impuls, ob Takt oder Daten, einen Flußwechsel in der Magnetschicht der Diskette, wobei die Richtung alternierend ist (s.o.). Beim Lesen von Diskette wird dann wieder bei jedem Flußwechsel ein Impuls an den FDC gesendet, der den Takt abtrennt und die Daten aus den Lücken holt.

Nun wird auch klar, warum man dieses Verfahren FM nennt. Bei einer dauernden Aufzeichnung von lauter Nullbits wird ein Signal mit der Taktfrequenz auf die Diskette gebracht, bei einer 1-Folge liegt die doppelte Frequenz vor.

Damit ist aber auch der Nachteil dieses Systems deutlich: es wird viel zu viel Energie auf die Speicherung des Taktes verwendet. Auch stellt das Frequenzverhältnis 1:2 recht hohe Anforderungen an die Taktrückgewinnung, wodurch die maximale Bitrate reduziert wird.

Abhilfe schafft hier das MFM-Verfahren. Hierbei lag die Überlegung zugrunde, daß es unnötig sei, bei einer 1-Folge auch noch den Takt zu senden, wo ja ohnehin die Taktfrequenz auch dann auftaucht, wenn man ihn weglässt. Probleme geben nur Nullfolgen, bei denen dann nach einer gewissen Anzahl, die der PLL bzw. dem Datenseparator maximal zuzumuten ist, von der Datenübermittlung auf die Taktfrequenz umgeschaltet wird.

Nun wird also nicht bei jedem Takt- und Datensignal ein Impuls an das Laufwerk geschickt, sondern immer nur dann, wenn ein Datenbit logisch 1 war. Erscheint eine zu lange Folge von Datenbits logisch 0, dann wird, wenn sich die Daten nicht ändern, nun auf die Ausgabe des Taktes übergegangen, bis wieder eine Datenänderung eintritt.

Diese gedächtnisbehafteten Codes, deren Entstehung fast ausschließlich mit sehr viel, mit Formeln gefülltem Papier verbunden ist, sind im Detail nur etwas für Signaltheoretiker. Hier soll es reichen, wenn im Endeffekt bei diesem Verfahren (MFM) bei einer doppelt so hohen Datenrate die Flußwechselfrequenz genau die gleiche ist wie bei FM. Ohne eine höhere Anforderung an das Diskettenmaterial kann also die doppelte Datenmenge darauf gespeichert werden.

Aufgrund des fehlenden Taktes existiert jedoch nun kein 'Grundton´ mehr, auf den der Schreib/Leseverstärker einschwingen könnte. Die Anforderungen besonders an die Phasentreue sind bei MFM, also doppelter Dichte, demnach höher. Und so kommt es, daß manche Laufwerke, die für einfache Dichte konstruiert sind, bei doppelter Dichte noch mitmachen, andere eben nicht. Dies ist hauptsächlich Material- und Abgleichtoleranzen zuzuschreiben.

Abwärtskompatibel sind jedoch alle Laufwerke. Ein für doppelte Dichte konstruiertes LW kann auch Disketten in einfacher Dichte verarbeiten, sprich lesen und schreiben.

FM und MFM-Codierung

Bei beiden Verfahren existieren sogenannte 'Bit-Zellen', in denen jeweils Datenbits übertragen werden. Die Bit-Zellen existieren jedoch nicht als magnetische Informationen auf der Diskette, sondern sind über Zeitbeziehungen definiert. Zwischen zwei Bit-Zellen treten nur Taktimpulse auf, in den Zellen können Datenbits auftreten, je nachdem, ob ein 1-Bit oder ein 0-Bit übertragen wurde.

Die einzelnen Datenbits werden aus dem im Datenregister abgelegten Wert, der in das DATA-SHIFT-REGISTER kopiert wird, durch Links-Schieben erzeugt. Das herausgeschobene Bit wird nach der Codierung an das Laufwerk gesendet. Beim Lesen wie beim Schreiben wird also mit dem höchtwertigen Bit begonnen.

FM - Single Density

Die Länge einer Bit-Zelle beträgt bei FM 4 Mikrosekunden. Daraus ergibt sich eine Wandlungszeit für 8 Bit von 64 Mikrosekunden. Dies ist die Zeit, welche der CPU zum Verarbeiten eines vom FDC gelieferten bzw. eines an den FDC zu liefernden Bytes bleibt. Diese Zeit kann im schlimmsten Fall (Taktfrequenz am oberen Limit, Drehzahlabweichung Schreiben zu Lesen maximal) auf 55 bzw. 47 Mikrosekunden beim Lesen bzw. Schreiben reduziert werden. Mit der nominalen Bit-Zelle ergibt sich eine Transfergeschwindigkeit der Daten von 250000 Bits pro Sekunde.

Am Anfang einer jeden Bit-Zelle wird ein Takt-Impuls (Clock) gesendet, innerhalb der Bit-Zelle, also zwischen 2 Clock-Impulsen, wird je nach Datenbit ein Datenimpuls (Bit = 1) oder keiner (Bit = 0) eingefügt.

MFM - Double Density

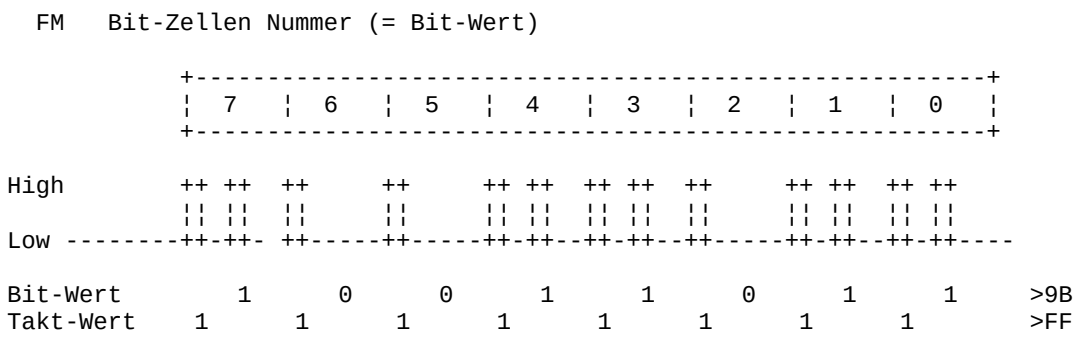
Die Länge einer Bit-Zelle beträgt hier 2 Mikrosekunden. Das entspricht einer Byte-Wandlungszeit von nominal 32 µs, die im schlimmsten Fall 27,5 bzw. 23,5 µs beim Lesen bzw. Schreiben betragen kann. Die Transfergeschwindigkeit liegt bei 500000 Bits pro Sekunde.

In MFM werden nur dann Datenimpulse in der Mitte einer Bit-Zelle geschrieben, wenn das entsprechende Datenbit logisch 1 war. Tritt ein einziges Null-Bit auf, so wird kein Datenimpuls gegeben. Folgen jedoch 2 oder mehr Null-Bits aufeinander, so wird nach dem ersten Null-Bit am Anfang einer Null-Daten Bit-Zelle ein Taktimpuls gesendet. Es wird also bei einer längeren Folge von Null-Bits von der Datenübertragung auf die Taktübertragung umgeschaltet. So ist gewährleistet, daß immer ein Referenzsignal vorhanden ist, mit des-

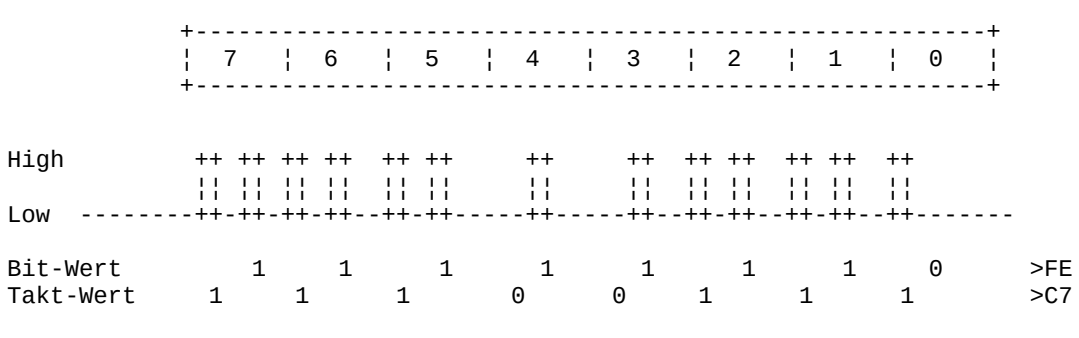
sen Hilfe sich der FDC auf die momentane Datenrate (bedingt durch Drehzahlschwankungen und unterschiedliche Taktfrequenzen bei Fremddisketten) einstellen kann.

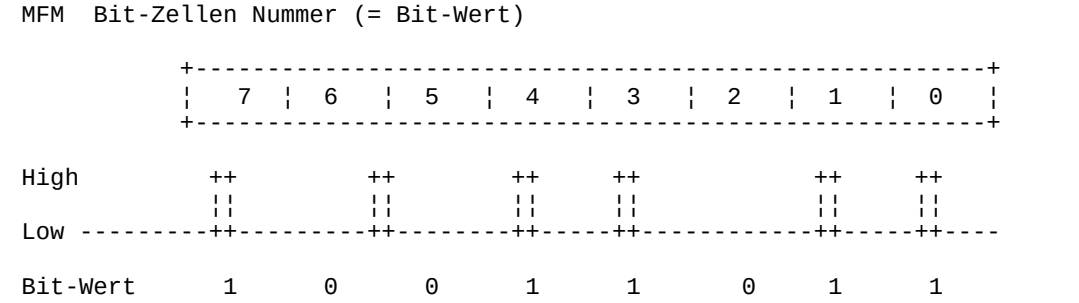
Beispiel

Es soll das Byte >9B in FM und MFM übertragen werden. Dabei ergeben sich folgende Diagramme der zum Laufwerk gesendeten Impulse:



Ein modifiziertes Signal wie z.B. die AF-Kennung besitzt die folgende Impulsstruktur:





Wie zuvor beschrieben, sind Impulse am Anfang einer Bit-Zelle immer Taktsignale, solche inmitten einer Bit-Zelle immer Datenbits.

Im Vergleich FM zu MFM muß beachtet werden, daß die Bit-Zellen bei MFM nur noch halb so groß (halb so lang) sind wie bei FM. Daraus ergibt sich, daß die maximale Impulsfrequenz dann auftritt, wenn eine kontinuierliche Folge gleicher Datenbits gesendet wird. Diese Frequenz entspricht der Frequenz, die eine dauernde 1-Folge in FM erzeugt, mit dem wesentlichen Unterschied, daß bei FM nur jeder zweite Impuls Daten repräsentiert, bei MFM aber jeder Impuls ein Datenbit darstellt. Dies zeigt direkt, daß das Double-Density-Verfahren (MFM) die doppelte Datenübertragungsgeschwindigkeit und die doppelte Speicherkapazität erzielt, ohne eine Erhöhung der maximal auftretenden Impulsfrequenz zu erfordern.

Spurstruktur

Nach all diesen theoretischen Überlegungen soll nun anhand einer eingehenden Besprechung der Spurstruktur die Umsetzung in die Praxis erläutert werden.

Die Spurstruktur einer Diskette wird beim Formatieren festgelegt. Schon dort muß die Einhaltung von Controllerspezifikationen wie minimale oder maximale Gap-Länge u.ä. sichergestellt werden.

Anhand einer Art Basic-Programm soll der verwendete Algorithmus erklärt werden:

```
100 REM Spurstruktur Einfache Dichte
110 Schreibe 22 mal >FF   REM Index-Gap, GAP 1 (CorComp: 12 mal >FF)
120 Schreibe 6 mal >00   REM Address-Gap
130 Schreibe 1 mal >FE   REM ID-Address-Mark (IDAM)
131 Schreibe 1 mal Spurnummer
132 Schreibe 1 mal Seitennummer
133 Schreibe 1 mal Sektornummer
134 Schreibe 1 mal >01   REM Sektorlänge 256 Bytes
135 Schreibe 1 mal >F7   REM Erzeugt 2 CRC-Bytes
140 Schreibe 11 mal >FF   REM ID-Gap, GAP 2
141 Schreibe 6 mal >00
150 Schreibe 1 mal >FB   REM DATA-Address Mark(DAM)
151 Schreibe 256 mal >E5   REM Leerdaten
152 Schreibe 1 mal >F7   REM Erzeugt 2 CRC-Bytes
153 Schreibe 45 mal >FF   REM Data-Gap, GAP 3
160 IF NOT(Alle Sektoren fertig) THEN GOTO 120
170 Schreibe so oft >FF bis Ende REM GAP 4
180 END
```

Der Vorgang des Schreibens einer Spur beginnt und endet mit dem vom Laufwerk gelieferten Index-Puls. Durch die unvermeidbaren Drehzahlschwankungen des Laufwerkmotors ist es daher um das Index-Loch (Bezugspunkt beim Formatieren) herum Essig mit einer guten Synchronisation auf den Datentakt. Daher ist der Post-Index-Gap (Zeile 110) unbedingt notwendig, um Zeit zur Synchronisation auf den ersten Sektor der Spur zu haben.

Die Anzahl in Zeilen 140 und 141 (Gap 2) muß exakt eingehalten werden, alle anderen Gaps können länger, sollten jedoch nicht kürzer werden, will man nicht Synchronisationsaussetzer mit den entsprechenden Wartezeiten haben. Gap 4 muß so groß sein, daß dem FDC mehr Bytes angeboten werden, als er bis zum nächsten Index-Puls schreiben kann. Durch die Änderung der Umdrehungsgeschwindigkeit (Regelschwankungen etc.) variiert die effektive Spurlänge um einige Bytes, so daß hier Reserven für die niedrigste zulässige Drehzahl vorgesehen werden müssen.

Insbesondere Gap 2 und 3 sind von großer Bedeutung beim Schreiben von Sektoren, da der Beginn eines neu geschriebenen Sektors aus einer festgelegten Zeitverzögerung nach dem Ende des korrespondierenden Adreßfelds bestimmt wird. Näheres in Kapitel 16. Da auch während des Schreibens die Drehzahl schwanken kann, ist die Position des Sektor - Endes nicht bzw. nicht exakt vorhersagbar. Es muß daher eine 'Auslaufzone' hinter dem Sektor bereitstehen, die belegt werden kann, ohne andere Sektoren bzw. deren Adreßfelder zu zerstören. Diese Auslaufzone stellt Gap 3 (Zeile 153) dar, dessen Länge mindestens 10 Bytes >FF und 4 Bytes >00 betragen muß.

Das Spurformat bei doppelter Dichte ist fast identisch, abgesehen von einigen Besonderheiten aufgrund der geforderten schnellen Synchronisation.

```
100 REM Spurstruktur Doppelte Dichte
110 Schreibe 32 mal >4E         REM Index-Gap, GAP 1 (Myarc: 50 mal >4E)
120 Schreibe 12 mal >00         REM Address-Gap
130 Schreibe 3 mal >F5          REM CRC-Start
131 Schreibe 1 mal >FE          REM IDAM
132 Schreibe 1 mal Spurnummer
133 Schreibe 1 mal Seitennummer
134 Schreibe 1 mal Sektornummer
135 Schreibe 1 mal >01          REM Sektorlänge 256 Bytes
136 Schreibe 1 mal >F7          REM Erzeugt 2 CRC-Bytes
140 Schreibe 22 mal >4E         REM ID-Gap, GAP 2
141 Schreibe 12 mal >00
150 Schreibe 3 mal >F5          REM CRC-Start
151 Schreibe 1 mal >FB          REM DAM
152 Schreibe 256 mal >E5        REM Leerdaten
153 Schreibe 1 mal >F7          REM Erzeugt 2 CRC-Bytes
154 Schreibe 28 mal >4E         REM Data-Gap, GAP 3
160 IF NOT(Alle Sektoren fertig) THEN GOTO 120
170 Schreibe so oft >4E bis Ende REM GAP 4
180 END
```

Sowohl in einfacher wie auch in doppelter Dichte werden Bytes benötigt, welche die Kennungen für Adreßfeld und Sektorstart aus dem allgemeinen Datenstrom hervorheben. Bei einfacher Dichte sind dies die Bytes

>FE und >FB (die anderen werden in Kapitel 14 behandelt). Beide Bytes werden als >FE bzw. >FB geschrieben, jedoch fehlen ihnen an bestimmten Stellen Taktimpulse. Durch vorangegangene Synchronisationsdaten ist gewährleistet, daß der FDC bereits bytesynchron ist und diese Codierungsfehler korrekt erkennt. Da es ziemlich unwahrscheinlich ist, daß genau diese Fehler zufällig auftreten, werden diese Kennungsbytes sehr zuverlässig erkannt.

Bei doppelter Dichte ist die Sachlage etwas anders. Hier dienen nicht die ID-Bytes >FB und >FE selbst als Kennung, sondern die ihnen vorangehenden Bytes >F5, die nicht als >F5 sondern als >A1 mit einer fehlenden Taktflanke zwischen Bit 4 und 5 geschrieben werden. Auch hier wird ein Codierfehler simuliert, der erst dann korrekt erkannt wird, wenn der FDC bereits bytesynchron war.

Disketten

Das eigentliche Objekt unserer Betrachtungen tritt erst jetzt in Erscheinung - die Floppy-Disk oder genauer der Datenträger. Doch ist es wie bei jedem Datenträgersystem mit dieser Zuverlässigkeit, daß sich das Hauptaugenmerk auf die Daten richtet und von der 100% korrekten Funktion von Trägermedium und Laufwerk ausgegangen wird.

Doch das ist nicht zwangsweise so. Man kann durchaus durch Wahl von Diskettenmaterial und Laufwerk mitentscheiden, wie hoch die Fehlerquote in der Programmsammlung, bewirkt durch Kopierfehler oder ähnliches wird.

Als Fachausdrücke werden für die, die Diskette umgebende feste Hülle das Wort 'Jacket' benutzt, der eigentliche, beschichtete Datenträger wird mit 'Cookie' bezeichnet.

Bei der Wahl der geeigneten Disketten für das eigene System muß beachtet werden, daß beim TI prinzipiell 2 verschiedene Laufwerktypen zum Einsatz kommen: solche mit 40 Spuren und solche mit 80 Spuren. Hierbei ist die Seiten- bzw. Kopfanzahl ohne Belang bzw. muß im konstruktiven Zusammenhang beachtet werden. Exotische Laufwerke wie Apple-Restposten mit 35 Spuren sind nicht von Bedeutung, jedoch können sie zu den 40-Spur Laufwerken gezählt werden, da hier im Allgemeinen nur ein nachträglich montiertes Anschlagblech nicht mehr als 35 Spuren zulässt. Diese Laufwerke sind jedoch halbspurfähig, weshalb diese bei Anschlagausbau und entsprechender Modifikation der Stepper-Elektronik auch auf 80 Spuren 'aufgebohrt' werden können.

Daß und warum jedoch 80 Spur Laufwerke weniger für den Diskettentausch geeignet sind, soll weiter unten erklärt werden.

Um die folgenden Passagen wirklich zu verstehen, muß an dieser Stelle noch einiges nachgetragen werden, was thematisch zwar zu den Aufzeichnungsverfahren gehört, jedoch auch bei der Qualitätsbetrachtung wichtig ist, nämlich das Zusammenspiel zwischen Diskettenoberfläche und Schreib-/Lesekopf.

Bei einseitigen Diskettenlaufwerken befindet sich der Kopf dort, wo nach Einlegen der Diskette in das Laufwerk die Unterseite des Cookies liegt. Ihm gegenüber liegt, in Ermangelung eines zweiten Kopfes, ein sog. Andruckfilz der dafür sorgt, daß das Diskettenmaterial immer so fest am Kopf anliegt, daß eine einwandfreie magnetische Kopplung zwischen Cookie und Kopf gegeben ist. Dabei ist der Kopf feststehend montiert und der Filz an einem beweglichen Arm befestigt, der mittels des sog. Head-Load-Magneten von dem Cookie abgehoben bzw. auf dieses abgesenkt werden kann. Bei zweiseitigen Diskettenlaufwerken befindet sich an der Stelle des Andruckfilzes ein zweiter Schreib-/Lesekopf, der wie der Andruckfilz beweglich an einem kurzen Arm montiert ist.

Beim Öffnen der Verriegelung eines Floppy-Disk-Laufwerkes wird der Andruckfilz bzw. der zweite Kopf angehoben, um beim Herausziehen der Disketten diese nicht zu beschädigen bzw. nicht selbst beschädigt zu werden. Dies kann im Betrieb mittels der Head-Load-Magnete auch dann geschehen, wenn das Laufwerk längere Zeit nicht angesprochen wurde. Doch dazu mehr im folgenden Abschnitt.

Allgemeine Qualitätsmerkmale

Nach dem zuvor gesagten kann man also Diskettenlaufwerke in verschiedene Kategorien einteilen: solche mit nur einem Schreib-/Lesekopf, solche mit zweien, Laufwerke mit 40 Spuren und solche mit 80 Spuren und Laufwerke mit Head-Load-Magnet und solche ohne. Doch bei allen ist eines gleich: sind die Laufwerke angesprochen, so haben Cookie und Kopf immer einen sehr engen Kontakt, da die Köpfe nicht über der Cookie-Oberfläche fliegen, wie bei Harddisks oder großen Plattenlaufwerken, sondern richtig fest angedrückt werden. Dies stellt hohe Anforderungen an die Oberflächengüte von Disketten, die visuell nur unzureichend bewertet werden kann.

Aufgrund des so verursachten Abriebs geben Diskettenhersteller an, wie oft auf eine Spur zugegriffen werden kann bis deren Magnetschicht so beschädigt ist, daß die Funktion gefährdet wird. Praktisch sinkt dabei der Ausgangspegel der Magnetschicht (Remanenz oder MOL, Maximum Output Level) unter die geforderten 70% des Aufsprechpegels - die Daten drohen im Rauschen zu verschwinden.

Aus diesem Grund werden Disketten, nachdem sie mit der magnetisierbaren Schicht versehen wurden, in der späteren Aufzeichnungszone geschliffen, was die Rauhtiefe reduziert und die Oberflächenhärte erhöht. Hier hat man bereits das erste Qualitätsmerkmal: eine homogene, leicht glänzende Oberfläche.

Disketten, deren Cookie eine matte oder inhomogene Oberfläche hat, sollten nicht zur laufenden Benutzung oder besser gar nicht verwendet werden. Natürlich ist dies nur ein grober Anhaltspunkt und nur geeignet, eine erste Entscheidung zu erlauben. Doch es nutzt sich ja nicht nur die Diskette durch Reibung ab, auch der Kopf bekommt seinen Teil ab.

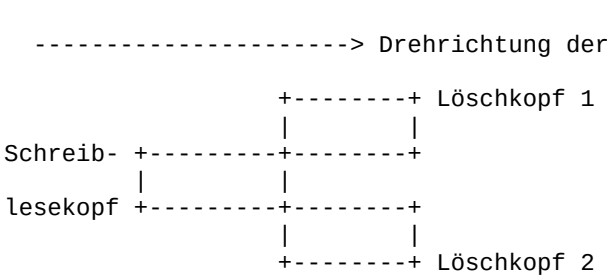
Neben der reinen Oberflächengüte ist der Rundlauf, speziell der Seitenschlag einer Diskette wichtiges Qualitätsmerkmal. Besonders bei zweiseitigen Laufwerken fällt ein verzogenes Cookie durch Schleifgeräusche auf, die nicht toleriert werden sollten. Dies merkt man leider aber erst dann, wenn die Diskette im Laufwerk liegt und rotiert. Diese Diskette sollte verschenkt werden oder zur Tauschdiskette degradiert werden. Tritt dies bei Disketten einer Firma öfters auf, so sollte man deren Disketten nicht mehr kaufen.

Ein Qualitätsmerkmal, das insbesondere für den Einsatz auf 80-Spur-Laufwerken maßgeblich ist, stellt die maximale Speicherdichte einer Diskette dar, angegeben in 'tracks per inch' oder als übliches Kürzel 'tpi'.

Hierzu ein kurzer Blick auf den Schreib-/Lesekopf eines Disklaufwerkes. Dieser besteht genau betrachtet aus 3 Systemen: dem eigentlichen Schreib-/Lesesystem und zwei jeweils rechts und links dahinter angeordneten Löschköpfen, den sogenannten 'Tunnel Erase Heads'. Beim Schreiben von Daten wird nun das Nutzsignal über den Schreib-Lesekopf, der nun Schreibkopf ist, als Magnetisierung (Flußwechsel) in die Diskettenoberfläche eingebracht. Sodann läuft die magnetisierte Zone an den Löschköpfen vorbei, welche die Ränder der Datenspur weglöschen.

Damit wird die Datenspur von den Nachbarspuren getrennt und so ein Spur-Übersprechen verhindert. Man verschenkt also Platz auf der Diskette, um die Datensicherheit zu erhöhen. Erinnern wir uns: Bei der Formatierung haben wir ähnliches kennengelernt.

Eine kleine Graphik soll dies erläutern:



Die Zahl vor dem Kürzel 'tpi' gibt (eher unzureichend) an, wie fein strukturiert die magnetisierbare Oberfläche der Diskette ist und wie schmal demzufolge eine Datenspur minimal werden kann, ohne daß die Daten 'in der Materialkörnung verloren gehen' (strenggenommen ist die Oberfläche kristallin, aber das ist hier von geringer Bedeutung).

Dabei ergibt sich aufgrund der Abmessungen eine Spurdichte von 1,9 Spuren pro Millimeter bei 48 tpi und eine Dichte von 3,6 Spuren pro Millimeter bei 96 tpi. Dieser Wert für die radiale 'Auflösung' der Diskettenoberfläche ist also abhängig von der Laufwerksmechanik. Für die axiale Auflösung spielt dies keine Rolle, da hier die Anzahl der Flußwechsel pro Zeit- bzw. aufgrund der festen Rotationsgeschwindigkeit (einer Spur) pro Radiant wesentlich ist.

Natürlich beeinflussen sich beide Parameter, da es recht aufwendig wäre, ein anisotropes Material zu verwenden, nur um unterschiedlichen Datendichten gerecht zu werden.

Wie bei den Formatierungsarten indirekt gezeigt, unterscheidet sich die Anzahl der Flußwechsel innerhalb einer Spur nicht wesentlich, wenn statt FM nun MFM benutzt wird - die FM-Flußwechsel übersteigen im Mittel eher die der MFM. Dies ist unabhängig davon, ob mit 40 oder 80-Spur-Laufwerken gearbeitet wird.

Da jedoch die Spurbreite bei 80 Spur Laufwerken nur halb so groß ist wie bei 40 Spur Laufwerken, erfordern 80-Spur-Laufwerke solche Disketten mit 96 tpi, 40 Spur Systeme nur solche mit 48 tpi. Das ist (hinkend) zu vergleichen mit der Qualität von Billigst-Eisenoxid Compact Cassetten und z.B. Reineisencassetten oder Video-Cassetten.

Durch die geringere Spurbreite bei 80 Spur Systemen muß u.a. der Aufsprechstrom reduziert werden, um ein Seiten- bzw. Spurübersprechen zu verhindern. Durch die engere Fokussierung des S-/L-Kopfes wird zudem eine tiefere Magnetisierung erreicht. Je nach Laufwerk kann dies dazu führen, daß auch 48 tpi Disketten auf 80 Spur Laufwerken verwendet werden können, sowohl bei einfacher wie doppelter Dichte. Diese Sicherheitsreserven weisen aber im Allgemeinen nur Markendisketten auf.

Ein wesentlich den Preis einer Diskette bestimmender Faktor ist die Angabe EINSEITIG oder ZWEISEITIG (doppelte Speicherdichte kann immer als gegeben angenommen werden). Disketten, die ausdrücklich nur für einfache Dichte gedacht sind, sollte man lassen, wo sie sind - dies sind nur bessere Schleifscheiben.

Ein- oder Zweiseitig gibt bei Markenware nur an, ob die Disketten auf einer oder allen beiden Seiten geprüft wurden. Dies ist ähnlich der Fertigung von Transistoren und ihrer Einstufung in Stromverstärkungsgruppen: je nachdem, ob die Verstärkung reicht oder nicht, wird der Stempel entsprechend aufgedrückt. Genauso bei Disketten. Bestehen Sie die Prüfung auf beiden Seiten, werden sie als zweiseitige Typen (Kennung 2D oder DS/DD) verkauft. Fallen sie bei der Prüfung der zweiten Seite aus, bzw. werden sie hier erst gar nicht getestet (spart immerhin Zeit!), so werden sie als einseitige Disketten angeboten (1D oder SS/DD). Dabei ist jedoch verschiedenes zu beachten.

So hängt es vom Prüfverfahren des jeweiligen Herstellers ab, ob die ausgesonderten Disketten (einseitig bzw. solche, die als weiße Ware auf Umwegen wieder in den Markt kommen) tatsächlich fehlerhaft arbeiten oder ob man den Vermerk '100% geprüft' besser gleich vergißt. Manche Hersteller prüfen die ganze Oberfläche jeder Seite, erkennen also Fehler zwischen den Spuren. Das soll sicherstellen, daß auch fehljustierte Laufwerke mit diesen Disketten arbeiten. Solche Disketten arbeiten dann in der Regel auch bei 80 Spur Systemen, wenn sie nicht gleich als 96tpi-Typen verkauft werden.

Andere Prüfverfahren setzen (fast) normale Laufwerke ein, wo geschrieben, zurückgelesen und hinterher gelöscht wird. Diese billigen und schnellen Verfahren sind naturgemäß weniger zuverlässig. Die Laufwerke werden übrigens so ausgestattet, daß sie nicht nur die Daten, sondern auch deren Pegel ermitteln können. Gelöscht wird übrigens (wenn nach ECMA/DIN verfahren wird) mit einem statischen Magnetfeld.

Generell kann man aber davon ausgehen, daß einseitige und doppelseitige Disketten eines Typs auf ein und derselben Fertigungsstraße hergestellt werden und man durch eigenen Disktest (DM-2 Modul oder Formatieren mit Verify) den Test der zweiten Seite nachholt und feststellt, ob es klappt oder nicht. Der vielfach benutzte Allgemeinplatz 'Es mag ja ein paar Mal gehen, aber man darf sich über plötzlichen Datenverlust nicht wundern.' ist daher fern ab der Realität bei der Herstellung von Floppy-Disks.

Technische Qualitätsnormen

Mit der großtechnischen Herstellung von Disketten und den daraus resultierenden Anforderungen an die Einhaltung bestimmter Normen bezüglich Oberflächengüte, Maßhaltigkeit, Parameterbeständigkeit etc. ergibt sich zwangsläufig die Notwendigkeit, bestimmte Normen festzulegen. Ziel dieser Normen ist dabei nicht primär, die Vergleichbarkeit der Produkte unterschiedlicher Hersteller zu sichern, sondern die Gewährleistung der Austauschbarkeit des Produktes durch einen Mindeststandard.

Bei Disketten betrifft dies die rein mechanischen Eigenschaften (Maße, Materialien, Abrieb etc.) sowie die magnetischen Eigenschaften des Datenträgers (Koerzitivität, Remanenzflußdichte, Temperaturabhängigkeit der gespeicherten Daten etc.). Daneben werden sog. physikalische Eigenschaften vorgegeben (wer auch immer diese Unterscheidung vorgab war sich wohl nicht im Klaren, daß auch die anderen beiden Eigenschaften zur Physik gehören...).

Diese Standards setzen die ECMA (European Computer Manufacturers Association) für Europa sowie international die ISO (International Organisation for Standardization). Auch das DIN (Deutsches Institut für Normung) hat, wie könnte es anders sein, hier seine Normen festgelegt. Die PTB (Physikalisch Technische Bundesanstalt) liefert Meßdisketten bzw. Meßparameter für Prüfanlagen.

Festgelegt werden z.B. die Normen für 5¼"-Disketten mit 40 Spuren in der Vorschrift ECMA-70. Die DIN-Norm (erschlägt alle Diskettenstandards) hat die Nummer DIN 66247. Und da es keinen Sinn ergibt, europäische Standards als Insellösung zu etablieren, entsprechen diese Normen in allen wesentlichen Punkten den entsprechenden ISO-Vorgaben (ISO 6596, 7487, 8630 und 8378).

Für den interessierten Leser hier ein Überblick über die genormten Merkmale, die eine Diskette aufweisen muß. Ein Auflistung der exakten Zahlenwerte und Prüfschemata würde den Rahmen für dieses Thema sprengen.

Die Lagerung von Disketten sollte möglichst bei einer Raumtemperatur von 23°C (maximal 26°C) und einer Luftfeuchtigkeit von ca. 50% (maximal 80%) geschehen. Dabei darf ein magnetisches Feld von maximal 4 kA/m wirken, andernfalls die Daten (erlaubterweise) in Mitleidenschaft gezogen werden. Dabei sollte beachtet werden, daß der normale Schreibstrom einer Feldstärke von etwa 24 kA/m entspricht, weshalb es bei der Lagerung auch etwas weniger sein darf (Disketten weg vom Monitor!).

Der Schreibstrom von 24 kA/m bei üblichen DD-Disketten ist übrigens der Grund, wieso die an sich höherwertigeren High-Density Disketten eines PC der AT-Klasse auf den gängigen Laufwerken am TI nicht zufriedenstellend arbeiten. Diese Disketten benötigen einen Schreibstrom von etwa 50 kA/m, weshalb sie nach Formatierung mit 24 kA/m einen völlig unzureichenden Wiedergabepegel (MOL) aufweisen.

Neben so selbstverständlichen Anforderungen wie der Einhaltung aller Maße (Jacket-Höhe, -Breite, -Dicke, Cookie-Durchmesser, Zentrierungslochdurchmesser, innere und äußere Begrenzung der Datenzone), wer-

den genaue Vorgaben zur erlaubten Krümmung/Wölbung des Jackets (diese beeinflussen die Jacket-Dicke) und zu den statischen und dynamischen Eigenschaften der magnetisierbaren Schicht auf beiden Seiten des Cookies gemacht. Die prinzipielle Überlegenheit des Kunststoffgehäuses von 3½"-Disketten wird hierbei übrigens nicht immer ausgespielt!

Genormt sind die Bereiche, in denen sich die zur Beschleunigung und konstanter Rotation des Cookies benötigten Drehmomente bewegen, der Durchmesser bzw. die Lichtdurchlässigkeit des Index-Loches zusammen mit der Undurchlässigkeit des Cookie-Materials für das Licht der zur Index-Erkennung benutzten Infrarot-LED's (Intensität am Indexloch dient als Bezugswert), die Pegelbereiche, innerhalb derer Signaleinbrüche bzw. Reste an sich gelöschter Signale erlaubt sind, der Wiedergabepegel einer einmalig formatierten Spur nach 3 Millionen Umdrehungen bei geladenem Kopf sowie nach 24-stündiger Lagerung unter den oben genannten Grenzwerten für Temperatur und Luftfeuchte.

Kopierschutztechniken und was man dagegen tun kann

Dieses, zugegeben etwas heikle Thema ist fast so alt wie die Datenspeicherung auf der Diskette.

Kopierschutztechniken wurden entwickelt, um das unberechtigte Kopieren von Programmen oder Daten zu unterbinden und so eine Kalkulation über Aufwand und Gewinn aus einem Programm oder anderen auf Datenträger gespeichertem Material erst zu ermöglichen. Der Kopierschutz ist also aus Sicht dessen, der ihn installiert, eine vernünftige, ja notwendige Sache.

Anders aus der Sicht des Anwenders dieser Daten oder Programme. Er ist in einem solchen Fall auf die Originaldiskette angewiesen, die, wie im letzten Kapitel besprochen, einem prinzipiellen Verschleiß unterliegt. Er muß daher ständig mit dem Ausfall dieser Diskette rechnen, insbesondere bei häufigem Gebrauch des Programms. Bedenkt man dabei noch, daß Rechner und Programme Betriebsmittel sind, deren Ausfall hohe Kosten verursachen kann, so sieht man damit die andere Seite eines Kopierschutzes, wenn keine Kopie verfügbar ist, die sofort einsetzbar ist. Daher bieten verschiedene Hersteller Sicherheitskopien ihrer Programme an, die beim Erwerb des eigentlichen Programms günstig mit verkauft werden.

Für einen TI-Besitzer sieht die Sache etwas anders aus: hier muß ein defektes Originalprogramm auf langen Wegen wieder beschafft werden, was, Betriebsmittel oder nicht, immer lästig ist. Kurzum: der Kopierschutz muß weg oder ein entsprechendes Programm muß her!

Wie ein solches Programm arbeiten kann, sei im folgenden beschrieben.

Dabei wird bei den elementaren Tricks begonnen und bei der eingehenden Manipulation von Spurinhalten und Adreßfeldern aufgehört. Voraussetzung für das Verständnis der folgenden Seiten ist jedoch die eingehende Kenntnis von Spurstruktur und Dateninterpretation. Dazu wird das Programm COPY-C des Verfassers als Experimentalwerkzeug empfohlen.

Es werden aber nur Themen behandelt, bei denen auf irgendeine Weise das Kopieren von Disketten unterbunden wird. Listschutz und andere Trivialitäten sind nicht Gegenstand der Untersuchungen, ebenso wie die Frage, wie ein Schutz aus dem Programm selbst entfernt werden kann, nicht gestellt oder beantwortet wird.

Alle Betrachtungen gelten, sofern nicht anders vermerkt, im Prinzip für alle auf dem TI verfügbaren Diskcontroller, auch MYARC und Percom. Ebenso ist die Speicherdichte prinzipiell ohne Belang, von der größeren Speicherkapazität einmal abgesehen.

Dabei können die Schutzmethoden in drei Kategorien eingeteilt werden:

- Methoden, bei denen der Schutz mittels des normalen DOS der Diskcontrollerkarte erzeugt und nachher geprüft wird (Nut zung der Level-1-Routinen).
- Methoden, bei denen der Schutz mittels eines speziellen Programms erzeugt, später jedoch mit dem Standard-DOS geprüft wird.
- Verfahren, die bei Erzeugung und Test auf eigene Level-0 oder Level-1-Routinen angewiesen sind.

Zusätzlich muß noch beachtet werden, ob geschützte Zonen der Diskette nur eine Kennung auf Raubkopien sind (prinzipiell also Flags, z.B. fast alle Quality-99 Software) oder ob sich in diesen Zonen programmrelevante Daten befinden (etwa TP'99 oder die Millers-Graphics Tools). Eine abweichende Methode ist die, daß der Loader das Hauptprogramm patcht und diese Patches im Programm gelegentlich abgefragt oder als feste Pointer verwendet werden (z.B. der GPL-Assembler, bei dem der Aufwand für den verschlüsselten Loader in keinem Verhältnis zur lachhaften Abfrage der o.a. Patches steht).

Im Vorgriff auf die folgenden Passagen soll hier kurz beschrieben werden, was unter einem sog. Standard-DOS zu verstehen ist, welche Beschränkungen diesem auferlegt sind und worin ein sog. eigenes DOS von den Standard-Spezifikationen abweicht.

Wie aus den vorangegangenen Kapiteln bekannt ist, lassen sich die für den TI verwendbaren Diskcontroller auf zwei verschiedenen Softwareebenen ansprechen.

Auf dem höchsten Level geschieht dies mit dem Standard-PAB und dem DSRLNK mit DATA 8. Hierbei sind sowohl einfache als auch komplexe File-Operationen möglich, wobei die Diskette nur in logische Verbunde mit der Bezeichnung 'File' aufgeteilt wird.

Das nächste Niveau sind die Level-1-Routinen mit einem reduzierten VDP-PAB und dem DSRLNK mit DATA 10. Hierbei tritt die Sektorstruktur der Diskette in Erscheinung, besonders bei den Routinen >10, >11 und in gewisser Weise auch bei >14, >15.

Die physikalische Einheit 'Sektor' ist ein beim TI-DOS immer 256 Byte langer Block von Daten, unabhängig von der gewählten Speicherdichte und seiner Position auf der Diskette. Ein solcher Sektor ist durch seine laufende Nummer immer exakt definiert d.h. er kann allein durch seine Nummer gefunden werden. Dabei werden die Sektoren innerhalb einer Spur immer von 0 bis 8 bei einfacher Dichte und von 0 bis 17 bei doppelter Dichte durchnumeriert. Die Entscheidung, ob der korrekte Sektor in der korrekten Spur gefunden wurde, ermöglichen Spur- und Seitennummer im Adressfeld des betreffenden Sektors.

Um nun auf einen Sektor zugreifen zu können, wird dem Standard-DOS die Sektornummer übergeben. Nachdem die Speicherdichte ermittelt wurde, wird diese Sektornummer durch die Anzahl der Sektoren pro Spur dividiert, wodurch sich im ganzzahligen Anteil die Spurnummer und im Rest die relative Sektornummer in dieser Spur ergibt. Überschreitet die Spurnummer dabei die festgelegte maximale Anzahl von Spuren, so wird diese Zahl am Zahlenkreis gespiegelt und ergibt so die Spurnummer auf der zweiten Diskettenseite.

Anschließend wird diese Spur mit dem Kopf des Laufwerkes angefahren. Nachdem diese Spur (theoretisch) erreicht ist, wird anhand eines beliebigen Adressfeldes geprüft, ob dies die korrekte Spur ist, andernfalls wird die Spur erneut angefahren. Dies ist möglich, da alle Adreßfelder einer Standard-Spur gleiche Spurnummern haben. Danach wird dem Floppy-Disk-Steuerchip der Befehl gegeben, einen Sektor zu lesen bzw. zu schreiben, dessen Adressfeldparameter wie Seite, Spurnummer und Sektornummer zuvor berechnet und an den Steuerchip übergeben wurden. Beim Lesen von Sektoren wird nach Erkennen des Adressfeldes auf die Sektorstartmarkierung gewartet, beim Schreiben wird mit einer Zeitverzögerung gearbeitet.

Man erkennt aus all dem, daß beim Standard-DOS viel gerechnet und wenig gedacht wird - schließlich wurde für die korrekte Spurstruktur bereits beim Formatieren gesorgt.

Es ist aber auch zu sehen, daß ein eigenes DOS sich an keine dieser Konventionen zu halten braucht, wenn es erst einmal aus normalen Sektoren geladen wurde. So ist es z.B. denkbar, einen Sektor mit einer zu hohen Sektornummer zu versehen, der von dem Standard-DOS nicht erkennbar ist. Dieses Thema wird jedoch weiter unten, quasi 'vor Ort' genauer behandelt.

Ein eigenes DOS, das ist im einfachsten Fall ein abgeschriebenes und gekonnt manipuliertes Stück aus einem ROM-Listing eines Diskcontrollers.

Schutzmethoden mit dem Standard-DOS

Diese Schutzmethoden zeichnen sich dadurch aus, daß sie auch ohne größeren Aufwand realisierbar sind. In der Regel ist jedoch auch die damit verbundene Schutzwirkung ohne größeren Aufwand zu eliminieren...

Flaggesteuerter Kopierschutz

Hier sei an erster Stelle das Protection-Byte in Sektor 0 einer TI-Diskette erwähnt, siehe auch Kapitel 7. Sofern das Programm nicht selbst dieses Byte prüft (z.B. ID-DATA oder -KONTO), kann es durch ein Leerzeichen ersetzt werden (mit jedem beliebigen Sektoreditor), woraufhin diese Diskette auch mit dem DM-2 Modul kopierbar ist. Ansonsten lassen sich solche Disketten mit jedem Sektorkopierer kopieren - der Schutz wird in diesem Fall mit kopiert.

Alle flaggesteuerten Schutzmethoden haben einen Nachteil: sie greifen nur, wenn das Kopierprogramm bzw. die File-Verwaltung dieses Flag auch prüft. So verhält es sich auch mit dem Schreibschutz eines Files. Es kann nur auf höchster Ebene (Filezugriff) geschützt werden; vor einem Sektorzugriff ist es nicht schreibgeschützt. Eine derart manipulierte Diskette ist jedoch als einfache Sektorkopie (Programm unter Verwendung z.B. der DOS-Routine >10) lauffähig.

Sektormanipulation

Sektoren, die über das normale DOS les- bzw. schreibbar sein sollen, müssen die normalen Sektorkennungen besitzen. Damit sind sie normal mit einem Sektorkopierer kopierbar. Der einzige Weg, eine Kennung in einen Standard-Sektor zu bringen, die nicht über Standard-Sektorkopie kopiert werden kann, ist, diesen Sektor zu zerstören. Das kann so geschehen, daß ein Programm mittels DOS-Routine >10 immer Daten auf ein und denselben Sektor schreibt und der 'Programmschützer' mitten drin das Laufwerk öffnet. Der Sektor ist dann mit ziemlicher Sicherheit defekt und jeder Standard-Zugriff auf diesen Sektor erzeugt einen Lesefehler. Dies kann abgefragt und im geschützten Programm ausgewertet werden (Beispiel COPY-A von R.Frommer). Kopieren lässt sich ein solcher Sektor auch mit einem Standard-Sektorkopierer, bei dem nach dem normalen Lauf, der ja alle Sektoren korrekt schreibt, ein eigenes Programm mittels Routine >10 diese Sektoren mit ihren gelesenen Daten immer wieder schreibt und der 'Programmwilderer' auch im richtigen Moment die Klappe öffnet.

So lassen sich nur Disketten kopieren, bei denen der Inhalt des manipulierten Sektors nicht ausgewertet wird. Ist dies jedoch der Fall, so müssen andere Verfahren angewandt werden.

Ein Sektor kann auch als defekt ausgewiesen werden, wenn er gar nicht existiert. So sind z.B. MACROWORD plus und EASYDATA 99 so geschützt, indem die Diskette nicht komplett formatiert wurde und so einige Spuren keine Sektoren enthalten. Der Zugriff auf sie erzeugt nur Fehlermeldungen. Derartige Disketten lassen sich jedoch mit partiell arbeitenden Sektorkopierern wie Nibbler auf eine fabrikneue Diskette kopieren, sofern kein Myarc-Diskcontroller verwendet wird, der ja bekanntlich nur 40 Spuren formatieren läßt (da COPY-C auf dieser Karte nicht arbeitet, ist sowieso alles zu spät...).

Doppelte Sektoren

Diese Sektoren können nur über eine eigene Formatierroutine erzeugt werden. Bekannter Vertreter dieser Gattung ist der SPAD-XIII Flugsimulator, der diese Besonderheit als Flag auswertet, ob die Diskette kopiert wurde oder eine Originaldiskette vorliegt.

Beim Formatieren werden zweimal die identischen Adreßfelder zusammen mit den korrekten Sektorinhalten geschrieben. Versieht man diese Sektoren mit verschiedenen Daten, so sieht es aus, als wäre der Sektor defekt und lieferte laufend verschiedene Daten. Ein Schreiben dieses Sektors über das Standard-DOS ist ein nicht kalkulierbares Glücksspiel, da auch der Verify des DOS scheinbar schiefgeht.

Das Programm liest nun diesen Sektor zweimal kurz nacheinander und wertet die unterschiedlichen Daten aus. Da bei diesem Verfahren ein Sektor nun den Platz von zweien einnimmt, muß ein anderer Sektor dafür entfallen. Damit ist dann wieder ein Flag verfügbar, nämlich das Fehlen eines anderen Sektors.

Sektoren mit definierten Fehlern

Hierzu muß einleitend bemerkt werden, daß Sektoren mit Datenfehlern (CRC-Fehler) vom Standard-DOS gelesen werden können. Da ein solcher Fehlertest erst NACH dem Lesen aller 256 Bytes vorgenommen wird, befinden sich die Daten bereits in dem spezifizierten Puffer. Die Information des Fehlerbytes ist dann gewissermaßen eine Zugabe. Schrittfehler oder nicht erfolgte Sektoridentifikation übergeben generell keine Daten in den Puffer bzw. liefern nur ein festes Bitmuster,

im Allgemeinen >00.

Spuren, bei denen ein bestimmter Sektor eine falsche Spurkennung aufweist, werden zwar korrekt angefahren, jedoch wird dieser Sektor nicht erkannt und ein Record-Not-Found-Fehler (RNF-Error) gemeldet. Haben alle Adreßfelder einer Spur ein falsches Spurbyte, so wird ein SuchFehler (Seek-Error) ausgegeben. Hierbei wurde zwar die Spur möglicherweise korrekt angefahren, doch simulierten die falschen Spurnummern einen Fehlzugriff. Mithilfe der Fehlercodes (siehe Kapitel 11) kann nach Zugriff auf diese Sektoren über das Standard-DOS die Fehlerart ermittelt werden. Kopierprogramme müssen diese Eigenschaften einer manipulierten Spur erkennen und korrekt behandeln.

Spurstrukturen 'Hausmacher-Art'

Damit begeben wir uns auf die Spielwiese dessen, was uns ein eigenes, im Idealfall vollkommen flexibles DOS ermöglicht: Das freie Spielen mit Adreßfeldern. Dies ist bereits die Stufe der Diskettenmanipulation, auf der die meistens Kopierschutzverfahren angesiedelt sind. Das Grundprinzip ist immer folgendes:

Eine bestimmte Anzahl von Spuren und Sektoren genügen den Anforderungen des normalen DOS, da aus ihnen ja schließlich der Loader geholt werden muß, der einen Zugriff mittels eines eigenen DOS' erst erlaubt.

Anschließend wird das eigentliche Programm mittels der nun geladenen Software installiert. Beispiele sind XB-Detective und alle 3 Millers Graphics Utilitites. Turbo-Pasc'99 und TI-Artist weisen jeweils nur einige wenige (programmrelevante) Sektoren mit verfälschten Kennungsbytes auf und besitzen kein ausgesprochen eigenes DOS.

Auf dem Niveau der Spurstruktur, bei der allein aus den Adreßfeldern die korrekte Reproduktion der Spur erfolgen kann, arbeiten alle Kopierprogramme, die im folgenden als 'Adressfeld-Scanner' bezeichnet werden sollen.

Das Adressfeldproblem

Für einen einfachen Adressfeld-Scanner wie Backup 1.1 oder Trackhack ist es in gewissem Sinn 'lebenswichtig', alle Adreßfelder einer Spur ab dem Indexloch in der korrekten Reihenfolge zu lesen. Da bei dem Floppy-Disk-Controller-Chip (kurz FDC genannt) jedoch während des Lesens von Adreßfeldern keine Möglichkeit gegeben ist, den Index-Puls der Floppy-Station zu testen, kann zwar der Anfang einer Spur mittels eines Pseudo-Seek-Befehls erkannt werden, jedoch nicht deren Ende. Adressfeld-Scanner müssen also immer davon ausgehen, daß sich in einer Spur nur eine bestimmte maximale Anzahl von Adreßfeldern befinden kann, ein paar mehr lesen und dann nach einer Wiederholung des ersten Adressfeldes suchen. Daraus bestimmen diese Programme die Anzahl der Sektoren pro Spur.

Durch das Lesen aller Adreßfelder werden jedoch die Bedingungen der vorangegangenen Abschnitte erfüllt. Doppelte Sektoren oder nicht existente Sektoren können jedoch sowohl die Berechnung der Sektoranzahl als auch den (notwendigen) Fluß beim Lesen der Sektoren stören. Zudem erheben einfache Adressfeldscanner wie die oben genannten Programme die gelesenen Adreßfelder zum Maß aller Dinge und versagen spätestens dann, wenn Disketten in einfacher Dichte auf Systemen mit Controllern für beide Dichten kopiert werden sollen (z.B. Track-Hack).

Adreßfelder mit CRC-Fehlern

Genau wie ein Sektor wird auch ein Adressfeld über eine Prüfsumme gegen Fehler abgesichert. Besonders dann, wenn ein Double-Density-FDC eine Single-Density-Diskette liest, werden Sequenzen als Adreßfelder erkannt, die keine sind. In der Regel weisen diese Adreßfelder (ab jetzt AF genannt) dann CRC-Fehler auf. Ein solches AF ist auf unserem momentanen Niveau der Spuranalyse ohne Bedeutung, da ein zu einem Adressfeld mit CRC-Fehler gehörender Sektor in keinem Fall über einen Sektor-Lesebefehl erreichbar ist. Wurde ein solches AF gelesen, so ist es aus der Liste zu streichen.

Auswertung der Adreßfelder

Wurden alle AF gelesen, alle fehlerhaften entfernt und die Anzahl der Sektoren in der momentan zu bearbeitenden Spur ermittelt, so kann mit der Reproduktion der Spur, d.h. der Formatierung der Kopie begonnen werden. Dies jedoch nur, wenn keine weitere Prüfung auf Spurlängenüberschreitung, nicht existente Sektoren oder ähnliches vorgenommen werden soll. Für viele Kopierschutzmethoden beim TI ist dies auch nicht notwendig, soll aber wenigstens kurz angesprochen werden.

Singuläre Adreßfelder

Singuläre AF sind solche AF, zu denen kein Sektor gehört. Diese AF täuschen gewissermaßen die Existenz eines Sektors vor. Da sich direkt an ein solches AF ein weiteres AF anschließen kann ist es möglich, in einer Spur ca. 50 AF unterzubringen, wovon keines einen Sektor 'im Schlepptau' hat (Beispiel GPL-Assembler).

Ein Kopierprogramm, welches auf eine Diskette trifft, die innerhalb einer Spur sowohl mit doppelten Sektoren als auch mit nicht existenten Sektoren geschützt ist, wird in der Regel nicht in der Lage sein, die doppelten Sektoren in der korrekten Sequenz zu lesen.

Es ist daher unbedingt nötig, nach dem Lesen aller AF zuerst die von ihnen angekündigten Sektoren zu lesen und den Ausgang des Leseversuches zu protokollieren. Ergibt sich dabei zu einem AF ein RNF-Fehler, dann handelt es sich um ein singuläres AF. Es muß zwar bei der Formatierung, nicht aber während des Lesens der originalen Sektoren (von der Originaldiskette) beachtet werden.

Ein gutes Kopierprogramm wird daher nach der Ermittlung der Adressfeldstruktur zuerst alle so angemeldeten Sektoren 'ins Leere' lesen und die dabei gewonnenen Statusmeldungen weiter auswerten.

Als weiterer Vorteil dieser Leseversuche ist anzusehen, daß die Berechnung der Summe aller Bytes der Sektoren dieser Spur nun auf einem festeren Fundament steht. Näheres bringt der folgende Abschnitt.

Berechnung der Spurlänge aus den AF

Das Lesen eines AF liefert 6 Bytes an Informationen, entsprechend den 6 Bytes, aus denen beim Formatieren das AF aufgebaut wird. In der korrekten Reihenfolge sind dies: Spurnummer, Seitennummer, Sektornummer, Sektorlängenkennung sowie 2 CRC-Prüfsummenbytes. Die CRC-Bytes können vernachlässigt werden, es sei denn, man will die Funktion des CRC-Generators im FDC überprüfen (...). Wichtigste Variable für ein Kopierprogramm ist jedoch die Sektorlänge. Allein aus ihr berechnet der FDC beim Lesen und Schreiben die Anzahl Bytes, die der folgende Datensektor enthält. Die steuernde CPU hat keine Möglichkeit, mehr Bytes zu lesen, als der FDC anhand des Längenbytes erkennt. Näheres zu den vom FDC gelieferten Informationen im betreffenden Kapitel 16, zur Spurstruktur sind bei den Aufzeichnungsverfahren (Kapitel 13) mehr Informationen verfügbar.

Durch Berechnung der Zahl der Bytes eines Sektors über das Längenbyte und Summation aller so gewonnenen Sektorlängen der momentan bearbeiteten Spur lässt sich die Anzahl der Datenbytes in einer Spur berechnen. Diese kann, bedingt durch notwendige Rücksichtnahme auf Drehzahlschwankungen des Laufwerks etc., einen Maximalwert nicht überschreiten, da in diesem Fall der letzte Sektor einer Spur den ersten überschreiben würde.

Ein Richtwert kann aus der Maximalzahl von 10 Standard-Sektoren pro Spur bei einfacher und der doppelten Anzahl bei doppelter Dichte berechnet werden. Pro Sektor sind das 256 Bytes, entsprechend einer maximalen Datenbyteanzahl von 2560 (>A00) bzw. 5120 (>1400).

Ein singuläres AF geht in diese Berechnung naturgemäß nicht ein. Existieren jedoch nichtsinguläre AF, mit denen sich eine zu große Spurlänge ergeben würde, so ist zumindest 1 Sektor nicht komplett enthalten. Dieser Sektor kann aus bestimmten Gründen nur Daten unterhalb des Wertes >F5 enthalten, sein Inhalt muß bereits beim Formatieren festgelegt worden sein! Ein solcher Sektor darf nicht mittels eines Sektorschreibbefehls geschrieben werden, da dann der o.a. Fall eintritt, daß er den ersten Sektor am Spuranfang mit Sicherheit überschreibt.

Beim TI wird das bei Advanced Diagnostics und DISkASSEMBLER gemacht.

Gar keine Adreßfelder auffindbar

Vor dem Beginn einer Spuranalyse muß bekannt sein, ob und in welcher Dichte eine Spur formatiert ist. Dies kann nur ausprobiert bzw. vom Bediener abgefragt werden. Es gibt zwei mögliche Verfahren zum Ausprobieren. Das schnellste ist, einen Spurinhalt in einer beliebigen Dichte zu lesen und nach Adressfeld- oder Sektormarken zu suchen, also bei SD nach den Bytes >FB, >FE oder >CB. Wird keines dieser Bytes innerhalb 60 Bytes nach Spuranfang gefunden, so handelt es sich bei der gewählten Dichte um die falsche. So schnell dieses Verfahren ist, so unzuverlässig ist es.

Sicherer ist es, in der willkürlich gewählten Dichte ein Adressfeld zu lesen. Wurde eines ohne CRC-Fehler erkannt, so wurde die richtige Dichte gewählt. Im anderen Fall muß auf eine andere Speicherdichte umgeschaltet werden, woraufhin der Versuch wiederholt wird. Verwendet man einen Speicher für die zuletzt erkannte Dichte, so ist es möglich, die Dichteerkennung zu beschleunigen.

Wurde jedoch auch hier kein korrektes AF entdeckt, so ist diese Spur im allgemeinen als nicht formatiert zu betrachten. Das bedeutet aber nur, daß sie keine AF oder Sektoren im klassischen Sinn enthält. Vielmehr ist anzunehmen, daß Daten spurorientiert abgelegt wurden. Dieses Thema wird in Abschnitt 'Spurkopie' besprochen.

Kann mit hinreichender Sicherheit angenommen werden, daß die Spur leer ist, so wird die Kopie mit >00 formatiert und ist fortan in dieser Spur unlesbar.

Reproduktion der Spur anhand der Adreßfelder

Wurde mithilfe der AF-Analyse eine Spurstruktur analysiert, so kann mit dem Kopieren begonnen werden. Zuerst muß dabei die Spur der Zieldiskette (der Kopie) formatiert werden, bevor mit dem Transfer derSektoren begonnen wird.

Formatieren der Spur anhand der Adreßfelder

Beim Aufbau der Spur in einem Puffer (vor dem Schreiben auf Diskette) müssen die vom 'blinden Lesen' ermittelten Fehlercodes der einzelnen Sektoren beachtet werden.

Begonnen wird mit dem normalen Index-Gap, der das erneute Synchronisieren nach dem durch das Index-Loch verursachten Aussetzer des Datenstroms gewährleistet.

Nach dem Adress-Gap wird das erste AF wie gelesen wieder einformatiert. Es wird mit einem CRC-Auslöser abgeschlossen. Hatte der korrespondierende Sektor die Fehlermeldung Record-Not-Found (RNF) verursacht, so wird der Data-Gap geschrieben und mit dem nächsten AF fortgefahren.

War der Sektor in Ordnung, so wird aus dem Längenbyte die effektive Sektorlänge mit >E5 rohformatiert, gefolgt vom CRC-Auslöser. Der Rest wiederholt sich nach gleichem Schema.

Bei überlangen Spuren kann es sein, daß beim Aufbau des letzten Sektors das Spurende erreicht wird. Die Überwachung der Spurlänge beim Formatieren muß also immer gewährleistet sein!

War ein überzähliger Sektor gemeldet, der eine Überschreitung der maximalen Datenbyteanzahl hervorrief, empfiehlt es sich, diesen Sektor zu lesen und ihn beim Spuraufbau mit diesen Daten in die Rohformatierung einzubeziehen. Das setzt voraus, daß die in ihm enthaltenen Daten programmrelevant sind, was jedoch beim TI bisher nicht aufgetaucht ist.

Anschließend kann der Pufferinhalt auf die Kopiediskette geschrieben werden.

Kopieren der Datensektoren

Auch hier spielen die Adreßfelder eine Rolle. Sie legen die Reihenfolge fest, in der die Sektoren gelesen werden müssen (kritisch ist dies allerdings nur bei doppelten Sektoren), wieviel Bytes pro Sektor gelesen bzw. geschrieben werden müssen und welche Sonderbehandlung ggfs. notwendig wird. Der letzte Punkt 'Sonderbehandlung' wird erst möglich, wenn neben den Adreßfeldern auch die FDC-Fehlercodes beim 'ins Leere lesen' der korrespondierenden Sektoren protokolliert wurden.

So muß ein Sektor mit einem RNF-Fehler beim Lesen wie beim Schreiben übergangen werden, da ein Leseversuch durch einen Timeout des FDC die Kontinuität des Lesens stören und ein Schreibversuch nachfolgende Sektoren zerstören würde.

Ein Sektor, der einen CRC-Fehler aufweist, muß so geschrieben werden, daß entweder keine oder eine falsche Prüfsumme erzeugt wird. Durch einfaches Aufhören mit dem Datentransfer von Seiten der CPU kann dies jedoch nicht erreicht werden, da der FDC aus Sicherheitsgründen in einem solchen Fall den Sektor mit Nullen auffüllt und die Prüfsumme dann halt unter Beachtung dieser Nullen trotzdem korrekt schreibt.

Das Ergebnis: Daten falsch aber Sektor in Ordnung.

Entweder wird beim Formatieren das CRC-Auslösebyte gleich falsch gesetzt und der Sektor beim Schreiben wie beim Lesen übergangen, oder aber der FDC wird wenige Bytes vor Abschluß des Sektors mit einem sofort auszuführenden Interrupt am Fertigstellen des Sektors gehindert.

Es sei nochmals gesondert darauf hingewiesen, daß der kontinuierliche Fluß des Ablaufs beim Lesen der Sektoren der Originaldiskette ein primärer Faktor für die Zuverlässigkeit eines Kopierprogramms ist. Doch auch beim Schreiben ist es wichtig, keine unnötigen Pausen zu verursachen. Dies ist insbesondere bei mehrfach vorhandenen Sektoren mit identischen Adreßfeldern wichtig.

Prüfung der Datensektoren

Ein Verify ist auch bei einem eigenen DOS kein Luxus, zumal er sehr einfach zu realisieren ist (auch wenn das bei der Programmierung des Moduls TI-IBM Connection wohl anders gesehen wurde).

Nachdem alle AF der Originaldiskette analysiert und die entsprechenden Sektoren zum Kopieren in einen Puffer geholt wurden, geschieht dies identisch auf der Kopie. Dabei kann auf die Adressfeldanalyse verzichtet werden, da deren Struktur durch die Formatierung hinreichend sicher reproduziert wurde. Wurden alle kopierten Sektoren gelesen, folgt der Vergleich der Fehlerprotokolle.

Stimmen diese überein, kann die Anzahl der Datenbytes der Spur berechnet werden (oder der vorher gesicherte Wert verwendet werden) und diese Anzahl Bytes zwischen beiden Pufferzonen verglichen werden. Bei einem Fehler können die Sektorinhalte aus dem Puffer der Originaldisk erneut geschrieben werden.

Spurkopie

Es kann nun aber vorkommen, daß eine Spur absolut keine erkennbare Struktur aufweist, jedoch nachweislich genutzte Daten enthält (z.B. Diskassembler).

In einem solchen Fall gibt es das Problem der Synchronisation.

Noch mehr als bei einem Sektorzugriff muß beim puren Lesen einer Spur damit gerechnet werden, daß entweder von vornherein unsauber synchronisiert wird, oder aber der FDC dann, wenn keine Bytes mit MissingClock auftauchen, nach einigen Bytes 'out of sync' ist.

Eine derartige Spur muß also mehrfach gelesen werden, noch besser wäre es, etwa eine Quersumme über den Spurinhalt zu bilden und so oft wiederholt zu lesen, bis die Änderungen dieser Quersumme einen Minimalwert unterschreiten.

Da ein Header einer derart geschützten Spur so aufgebaut sein muß, daß auch das Originalprogramm die Daten lesen kann, ist es sodann durchaus legitim, das Spurbild mit einem eigenen Header (Sync-Bytes, dann z.B. ein DAM oder IDAM, siehe Abschnitt 13, Spurstruktur) zu versehen und auf die Zieldiskette zu schreiben.

Aber Achtung: Hierbei darf man niemals übersehen, daß der FDC beim Spur schreiben, also Formatieren, NICHT DATENTRANSPARENT ist, und einige Bytes nicht so geschrieben werden, wie sie von der CPU kommen (hier liefert das Kapitel 16 weitere Details). Beim Diskassembler etwa befinden sich nur Bytes mit ASCII-Codes unter >7F in der betreffenden Spur.

Ausblick

Der Wettlauf zwischen Kopierschützern und denen, die den Schutz kopieren oder entfernen, würde Herrn Darwin eine Menge Freude bereiten. So analysieren die Kopierschützer die Kopierprogramme, decken Schwachstellen im Algorithmus auf und passen den Schutz eines neuen Programmes an. Kopierprogramme sind daher verderbliche Ware und erzwingen gewissermaßen ein laufendes Update. Sinnvoller ist es, dem Programm Features mitzugeben, mit deren Hilfe der Anwender selbst Disketten 'von Hand' kopieren kann.

Der Floppy-Disk-Controller-Formatter-Chip (FDC)

Der Chip um den sich so viele Geheimnisse ranken soll nun Gegenstand einer eingehenden Betrachtung sein. In diesem Kapitel werden Hinweise zu Funktion und Ansteuerung der verschiedenen auf dem TI eingesetzten FDC-Chips gegeben und im nächsten Kapitel mit Beispielprogrammen illustriert.

Anatomie einer Diskcontroller-Karte

Eine Diskcontrollerkarte besteht im wesentlichen aus den folgenden Komponenten:

- Dem DSR-ROM mit den Level 2,1 und 0 Routinen.
- Dem CRU-Interface zur Steuerung der Laufwerke, der Wait-State-Logik, dem DSR-Banking und des FDC.
- Dem FDC selbst.

sowie jeder Menge Leim (das sind die TTLs, die das Ganze zusammen halten).

Das DSR-ROM beinhaltet alle Programme und Programmteile die notwendig sind, ein standardisiertes DOS zu realisieren. Auf dem niedrigsten Level, Level 0, stehen Operationen wie 'Spur anfahren', 'Spur schreiben (formatieren)', 'Sektor schreiben/lesen' und 'Kopf auf Spur Null fahren' zur Verfügung. Auf Level 1 können z.B. ganze Disketten formatiert, Files kopiert und Pufferzonen reserviert werden. Auf Level 2 findet die ganze Datenverwaltung im logischen Verbund (File) statt.

Das CRU-Interface erlaubt die Anwahl bestimmter Laufwerke, das Umschalten der Schreib-Leseköpfe, die Wahl der Speicherdichte sowie den Start der Laufwerkmotoren und die Freigabe von Head-Load und CPU-Wait-States durch den FDC.

Der FDC selbst stellt das Bindeglied zwischen Diskstation und Rechner dar, wozu er über eine gewisse 'Eigenintelligenz' verfügt - mit all den Vorbehalten, die dieses Wort in Bezug auf Computer fordert.

Der FDC ist in der Lage, einfache Kommandos wie 'Spur suchen', 'Sektor suchen und lesen/schreiben' usw. auszuführen. Diese oft zeitkritischen Arbeiten werden der CPU abgenommen, die nach Erteilung eines Befehls/Kommandos an den FDC nur noch für die Anlieferung bzw. Abholung von Daten sorgen muß oder im einfachsten Fall nur abwartet, bis der FDC den Befehl abgearbeitet hat und dann die korrekte Ausführung prüft.

Im TI kommen FDC-Chips von Western Digital zum Einsatz, vorwiegend die Typen 1771 (Original Controller), 1772 (Myarc DDCC-1) und 1773 (CorComp Rev.1 und Atronic). Alle FDC besitzen 5 Register, die im DSR-Adreßbereich von >5FF0 bis >5FFE liegen (Myarc: >5F01 - >5F07). Durch die Aufspaltung auf 8 Adressen (nur gerade Byte-Adressen werden verwendet), wird die Software transparenter, da jede Speicheradresse nur für eine Funktion nutzbar ist. Bei Myarc findet keine Aufspaltung statt - alle Ports sind bidirektional, daher nur 4 Bytes, jedoch an ungeraden Adressen. Details zum Myarc-Controller finden sich im ROM-Listing.

Bedeutung der FDC-Register

Mit Einschalten der Floppy-Disk-Steuerkarte auf der ihr zugewiesenen CRU-Seite (allgemein >1100) mit dem SBO 0 Befehl, werden 8 Adressen am oberen Rand des DSR-ROMs zugänglich, die wie folgt belegt sind:

- >5FF0 Status-Register des FDC lesen. Dieses Register kann nur gelesen werden und liefert nach bzw. während einer Operation des FDC Informationen über den Verlauf der Befehlsausführung.
- >5FF2 Spur-Register lesen. Hier findet sich im Normalfall die Nummer der Spur, über der sich der Schreib/Lesekopf des gewählten Disklaufwerkes gerade befindet.
- >5FF4 Sektor-Register lesen. Dieses Register enthält die Nummer des Sektors der gelesen bzw. geschrieben wurde oder werden soll.
- >5FF6 Daten-Register lesen. Hier werden während eines Spur-, Adreßfeld- oder Sektorlesebefehls die von der Diskette seriell gelesenen Daten in 8 Bit Parallel-Form an den Rechner übergeben. Liest der Rechner dieses Register bevor ein komplettes Byte von Disk gewandelt wurde, so wird er bis zum Ende der Seriell-Parallel-Wandlung angehalten.
- >5FF8 Kommandoregister schreiben. Ein in dieses Register geschriebenes Byte veranlaßt den FDC zur sofortigen Ausführung des Kommandos, welches das betreffende Byte vorgab. Der Inhalt dieses Registers kann nicht rückgelesen werden!

- >5FFA Spur-Register schreiben. Bei einem Wechsel des Laufwerks muß in dieses Register die Nummer der Spur geschrieben werden, über welcher der S/L-Kopf beim letzten Zugriff auf dieses Laufwerk stehen gelassen wurde. Ein in dieses Register geschriebenes Byte kann an >5FF2 rückgelesen werden.
- >5FFC Sektor-Register schreiben. In dieses Register wird die Nummer des Sektors geschrieben, der als nächster gelesen bzw. geschrieben werden soll. Rücklesbar über >5FF4.
- >5FFE Daten-Register schreiben. Beim Schreiben auf Diskette übergibt die CPU über dieses Register die Daten. Verfrühte Anlieferung wird wie beim Lesen aus >5FF6 mit Wartezyklen der CPU synchronisiert.

CRU-Interface

Auf der CRU-Seite des Controllers sind bis zu 20 CRU-Bits mit verschiedenen Funktionen belegt. 8 davon sind bei allen in dieser Diskinfo behandelten Typen verfügbar, 8 Eingabebits gibt es nur beim TI-Controller und 8 spezielle Eingabebits werden vom CorComp-Controller benutzt. Welche wie belegt sind zeigt die folgende Tabelle (Ausgabebits 8 - 11 nicht bei TI-Controller!).

Relative Bit-Nr.	'E'in-,'A'usgang	Beschreibung
0	A	Schaltet Karte an/aus
1	A	Negative Flanke triggert Laufwerkmotoren. Sequenz SBZ 1, SBO 1.
2	A	Wartezyklen erlauben.
3	A	Head-Load aktivieren.
4	A	Laufwerk 1 anschalten.
5	A	Laufwerk 2 anschalten.
6	A	Laufwerk 3 anschalten.
7	A	Seiten-/Kopfwahl. SBZ 7 Unterseite (normal), SBO 7 Oberseite.
8	A	Laufwerk 4 anschalten.
9		nicht benutzt.
10	A	Dichte umschalten. SBO >A - Single Density, SBZ >A - Double Density.
11	A	Zweite Bank des DSR-ROMs schalten.

Folgende Eingabebits stehen nur beim TI-Controller zur Verfügung:

Bit 0	Head-Load Ausgang des FDC (hier 1771)
Bit 1	Sensor, daß LW 1 vorhanden ist. Vor Zugriff auf ein Laufwerk kann dessen Existenz so getestet werden Pegel 1 - nicht angeschlossen Pegel 0 - angeschlossen
2	Sensor für LW 2.
3	Sensor für LW 3.
4	Motor-An Sensor. Pegel 1 - aus, 0 - an.
5	nicht benutzt, an 5 Volt gelegt.
6	nicht benutzt, an 5 Volt gelegt.

7	nicht benutzt, an Masse gelegt.

Mit den folgenden Eingabebits testet der CorComp-Controller die Stellung der 8 DIP-Schalter auf der Karte und ermittelt so die jeweils gewünschte Step-Zeit.

Bit >12	LW 1 Low-Order Step-Bit
>16	LW 1 High-Order Step-Bit
>15	LW 2 Low-Order Step-Bit
>0F	LW 2 High-Order Step-Bit
>14	LW 3 Low-Order Step-Bit
>0E	LW 3 High-Order Step-Bit
>13	LW 4 Low-Order Step-Bit
>11	LW 4 High-Order Step-Bit

Die etwas chaotische Reihenfolge ergibt sich aus einem möglichst einfachen Layout für den DIP-Schalter an den TMS9901 des CorComp-Controllers. Da die Eingabebits ohnehin aus einer Tabelle heraus abgearbeitet werden, spielt deren tatsächliche Nummer eine eher untergeordnete Rolle. Die Bedeutung der Step-Bits wird bei den Befehlen des FDC erklärt.

Befehlsvorrat des FDC

Die verschiedenen Befehle, die der FDC versteht, können aufgrund der von ihnen ausgelösten Aktionen in 4 verschiedene Gruppen eingeteilt werden, die mit römischen Ziffern gekennzeichnet werden. Die Befehlstypgruppen sind wie folgt benannt:

Typ I	Steuerung der Kopfposition
Typ II	Sektoroperationen
Typ III	Spuroperationen
Typ IV	Interruptsteuerung

Alle Befehle bestehen - wie der Maschinencode der CPU - aus einem Grundbefehlswort, in dem durch einzelne Bits eine Befehlsspezifikation erfolgt.

Die vollständig definierten Befehle werden über das Kommandoregister direkt an den FDC übergeben, der direkt im Anschluß an diese Übergabe mit der Abarbeitung des Befehls beginnt. Dabei muß sichergestellt sein, daß ein eventuell vorher gegebener Befehl bereits abgearbeitet ist (Test über BUSY im Statusregister, siehe dort). Eine Ausnahme bilden die Typ IV-Kommandos. Nur sie werden in das Kommandoregister übernommen, wenn das BUSY-Bit gesetzt ist. Sie stoppen den aktuellen Befehl und lösen nach dem Eintreten bestimmter Bedingungen einen Interrupt aus, der jedoch im TI nicht direkt ausgewertet wird. Alle anderen Befehlstypen werden während der Bearbeitung eines Befehls nicht in das Kommandoregister übernommen.

Leider ist der FDC bei Befehlen, die aufgrund eines 'drive not ready' nicht ausgeführt werden können, etwas nachtragend. Wird in einem solchen Fall nicht über einen Interrupt der FDC rückgesetzt, so 'merkt' sich der Chip den Befehl, und führt in bei nächster Gelegenheit aus. So kann es fatale Folgen haben, wenn eine Formatierung versucht wird, die fehlschlägt und gleich danach mit einem Sektor-Lesebefehl versucht wird, nachzusehen, was denn los war…

Für die Abarbeitung der Befehle im FDC wird ein controllereigener Takt verwendet. Da das Timing aus diesem Takt abgeleitet wird und bei den Double-Density-Systemen zusätzlich ein über CRU-Bit 10 schaltbarer Teiler vorhanden ist, müssen folgende Mindestwartezeiten zwischen zwei Zugriffen auf Register des FDC beachtet werden (entnommen aus den Datenblättern):

- Kommandoregister schreiben, dann Statusregister lesen: 32 µs
- Register schreiben, dann selbes Register lesen: 16 µs

Die Zeiten verdoppeln sich, wenn ein 1771-Chip angesprochen wird bzw. wenn Single Density aktiviert wurde.

Da die TMS9900 CPU keinen direkten Bit-Test wie z.B. die 6502-CPU auf Speicherinhalte anwenden kann, wird der Sonderfall des direkten Tests von Status-Bit 0 (BUSY-Bit), bei dem nur 24 µs Wartezeit anfallen, hier nur der Vollständigkeit halber erwähnt.

Das Statusregister

Von diesem Register wird auf den folgenden Seiten sehr viel die Redesein. Daher sollen die Bedeutungen der 8 Bits bei den verschiedenen Befehlen vorab erklärt werden, auch wenn die Befehle, die zu diesen Bedeutungen führen, erst später besprochen werden. Je nach Art des abgearbeiteten Befehls variiert die Bedeutung einiger Bits.

Typ I Kommandos

Bit	Kurz-Name	Funktion
7	NOT READY	Laufwerk ist nicht bereit (z.B. Drehzahl nicht in Ordnung).
6	WRITE PROTECT	Diskette ist schreibgeschützt. Kann zum gefahrlosen Testen des Schreibschutzes benutzt werden.
5	HEAD LOADED	Kopf des gewählten Laufwerkes liegt auf Disk.
4	SEEK ERROR	Dieses Bit ist nur gültig, wenn das v-Bit im entsprechenden Befehl gesetzt war. Ist dieses Bit log. 1, so wurde die angefahrene Spur nicht verifiziert. Das kann durch einen Fehlschritt des Laufwerkes oder aber durch fehlerhafte Adreßfelddaten bewirkt werden.
3	CRC-ERROR	Auch dieses Bit ist nach Typ I Kommandos nur dann gültig, wenn das v-Bit aktiv war. Ist dieses Status-Bit log.1, so wurde während der Verifikation ein defektes AF gelesen.
2	TRACK ZERO	Durchgeschaltete Leitung vom Laufwerk, jedoch im Pegel invertiert.
1	INDEX-PULSE	Durchgeschaltete Leitung vom Laufwerk, invertiert.
0	BUSY	FDC ist noch beschäftigt, nicht stören!

Typ II und III Kommandos

7	NOT READY	Laufwerk ist nicht bereit.
6	RECORD-TYPE	1771: siehe Befehl 'Sektor lesen'.
	WRITE PROTECT	Alle: Disk ist schreibgeschützt.
5	RECORD-TYPE	1771: siehe Befehl 'Sektor lesen'.
	WRITE FAULT	Alle: Schreibfehler.
4	RECORD NOT FOUND	Kein AF bzw. gewünschter Sektor nicht gefunden
3	CRC-ERROR	Daten fehlerhaft (Bit 4 = 0) oder CRC-Adreßfehler (Bit 4 = 1).
2	LOST DATA	DRQ wurde nicht korrekt bedient, d.h. zu spät Daten an Datenregister geliefert bzw. geholt.
1	DATA REQUEST,DRQ	Datenregister ist voll (Lesebefehl) bzw. geleert (Schreibbefehl).
0	BUSY	FDC ist noch beschäftigt.

Typ I Kommandos des FDC

Wie bereits einleitend bemerkt, finden sich in dieser Befehlsgruppe all jene Kommandos, welche die Position des Schreib-Lesekopfes des angewählten Laufwerkes beeinflussen.

	7	6	5	4	3	2	1	0	
Restore	0	0	0	0	h	v	s1	s2	Kopf über Spur Null fahren
Seek Track	0	0	0	1	h	v	s1	s2	Spur direkt anfahren
Step	0	0	1	u	h	v	s1	s2	1 Schritt in letzte Richtung
Step In	0	1	0	u	h	v	s1	s2	1 Schritt nach innen
Step Out	0	1	1	u	h	v	s1	s2	1 Schritt nach außen

Die Options-Bits sind wie folgt definiert (aktiv wenn log. 1):

h

1771/73: Head-Load-Timing. Aktiviert Head-Load.

1772: Spin-Up-Delay. Bewirkt eine Verzögerung von 6 Umdrehungen der Diskette um die Nenndrehzahl zu erreichen.

v Verify-On-Track.

Ist dieses Bit gesetzt, dann liest der FDC nach Erreichen der gewünschten Spur ein beliebiges AF, dessen Spurkennung er mit dem Inhalt des Track-Registers vergleicht. Der 1771 legt zuvor jedoch noch eine 10 ms lange Pause ein, sofern weniger als 2 Umdrehungen zuvor nicht ein Typ II Kommando kam, dessen e-Bit Null war. Stimmt entweder die Spurnummer oder die CRC-Prüfsumme eines AF mit korrekter Spurnummer nicht, so wird der Befehl automatisch für die Dauer von 5 (2 bei 1771) Umdrehungen der Diskette wiederholt. Wird in dieser Zeit keine passende Spurnummer gefunden, so wird das SEEK-ERROR-Bit im Status-Register gesetzt. Wurde eine korrekte Spurnummer in einem fehlerhaften AF entdeckt, wird das CRC-ERROR-Bit gesetzt.

u Update-Flag.

Wird dieses Bit gesetzt, dann wird der Inhalt des Track-Registers mit jedem Schritt des Kopfes nach innen oder außen inkrementiert bzw. dekrementiert.

s1,s2 Step-Timing-Bits.

Diese Bits legen fest, welche Folgefrequenz die Step-Impulse haben, mit denen der Kopf im momentan bearbeiteten Befehl bewegt wird. s1 ist dabei das 'High-Order-Bit' und s2 entsprechend das 'Low-Order-Bit'. Es ergeben sich folgende Step-Zeiten:

s1	**s2**	**Resultierende Wiederholrate**
0	0	6 Millisekunden
0	1	12 Millisekunden
1	0	20 Millisekunden
1	1	30 Millisekunden

Beim CorComp-Controller alter Bauart (beiges Gehäuse) ergeben sich durch die 2 Mhz (statt 1 Mhz) Taktfrequenz die doppelten Wiederholraten d.h. die Zeiten sind halbiert. Die Raten für 1772 sind 2, 3, 5 und 6 ms.

Die Breite der Step-Impulse ist bei DD (Double-Density) 4 µs und bei SD (Single-Density) 8 µs.

RESTORE - Kopf über Spur Null bringen

Dies ist der einzige Befehl, mit dem es möglich ist den Aufenthaltsort des Schreib-Lesekopfes über Hardware zu erfahren. Dieser Befehl muß immer dann ausgeführt werden, wenn zum ersten Mal auf dieses Laufwerk zugegriffen wird.

Nach Erhalt dieses Befehls prüft der FDC die Leitung TRK 00 des angesprochenen Laufwerks. Ist dieser bereits aktiv, so wird das Track-Register mit >00 geladen und der Befehl beendet.

Ist dieses Signal nicht aktiv, so wird die Richtungsleitung DIR auf 'nach außen', also log. 0 gesetzt. Anschließend werden Schritt-Impulse mit der gewählten Wiederholrate so lange über die Leitung STEP ans Laufwerk gegeben, bis die Leitung TRK 00 aktiv wird.

Wird nach 255 Impulsen kein TRK 00 gemeldet, dann wird der Befehl abgebrochen und das SEEK-ERROR-Bit im Status Register gesetzt. Dies geschieht nur, wenn das v-Bit im RESTORE-Befehl gesetzt war.

Wird die Spur Null (TRK 00) korrekt erreicht und ist das v-Bit im Befehl gesetzt, so wird ein Verify wie oben besprochen durchgeführt. Das SEEK-ERROR-Bit wird entsprechend gesetzt oder gelöscht. Das Track-Register wird jedoch in jedem Fall auf >00 gesetzt, wenn TRK 00 aktiv wurde.

SEEK - Suche Spur

Dieses Kommando bewirkt eine softwareabhängige Suche nach einer bestimmten Spur. Dabei wird vom FDC angenommen, daß sich die Nummer der Spur, über welcher sich der Kopf des aktuellen Laufwerks gerade befindet, im Spurregister (Track-Register) befindet. Die Nummer der anzufahrenden Spur wird im Datenregister erwartet.

Der FDC berechnet selbständig die Differenz, erkennt die notwendige Richtung (DIR) und gibt die erforderliche Anzahl Schrittimpulse mit der in s1 und s2 spezifizierten Wiederholrate an das aktive Laufwerk. Das Spurregister wird dabei so lange inkrementiert bzw. dekrementiert, bis dessen Inhalt mit dem im Datenregister angegebenen Wert übereinstimmt.

!!ACHTUNG!!

Der Programmierer muß hier darauf achten, daß im Extremfall 256 Step-Impulse (bis zum Overflow) gegeben werden können, wenn die Start- oder Zielspur falsch angegeben wurde. Es muß zur Schonung der Laufwerke sichergestellt sein, daß keinesfalls die maximal zulässige Spurnummer überschritten wird!

Je nach Zustand von h und v-Flag wird eine Pause vor Beginn der Ausführung eingelegt bzw. anhand eines AF das korrekte Erreichen der spezifizierten Spur geprüft und im Status-Register protokolliert.

STEP - Schritt in letzte Richtung

Da die DIR-Leitung nach Befehlsende auf ihrem letzten Pegel bleibt, kann mit diesem Befehl in die zuletzt eingeschlagene Richtung weiter'gefahren' werden. Es wird 1 Step-Impuls ausgelöst, die in s1 und s2 angegebene Zeit gewartet und anschließend der Befehl beendet.

War das v-Bit im Befehl gesetzt, wird nach Ablauf der Wartezeit mittels eines AF die Spurnummer im Spur-Register geprüft und ggfs. ein SEEK-ERROR gemeldet.

War das u-Bit gesetzt, so wird der Inhalt des Spur-Registers je nach Schrittrichtung in- oder dekrementiert. Durch Setzen des h-Bits kann die Anlaufpause aktiviert werden.

STEP IN - Schritt nach innen

Nach Erhalt dieses Befehls schaltet der FDC die Leitung DIR auf log. 1 (innen) und gibt einen Schritt-Impuls an das aktive Laufwerk.

Die Optionsbits h und v werden wie bei den zuvor besprochenen Befehlen behandelt, bei aktivem u-Bit wird der Inhalt des Spur-Registers inkrementiert.

STEP OUT - Schritt nach außen

Wie STEP-IN, jedoch wird hier DIR auf log. 0 gelegt und bei aktivem u-Bit der Inhalt des Spur-Registers dekrementiert.

Typ II Kommandos des FDC

In dieser Befehlsgruppe finden wir die Sektorlese- und Sektorschreibbefehle. Die Anzahl der bei einem Sektorzugriff zu bewegenden Bytes wird allein durch das Längenbyte im Adreßfeld des gewählten Sektors bestimmt. Der Programmierer hat keine Möglichkeit, mehr Bytes aus einem Sektor zu lesen, als dies das beim Formatieren festgelegte AF vorsieht.

Im Gegensatz zum FDC des Typs 1795/97 ist es beim 1772 oder 1773 nicht möglich, die Interpretation des AF-Längenbytes umzuschalten, womit dann Sektoren 'übergelesen' werden können. Ein Kopierprogramm für einen 1772 oder 1773 ist beim Auftauchen solcher Sektoren an der Grenze der Hardware angekommen.

Die beim 1771 wählbare Interpretation des Längenbytes als Vielfaches von 16 wird weiter unten erwähnt.

Hier nun die Befehlsstruktur:

	7	6	5	4	3	2	1	0	
Read Sector	1	0	0	m	b	e	a1	0	Sektor lesen
Write Sector	1	0	1	m	b	e	a1	a2	Sektor schreiben

Die Optionen sind:

m Multiple-Record-Flag.

Erlaubt das en-bloc Lesen bzw. Schreiben von mehreren Sektoren mit fortlaufenden Nummern. Dazu wird das Sektor-Register laufend inkrementiert. Wird dabei die größte Sektornumer innerhalb der Spur überschritten, so wird der Befehl nach 5 Umdrehungen der Diskette abgebrochen. In diesem Fall wird das RECORD-NOT-FOUND-Bit im Status-Register gesetzt. Weist ein Sektor einen CRC-Fehler auf, so wird der Befehl abgebrochen und das CRC-ERROR-Bit gesetzt. Der Befehl kann bzw. sollte nach Abarbeitung aller gewünschter Sektoren mit einem Interrupt beendet werden um nicht den Timeout abwarten zu müssen. Ist das m-Bit nicht gesetzt, so wird jeweils nur 1 Sektor bearbeitet.

b 1771: Block-Length. Sektorlängeninterpretation.

Bit gesetzt: IBM-Zuordnung, wie 1772, 1773.

Bit nicht gesetzt: Sonderformat.

AF-Längenbyte	Sektorlänge in Bytes
>00	4096
>01	16
>02	32
>03	48
…	…
>FF	4080

Man kann also das Längenbyte des AF als Vielfaches der Byteanzahl 16 ansehen, wobei bei >00 ein Übertrag hinzukommt. Ein 'überlesen' eines derart gekennzeichneten Sektors ist jedoch nur in Ausnahmefällen möglich.

1772: Siehe Typ I Kommandos h-Bit.

1773: Side-Compare. Testet das Spurnummerbyte im AF auf den Wert dieses Bits (a1 muß 1 sein, sonst erfolgt kein Test). Die ser Test wird zum Update des SEEK-ERROR-Bits herangezogen.

e Enable-Delay. Head-Load Beruhigungszeit.

1771: Head-Load-Magneten aktivieren, 10 ms Pause.

1772, 1773: 30 ms Pause vor Befehlsbeginn.

a1 1772: Write-Precompensation-Disable.

Ist dieses Bit aktiv, so wird auf den inneren Spuren so geschrieben wie auf den äußeren. Wird dieses Bit Null gesetzt, so wird auf den inneren Spuren zur Vermeidung von Bit-Verschiebungen ein Versatz von 125 ns zwischen Daten und Takt eingefügt. Diese Option ist nur in DD verfügbar.

a1 1773: Enable-Side-Compare.

Aktiviert Seitentest mit 'b'.

a2 1772, 1773: Data-Address-Mark.

Normalerweise ist dieses Bit nicht gesetzt, weshalb beim Schreiben eines Sektors zu Beginn der Datenzone das Byte >FB geschrieben wird, was einen Sektor mit gültigen Daten anzeigt. Bei gesetztem Bit wird eine Kennung >F8 geschrieben, was ein sog. 'DELETED-DATA-MARK' ist. Beim Lesen eines solchen Sektors wird der Wert dieses Bits im Status-Bit 'Record-Type' übergeben.

a2 1771

Hier werden a1 und a2 gemeinsam betrachtet:

a1	a2	Sektorkennung	Data-Mark Typ
1	1	>F8	DELETED-DATA-MARK
1	0	>F9	User-Defined 1
0	1	>FA	User-Defined 2
0	0	>FB	Normales DATA-MARK

Beim Lesen der Sektoren über einen 1771 werden die Bits a1 und a2 in den Bits 6 und 5 des Status-Registers übergeben.

Die Standard-Sektorlängenzuordnung (1772, 1773 oder 1771 mit b-Bit log.1) ist wie folgt:

AF-Inhalt	Sektorlänge in Bytes
>00	128
>01	256
>02	512
>03	1024

Sektor lesen

Die Funktionen, die der FDC beim Lesen eines Sektors ausführt sind vielfältig. Bevor jedoch der Lesebefehl gegeben wird, muß im Sektorregister die Nummer des Sektors abgelegt werden, der gelesen werden soll. Ebenso muß im Spurregister ein Wert stehen, der im AF des betreffenden Sektors zu erwarten ist. Nach Befehlsbeginn sucht der FDC ein AF, dessen Spur- und Sektornummern mit den Registerinhalten übereinstimmen. Vor dieser Aktion werden ggfs. spezifizierte Pausen eingelegt.

Wurde ein passendes AF ohne CRC-Fehler gefunden, so wird auf das DATA-ADDRESS-MARK gewartet, nach dessen Eintreffen mit dem Transfer der Daten zur CPU begonnen wird. Dabei muß die CPU entweder laufend das DRQ-Bit testen, ob ein Byte komplett angeliefert wurde (das Datenregister also mit dem Wert des vollen DATA-SHIFT-Registers geladen wurde), oder die CPU greift zu früh zu und wird per Hardware über den DRQ-Pin des FDC in Wartezyklen gezwungen (so beim TI). Wurden alle spezifizierten Bytes gelesen, so wird die CRC-Prüfsumme vom FDC gelesen und mit der inzwischen berechneten verglichen. Bei Übereinstimmung beider Werte ist der Lesevorgang korrekt abgeschlossen, andernfalls wird das CRC-ERROR-Bit im Status-Register gesetzt, woraufhin die CPU den Leseversuch wiederholen kann.

Ein korrektes AF (CRC O.K., Spur und Sektor O.K.) muß beim 1771 innerhalb 2, bei 1772 und 1773 innerhalb 5 Umdrehungen der Diskette gefunden werden. Zusätzlich muß bei 1772 und 1773 das DATA-ADDRESS-MARK bei SD innnerhalb 30 (1771 erlaubt nur 28 Bytes) und bei DD innerhalb 43 Bytes nach der CRC-Prüfsumme des zuletzt gelesenen AF gefunden werden. Wird eine dieser Bedingungen verletzt, so wird der Befehl abgebrochen, wobei im Status-Register das RECORD-NOT-FOUND-Bit gesetzt wird.

Sektor schreiben

Dieser Befehl ist das logische Gegenstück zum Sektorlesebefehl. Auch hier muß vor Beginn der Abarbeitung das Sektorregister mit der Sektornummer und das Spurregister mit der korrekten Spurnummer geladen werden. Zusätzlich muß der Typ des DATA-MARKs bestimmt werden.

Nun sucht der FDC das korrekte AF mit intakter Prüfsumme. Wurde dieses gefunden, so wartet der FDC die Zeit von 11 Bytes (SD) bzw. 22 Bytes (DD) und aktiviert dann, wenn sich bereits ein Byte im Datenregister befindet, die Schreibelektronik. Vor dem eigentlichen Datentransfer werden 6 Bytes (SD) bzw. 12 Bytes (DD) >00 geschrieben, um GAP 2 zu erhalten. Ist dies geschehen, so muß die CPU zu den richtigen Zeitpunkten die Daten im Datenregister >5FFE abliefern, wobei deren Zahl wie zuvor bekannt sein muß. Nach dem letzten Datenbyte schreibt der FDC noch die inzwischen berechnete CRC-Prüfsumme (2 Bytes).

Das LOST-DATA-Bit im Status-Register wird gesetzt, wenn die CPU auch nur 1 Mal ein Byte zu spät anlieferte. Befand sich nach Ablauf der Wartezeit noch kein von der CPU geliefertes Byte im Datenregister, so wird nichts geschrieben, der FDC beendet den Befehl bereits jetzt und setzt das LOST-DATA-Bit.

Beendet die CPU die Datenübergabe zu früh oder kommt sie in Verzug, so schreibt der FDC ein 0-Byte und läßt diesen Wert in die CRC-Erzeugung normal eingehen.

Wurde versucht, auf eine nicht formatierte Diskette zu schreiben, so wird ein Schreibfehler gemeldet.

Typ III Kommandos

Diese Kommandos kommen dann zur Anwendung, wenn Spuren entweder geschrieben, gelesen oder analysiert werden sollen. Folgende Befehle sind verfügbar:

	7	6	5	4	3	2	1	0	
Read Address	1	1	0	0	0	e	0	0	Adreßfeld lesen
Read Track	1	1	1	0	0	e	0	s	Spurinhalt lesen
Write Track	1	1	1	1	0	e	0	0	Spur schreiben/formatieren

Das e-Bit im Read-Address-Befehl wird wie gehabt interpretiert, beim Read-Track und Write-Track-Befehl des 1771 ist das e-Bit fest auf 1 zu setzen, 1772/73 werten es auch hier aus.

Das s-Bit ist eine Spezialität des 1771. Es erlaubt während des Lesens einer Spur die laufende Neu-Synchronisation nach jedem DATA-ADDRESS-MARK, das ja, wie bereits zuvor beschrieben, einen gewollten Codierungsfehler als Kennung besitzt. Ist das s-Bit gesetzt, so wird NICHT laufend neu synchronisiert, ansonsten wird synchronisiert. Die Typen 1772/73 kennen dieses Bit nicht, sie synchronisieren laufend. Bei ihnen ist s auf Null zu setzen.

Adreßfeld lesen

Nach Auslösung dieses Befehls liest der FDC das erste Adreßfeld, das gerade vorbei kommt. Er übergibt Daten an die CPU ähnlich dem Sektorlesebefehl.

Es werden immer 6 Bytes übergeben: Spurnummer, Seitennummer, Sektornummer, Längenkennung und 2 CRC-Bytes.

Wies das gelesene AF einen CRC-Fehler auf, so wird das CRC-ERROR-Bit im Status-Register gesetzt. Nach Ausführung des Befehls befindet sich die Spurnummer (beim 1771 die Sektornummer) des gerade gelesenen AF im Sektorregister. Damit kann die korrekte Datenübergabe an die CPU kontrolliert werden.

Spur lesen

Die Abarbeitung dieses Befehls beginnt mit der nächsterreichbaren positiven Flanke des Index-Pulses und endet mit der folgenden positiven Flanke. Alle in dieser Zeit ankommenden Bytes werden über das Datenregister >5FF6 der CPU zur Verfügung gestellt. Die Anzahl der Bytes variiert mit der Laufwerksdrehzahl. Die CPU sollte daher immer mehr Bytes zu lesen versuchen. Bei Timing-Fehlern wird das LOST-DATA-Bit wie gehabt gesetzt, das CRC-ERROR-Bit ist jedoch ohne Bedeutung. Liest die CPU mehr Daten als vorhanden, so entsprechen die 'übergelesenen' Daten dem zuletzt von Diskette gelesenen Byte.

Bei einfacher Dichte werden Adreßfeld-ID, AF-Inhalt und CRC, DATA-ADDRESS-MARK, Daten und CRC richtig gelesen, die GAPs können in der Länge variieren. Entgegen dieser Aussage des Datenblattes ist es jedoch nicht sicher, das DAM (DATA-ADDRESS-MARK) korrekt zu lesen.

Ebenso ist es bei doppelter Dichte (beim TI-DD-Format) entgegen den Datenblattangaben nur möglich, AF-ID, AF-Inhalt und CRC sowie DAM korrekt zu lesen. Die Daten der Sektoren sind in der Regel ab etwa dem zwanzigsten Byte fehlerhaft.

Spur schreiben/formatieren

Das ist der Befehl, der es unmöglich macht ein ideales Kopierprogramm zu schreiben. Wäre es möglich, die Daten so in die Spur zu schreiben, wie sie der Lesebefehl las, dann wäre es Essig mit Kopierschutz. Doch leider - Absicht oder nicht - schreibt der FDC nicht alle Daten so, wie er sie geliefert bekommt. Und das hat leider gute Gründe.

Doch zuerst zur Funktion.

Wie beim Spur-Lesebefehl beginnt die Abarbeitung mit der nächstfolgenden positiven Flanke des Index-Pulses und endet mit der darauf folgenden positiven Flanke. In der Zwischenzeit übergibt die CPU dem FDC die zuvor bereitgestellten Daten.

Die Status-Bits LOST-DATA und WRITE-FAULT sowie WRITE-PROTECT werden wie beim Sektorschreibbefehl verwendet, jedoch ist RECORD-NOT-FOUND sinnvollerweise nicht gültig.

Je nach gewählter Speicherdichte werden einige Bytes nicht so geschrieben, wie sie ins Datenregister gebracht werden. Wie die Interpretation im Einzelfall aussieht, zeigt die Tabelle:

	Interpretation Single Density		**Double Density**
Byte	**1771**	**1772/1773**	**1772/1773**
>00 - >F4	Byte schreiben, Takt >FF	Byte schreiben, Takt >FF	Byte schreiben
>F5	Byte schreiben, Takt >FF	Nicht erlaubt !	>A1 schreiben, p
>F6	Byte schreiben, Takt >FF	Nicht erlaubt !	>C2 schreiben
>F7	2 CRC-Bytes schreiben	2 CRC-Bytes schreiben	2 CRC-Bytes schreiben
>F8 - >FB	Byte schreiben, Takt >C7 (DAM) p	Byte schreiben, Takt >C7, p	Byte schreiben
>FC	Byte schreiben, Takt >D7 (IAM)	Byte schreiben, Takt >D7	Byte schreiben
>FD	Nicht erlaubt !	Byte schreiben, Takt >FF	Byte schreiben
>FE	Byte schreiben, Takt >C7 (IDAM) p	Byte schreiben, Takt >C7, p	Byte schreiben
>FF	Byte schreiben, Takt >FF	Byte schreiben, Takt >FF	Byte schreiben

'p' als Index gibt an, daß der CRC-Generator ab dem folgenden Byte mit der Berechnung der Prüfsumme beginnt. Zu jedem Zeitpunkt kann die seit Übergabe dieses Bytes berechnete Prüfsumme durch Übergabe von >F7 geschrieben werden. Beim Aufbau des Spurbildes (vor dem Schreiben!) muß daher auf korrekten Abstand zwischen Start des CRC-Generators und dessen Auswertung geachtet werden. Nach Auslösen der 2 CRC-Bytes muß der CRC-Generator neu gestartet werden.

'DAM' DATA-ADDRESS-MARK.

Dieses Byte ist das erste Byte eines Sektors und kennzeichnet den Beginn einer Datenzone. Es wird beim Lesen eines Sektors nicht mit den Sektordaten übergeben.

'IAM' INDEX-ADDRESS-MARK.

Bei den auf TI-Disketten anzutreffenden Spurstrukturen findet dieses Byte keine Anwendung. Es kann jedoch vor dem ersten Sektor einer Spur (genau: vor dessen AF) eingefügt werden, um die Synchronisation beim Spur-Lesen zu verbessern.

'IDAM' ID-ADDRESS-MARK,

Ist das Kennbyte eines AF.

Typ IV Kommandos

Genau genommen handelt es sich hier nicht um mehrere Kommandos, sondern um nur eines, das jedoch 4 Optionen besitzt.

Name	Bit-Nr. im Befehlswort								Funktion
	7	6	5	4	3	2	1	0	
Force Interrupt	1	1	0	1	i1	i2	i3	i4	Interrupt auslösen

Mit diesem Befehl kann ein momentan in Arbeit befindlicher Befehl abgebrochen und der FDC in einen definierten Zustand gebracht werden.

Dieser Befehl sollte angewandt werden, wenn entweder eine Power-Up-Initialisierung durchgeführt wird oder wenn der Controller mit einem falschen Befehl durcheinander gebracht wurde.

Die Spezifikationsbits i1 bis i4 erlauben es, nach Eintreten bestimmter Bedingungen einen Prozessor-Interrupt auszulösen. Von dieser Möglichkeit wird beim TI kein Gebrauch gemacht (auch jeder andere Befehl an den FDC löst nach Beendigung einen Interrupt aus).

Die Spezifikation der Interrupt-Bedingung ist wie folgt möglich:

Bit Funktion

- i1 alle : Sofortiger Interrupt
- i2 alle : Interrupt nach nächstem Index-Puls
- i3 1771/73: Interrupt nach Betriebsbereitschaft des Laufwerks (READY wechselt von Low nach High). 1772 : nicht ausgewertet.
- i4 1771/73: Interrupt mit Ende der Betriebsbereitschaft des LW (READY wechselt von High nach Low). 1772 : nicht ausgewertet.

Ist kein Bit gesetzt, so wird kein Interrupt ausgelöst, der Befehl jedoch ausgeführt.

Damit sind alle Befehle der verschiedenen FDC in Disk-Controllern für den TI-99/4A beschrieben. In der Software zum Original-TI-Controller wird nicht von allen Möglichkeiten Gebrauch gemacht, die der 1771 den anderen voraus hat.

Das nächste Kapitel zeigt anhand von Beispielprogrammen, wie der Programmierer den FDC mit den beschriebenen Befehlen dazu veranlassen kann, bestimmte Dinge zu tun.

Im Anschluß daran wird mit Auszügen aus den FDC-relevanten Passagen verschiedener DSR-ROMs von Diskcontrollern illustriert, wie die Funktionen in vorhandener Software realisiert sind.

Ansteuerung des FDC

Dieses Kapitel ist gewissermaßen eine Einstimmung auf Kapitel 18ff, in denen sich ausführlich kommentierte ROM-Listings zu den vier gängigsten Disk-Controllern finden.

Vorbereitende Arbeiten

Wie bereits im Kapitel 10 anhand des Prozessor-Interface angesprochen, ist der FDC nur bedingt in der Lage, die angeschlossenen Laufwerke zu steuern - er kann immer nur das bedienen, welches ihm die CPU zuteilte. Die CPU muß also dafür sorgen, daß das korrekte Laufwerk angeschaltet wurde. Auch die Seite (der Kopf) des Laufwerkes muß korrekt vorgewählt werden, da die eingesetzten FDC-Typen diese Leitung nicht verwalten (es gibt welche, die das können, z.B. 2795-02 und 2797-02). Zudem muß bei den 1772/73-Typen die Einhaltung der Head-Load Beruhigungszeiten per Software sichergestellt werden und bei allen außer 1771 die notwendige Codierungsart (FM oder MFM, Single oder Double Density) korrekt eingestellt sein.

Erst wenn diese Optionen von Seiten der CPU aktiviert wurden, kann dem FDC der erste Befehl gegeben werden.

Allgemeiner Ablauf der Ansteuerung

Betriebsbereitschaft des Laufwerks

Bevor ein Befehl an den FDC übergeben wird, muß das angewählte Laufwerk betriebsbereit sein, d.h. die Nenndrehzahl des Antriebs muß erreicht sein. Der 1772 kann diese Aufgabe übernehmen, 1771 und 1773 benötigen eine entsprechende Programmierung. Dem Laufwerksmotor muß ca. 1 Sekunde Zeit gegeben werden, auf die Nenndrehzahl zu kommen.

Daher muß vor Befehlsübergabe das READY-Bit des Statusregisters getestet und ggfs. eine Warteschleife ausgeführt werden.

Beispielprogramm:

```
DEMO1 LI   R12,>1100   CRU-Basis Diskcontroller
      SBO  0           Karte an und FDC-Register einblenden
      SBO  4           z.B. Laufwerk 1 anschalten
      SBZ  1           Laufwerkmotoren anwerfen bzw. bereits
      SBO  1           laufende 'bei Laune' halten
      MOVB @>5FF0,R0   Statusregister lesen
      JLT  READY       Bei gesetztem höchsten Bit ist das LW
*                      bereits auf Nenndrehzahl
      LI   R0,DELAY    Warteschleife für ca. 1 Sekunde
LOOP1 SWPB R0          Dummy-Befehl
      SWPB R0          macht gar nichts
      DEC  R0          Wert reduzieren
      JNE  LOOP1       bis auf Null
READY MOVB @CMD,@>5FF8 Kommando ins Kommandoregister
      SWPB R0          Wartezeit vor nächstem Zugriff auf FDC
      SWPB R0          dto.
      SBO  3           Head-Load aktivieren
      RT               Ende
```

ROM-Listings der gängigen Diskcontrollerkarten

Vorbemerkungen

Mit den auf den folgenden Seiten abgedruckten kommentierten ROM-Listings wird die Behandlung des Themas 'Floppy-Disk-Ansteuerung' abgeschlossen. Wie bereits zuvor angemerkt, sind die Listings nicht komplett, sondern umfassen nur die Teile, die mit dem FDC direkt oder indirekt zusammenhängen. Der interessierte Leser kann die kompletten Listings beim Autor erhalten.

Zu diesen Listings sei hier nur folgendes als wichtige Anmerkung festgehalten:

Obwohl in Kapitel 3 auf Seite 1 von einer genormten Softwareschnittstelle bezüglich der Disk-DSR gesprochen wurde, existiert diese Normung de facto nicht. TI hat es seinerzeit bei einer eher globalen Definition des nutzbaren PAD-Workspace durch eine DSR bewenden lassen (nachzulesen in den holländischen Schemablaaden) und mit der Erstellung der TI-Diskcontroller DSR quasi einen Standard geschaffen, der später lediglich interpretiert wurde. So ist verständlich, daß in Ermangelung einer schriftlichen Vorschrift für die Nutzung des VDP-RAMs, durch Diskcontroller anderer Hersteller fast zwangsläufig Inkompatibilitäten entstanden. Schließlich war das TI-Urmuster nur für 3 Laufwerke und lediglich auf Single-Density ausgelegt (obwohl man die Diskkapazität recht ordentlich dimensionierte). Doch muß man TI in jedem Fall den Vorwurf machen, das DSR-Konzept für den Diskcontroller nicht zu Ende gedacht zu haben (damals war die CES daran schuld, welche die Soft- und Hardwerker in Druck brachte). So ist es zwar möglich, prinzipiell beliebig viele Diskcontroller im System zu haben (allerdings sollte man dann nie CALL FILES ausführen!), es ist aber in keinem Fall feststellbar (da nicht vorgesehen), herauszubekommen, von wo (CRU-Basis des Controllers) ein Programm geladen wurde.

Neuigkeitswert der ROM-Listings

Das TI-ROM-Listing wurde im Zusammenhang mit der Entwicklung eines softwareverträglichen 80-Track-DOS Anfang 1987 erstellt. Mit Erscheinen des Technical Drive Book von Monty Schmitt ist eine zweite 'Auslegung' des DOS verfügbar. Wie bei allen ROM-Listings, so ist insbesondere dann, wenn mit Sachkenntnis vorgegangen wurde, das Ergebnis in der Regel gleich bis identisch. Das liegt in der Natur der Sache und ist nicht zu ändern. Gleichwohl kann daraus nicht abgeleitet werden, es sei abgeschrieben worden!

Das Atronic-ROM-Listing entstand um die Jahreswende 1985/86, und diente dem Autor als Vorlage zur Erstellung der anderen Ausführungen. Dieses DOS zeichnet sich durch die klare Programmierung aus, der zwar der letzte Schliff fehlt, die jedoch den Neuling am wenigsten verwirren wird. Nach derzeitigem Wissensstand ist es das einzige ROM-Listing zu diesem Controller, der über lange Jahre seine enorme Zuverlässigkeit und Softwareverträglichkeit bewiesen hat.

Das CorComp-Listing bezieht sich auf die zweite Version der DSR-ROMs von Millers Graphics, also die zweite mit eigenem Titelbild. Durch die Vielzahl der Optionen mag der sofortige Überblick bisweilen auf sich warten lassen, doch wird gezeigt, was man mit relativ wenig Aufwand im Atronic-Controller noch hätte realisieren können.

Das ROM-Listing zum Myarc DDCC-1 (alt) schließlich ist, nach Ansicht des Autors, das Paradebeispiel dafür, wie man es NICHT machen sollte. Speicherintensive Verwaltung fast nutzloser Daten, die Verwendung redundanter Flags sowie die 'Fast-Schon-Endlosschleife' bei einem Lost-Data-Zustand beim Sektor-I/O (mit welcher der Autor manche schlechte Erfahrung machen mußte) und der fast chaotisch zu nennende Umgang mit dem VDP-RAM sollten der Erheiterung, keinesfalls aber als Vorbild dienen.

Nur das TI- und das Atronic-Listing stellen den letzten Stand der Softwareentwicklung des jeweiligen Herstellers dar (in beiden Fällen gestoppt durch Produktionsschluß), zu den beiden anderen Typen gibt es mittlerweile z.T. mehrfach revidierte DOS-Versionen.

Der Leser, der über ein CPS99 oder einen neueren Controller aus USA verfügt, sollte mit den hier verfügbaren Hilfestellungen durch die kommentierten alten Versionen in der Lage sein, selbst ein Listing der neuen Generation zu erstellen. Es muß jedoch angemerkt werden, daß die Unterschiede seltener den FDC-Bereich als Dinge wie Hilfsroutinen u.ä. betreffen.

Hardwarebedingte Unterschiede

Die DD-Systeme (Double Density) steuern ihren jeweiligen FDC im 'Klartext' an, der TI-Controller verwendet einen FDC mit invertiertem Datenbus (siehe auch Kapitel 16). Beim Vergleich der FDC-Befehle im Listing mit denen in Kapitel 14 ist auf diesen Umstand zu achten.

Durch den bei DD auftretenden hohen Datendurchsatz ist es nicht mehr möglich, die Ansteuerung des FDC beim Datentransfer aus dem Wait-State-ROM des Diskcontrollers auszuführen. Die DD-Systeme kopieren

daher die entsprechenden Routinen kurz vorher ins PAD-RAM, das sie zuvor im eigenen System-RAM sicherten. Das hierzu benutzte System-RAM dient daneben der Verwaltung der letzten Spur eines jeden Laufwerks, sowie der Speicherung anderer Informationen. Besonders intensiv wird das System-RAM vom Myarc-Controller benutzt, äußerst sparsam geht man bei Atronic damit um.

Alle ROM-Listings der DD-Systeme entstammen einem Speicherbereich, der nicht über das normale DSRLNK zugeschaltet wird. Alle 3 verfügen im Gegensatz zum TI-Controller über ein ROM, das in zwei Bänke aufgespalten ist. Die zweite Bank wird über ein CRU-Bit an- und abgeschaltet. Bei Myarc bleibt dabei das RAM an der gleichen Stelle, wohingegen bei Atronic und CorComp das RAM erst in der zweiten Bank verfügbar ist.

Identifikation der Controllerkarten

Für bestimmte Anwenderprogramme kann es recht nützlich (bisweilen lebenswichtig, z.B. Kopierschutz) sein, den genauen Typ eines Diskcontrollers zu ermitteln. Die Erkennung des FDC-Chips wird im entsprechenden Kapitel behandelt. Zur Kartenerkennung gibt es folgende Methoden:

Identifikation des DSR-ROMs

Das ist sicher die schlechteste aller Methoden. Hier wird ein indirektes Verfahren angewandt, wobei aus einer Soft- auf eine Hardware geschlossen wird. Die Abfrage eines bestimmten Bytes, egal welches es nun ist, funktioniert nur dann, wenn bei deren Programmierung sicher ist, daß sich ALLE Controller in diesem Byte unterscheiden und alle zukünftigen dies auch tun werden. Somit dürfte klar sein, daß es so nicht geht.

Eine Prüfsumme ist auch nicht besser, da eine leicht modifizierte DSR (Offset des ersten Datensektors, geänderte Retry-Zahl etc.) ja nicht bedeutet, man habe einen anderen FDC im System.

Analyse des CRU-Mappings

Das klappt nur auf Systemen, welche über Input-Bits verfügen. Man kann zwar erkennen, ob das Paging-Bit (>B oder >3) das ROM umblendet, aber auch wieder nur über Bytevergleiche oder Prüfsummen, zudem kann man dann nur TI oder Nicht-TI ermitteln.

Zu den Inputs: Der TI-Controller erkennt damit die Präsenz eines jeden Laufwerks, CorComp und Myarc steuern damit (und über DIP-Schalter auf der Karte) die Step-Zeiten der Laufwerke und der Atronic hat keine Inputs. Da aber nicht eindeutig erkannt werden kann, ob z.B. alle DIPs ON sind, also NULL liefern oder ob gar kein Input da ist, fällt auch diese Methode aus.

Analyse des System RAMs

Ist schon besser, bzw. genau das, worauf es hinauslaufen sollte. Gibt es nach Umschalten von CRU-Bit >B immer noch kein RAM, ist's ein TI-Controller, liegt es vor und nachher auf >5000 als 8-Bit-RAM vor, so handelt es sich höchstwahrscheinlich um einen Myarc. Liegt 4-Bit-RAM ab >4000 vor, so ist es wahrscheinlich ein CorComp, sonst ein Atronic.

Doch auch diese Methode scheitert in bestimmten Situationen. Im Kapitel 15 wurde bei der Besprechung der FDC-Programmierung ein Verfahren gezeigt, welches in Zusammenhang mit den hier genannten Hardwareanalysen das fast 100%ige Erkennen des FDC-Typs und damit des ControllerBoards erlauben.

Unterschiede in der Software

Die Unterschiede zwischen den 3 ersten Systemen (TI, Atronic, CorComp) sind recht gering. Insbesondere beim Vergleich Atronic-CorComp fallen fast identische Sequenzen auf. Wer über die kompletten ROM-Listings, also auch des Level-2-Bereiches verfügt, wird noch mehr Gemeinsamkeiten finden. Lediglich Myarc ist auch hier die (unrühmliche) Ausnahme.

Dennoch gibt es funktionelle Unterschiede.

Die Formatier-Routine

TI und Atronic formatieren in einfacher Dichte mit einem sog. 'Running-Interlace' oder auch 'Track-Skew'. Dabei werden nicht alle Spuren identisch formatiert (immer mit dem gleichen Sektor beginnend), sondern es wird ein Versatz der Sektornummern von Spur zu Spur erzeugt, der eine Verzögerung in der Größenordnung der Step- und Settle-Time bewirkt. Dies ermöglicht ein fließendes Lesen und Schreiben auch wenn Schrittimpulse dazwischenliegen.

Myarc und CorComp formatieren starr, jede Spur gleich. Jedoch erlaubt CorComp durch Setzen eines Bits im System-RAM, daß das Sektor-Interlace vom Anwender definiert wird. Damit wird zwar immer noch jede Spur gleich formatiert, jedoch kann das Interlace im Einzelfall die Zugriffszeiten optimieren. Myarc formatiert

nur mit 16 Sektoren pro Spur bei doppelter Dichte, was wohl einer zu starken Anlehnung an die Western Digital Datenblätter zuzurechnen ist.

Atronic und CorComp formatieren gewissermaßen blind - der Anwender muß selbst prüfen, ob die Formatierung gelang. TI formatiert bei zweiseitigen Disketten zwar auch blind, prüft jedoch (noch in der Level 1 Routine) bei zweiseitiger Formatierung, ob die zweite Seite korrekt formatiert wurde. Myarc prüft sofort nach der ersten doppelseitig formatierten Spur, ob die zweite Seite korrekt ist und schaltet gegebenenfalls selbst auf einseitige Formatierung um.

Auch die Anzahl zu formatierender Spuren wird unterschiedlich interpretiert. TI und Atronic halten sich genau an die Vorgaben über Seiten- und Spuranzahl. Myarc erlaubt nur exakt 40 Spuren, alle anderen Werte erzeugen eine Fehlermeldung.

Besondere Eigenmächtigkeit in Bezug auf die Vorgaben zeigt der CorComp-Controller. Seine Software prüft, ob mehr als 40 Spuren zu formatieren sind. Wenn nicht, dann werden soviele Spuren und Seiten formatiert, wie es beim Aufruf der Routine spezifiziert wurde. Andernfalls wird die Spuranzahl halbiert und auf zweiseitige Formatierung übergegangen. Damit werden Variationsmöglichkeiten für eigene Software verschenkt.

Die Verify-Routine

Bei TI, Atronic und CorComp wird eine Byte-Für-Byte Verify-Routine verwendet. Hierbei wird bei einem Sektor-Schreib-Befehl neben der eigentlichen Schreibroutine eine Vergleichsroutine ins PAD-RAM kopiert, die unmittelbar nach dem Schreiben aufgerufen wird. Je nach Ergebnis dieses 'On-Line' Datenvergleichs wird der Schreibversuch wiederholt oder aber die Routine beendet. Dabei wird der Verify-Vorgang wie ein Sektor-Lese-Befehl behandelt.

Ganz anders, und eigentlich recht trickreich, ist dies beim Myarc-DOS realisiert. Nachdem der Sektor geschrieben wurde, wird ein modifizierter Sektor-Lese-Zyklus gestartet, bei dem die Daten von Diskette quasi 'ins Leere', nämlich ins Konsolen-ROM geschrieben werden. Dieser Adreßbereich wird benutzt, da auf ihn ohne Wait-States zugegriffen werden kann, und zudem wegen des ROMs nichts zerstört werden kann. Es wird also nicht explizit verglichen, ob die Daten korrekt von der Diskette kommen. Soweit CRC und Lost-Data Fehlertypen betroffen sind, wird diese Aufgabe bereits vom FDC übernommen. Auch wurden eventuelle Lost-Data-Zustände beim Schreiben ggfs. bereits dort gemeldet. Soweit mag also diese Art des Datentests ausreichend erscheinen. Doch eine Fehlerart, die auf den ersten Blick konstruiert aussieht, wird mit diesem abgemagerten Verify nicht erkannt: die Verfälschung von geschriebenen Daten, die keinen CRC-Fehler erzeugen. Dies kann bei der Übergabe von Daten an den FDC passieren, und ist nur mit endlicher Wahrscheinlichkeit auszuschließen. Im Endeffekt also wieder ein Punktgewinn für die Konkurrenz.

Die Dichte-Erkennung

Alle DD-Systeme müssen erkennen, in welcher Codierungsart die Daten auf der jeweiligen Diskette vorliegen. Dies ist nur durch Probieren möglich. Das Verfahren ist dabei immer gleich: Es wird mit einer willkürlich festgelegten Dichte versucht, auf die Daten der Diskette zuzugreifen. Verlief der Versuch erfolgreich, so bleibt nichts weiter zu tun.

Wurde jedoch ein Lost-Data, CRC oder RNF-Fehler erkannt, dann wird der Versuch mit einer anderen Dichte wiederholt. Führte dieser neue Versuch zum Erfolg, dann wird diese Dichte für das betreffende Laufwerk notiert und beim nächsten Zugriff auf dieses Laufwerk gleich mit der korrekten Dichte begonnen.

Bei Diskettenwechsel auf eine andere Dichte arbeitet dieser Algorithmus gleichermaßen.

Wann nun auf eine andere Dichte umgeschaltet wird, ist verschieden. Atronic schaltet nach 5 Fehlversuchen auf eine andere Dichte um (danach laufend), und bricht nach insgesamt 10 erfolglosen Versuchen ab. CorComp erlaubt ebenfalls insgesamt 10 Versuche, wechselt jedoch nach jedem Fehlversuch die Dichte.

Myarc schließlich wechselt ebenfalls nach einem Fehlversuch die Dichte, benutzt jedoch eine Adreßfeld-Leseroutine, um die wirkliche Dichte zu ermitteln. Durch dieses direkte Verfahren sah man sich in der Lage, nur 5 Retries zulassen zu müssen.

Gegen das Atronic- und Myarc-Verfahren ist prinzipiell nichts einzuwenden. Lediglich die laufende Umschalterei bei CorComp läßt Probleme erwarten, die eine defekte Diskette auslösen könnte. In praxi arbeiten alle 3 DD-Systeme in diesem Punkt weitgehend korrekt, CorComp weist tatsächlich gelegentlich Fehler der beschriebenen Art auf. Die weiteren Details können dem jeweiligen ROM-Listing entnommen werden.

Indizierte Adressierung

Die DOS-Routinen bei TI und Atronic sind mit jedem Wert des Workspace-Pointers lauffähig. Lediglich in R15 muß sich die Adresse VDPWA (>8C02) befinden, und die Parameterzeiger müssen relativ zum WP

gleich liegen. Zwar hat CorComp das auch versucht, doch klappt so etwas nur, wenn es im ganzen Programm konsequent durchgehalten wird. Genau das ist aber bei CorComp nicht der Fall (siehe Adressen >4AF0 und >5682). In diesen Befehlen wird absolut auf Adressen zugegriffen, in denen sich bestimmte Registerinhalte des aktuellen WS befinden.

Bei allen dreien befindet sich in Register 9 der Wert >8300, die Standard-Basis des PAD-RAMs. Bei Myarc wird hierzu Register 4 verwendet, welches >83E0, also den echten Workspace-Pointer enthält.

Dadurch werden die Pointer aber nur mittels negativem Offset erreicht, was recht undurchsichtig wirkt. Allerdings wird von R4 nur auf Level 2 richtiger Gebrauch gemacht, sonst wird absolut adressiert. Hier muß also GPLWS geladen sein.

Hinweise zur Lesart der ROM-Listings

Da es sich nur um Auszüge handelt, wurden einige Passagen aus Platz-

gründen gestrichen. An diesen Stellen findet sich das Symbol

```
*
*  . . .
*
```

was in Anlehnung an die Notation des Assemblers eine Unterbrechung im Source-Code angibt. An einer solchen Stelle fehlen Adreßbereiche, die dem Rotstift zum Opfer fielen.

Ansonsten wären die Listings nach korrektem Abtippen und anpassen der Labels direkt assemblierbar, wodurch die Erstellung eines eigenen DOS möglich wird.

Besonderheiten im TI-ROM-Listing

SEEK-Befehl und Verifikation der Seite

Da der 1771-FDC keinen Seitenvergleich beim Anfahren einer Spur erlaubt, muß mit anderen Methoden ermittelt werden, ob die korrekte Spur erreicht und der richtige Kopf des Laufwerks geschaltet wurde.

Daher wird nach dem theoretischen Erreichen der Spur, deren Nummer über den sehr wohl möglichen Verify-On-Track geprüft. Wenn diese Nummer korrekt war, wird auf der gewählten Seite ein beliebiges Adreßfeld gelesen und dessen Seitenkennung geprüft. Es handelt sich also um das gleiche Verfahren, das die moderneren FDC (1773) beim Sektorzugriff selbst anwenden.

Invertierte FDC-Kommandos

Durch den invertierten Datenbus des 1771 liegen die Kommandos bereits invertiert vor. Dabei ist bei den Typ I Kommandos zu beachten, daß die Step-Raten für jeden Befehl (RESTORE, SEEK, STEP IN) fest im Befehl eingebaut wurden. Soll der TI-Controller also kürzere Step-Zeiten 'verbraten' bekommen, so müssen alle Typ I Kommandos geändert werden!

Softwareänderungen

Nur per Software ist der 1771 nicht dazu zu bringen, in doppelter Dichte zu arbeiten. Man kann jedoch durch geringfügige Änderungen eine Möglichkeit schaffen, 80 Spur Laufwerke anzusteuern, die es dann bei 2 mal 80 Spuren in Single Density erlauben, die gleiche Menge an Daten auf einer Diskette zu halten, wie es bei 2 mal 40 Spuren in doppelter Dichte möglich ist, nämlich 360 KByte. Hierzu müssen lediglich die Vergleichszahlen am Anfang der Sektor-I/O-Routine entsprechend erhöht werden. Erfolgt noch eine Anpassung der Step-Raten auf die normalerweise sehr schnellen 80 Spur Laufwerke, so wird der Geschwindigkeitsunterschied zu Double Density auf das theoretische Minimum (Faktor 1.8 ohne Step-Zeiten) reduziert. Ein solches 'Einfach DOS-80' ist zwar softwareneutral (da man nicht an die 5 undeklarierten Bytes im VDP muß, um die Spuranzahl je Laufwerk zu verwalten), jedoch können zweiseitige 40 Spur Disketten damit nicht mehr bearbeitet werden.

Besonderheiten im Atronic-ROM-Listing

Überflüssiges in der Formatierroutine

Das Atronic-DOS verwendet 2 getrennte Routinen zur Formatierung in einer der beiden möglichen Speicherdichten. Dabei wird jeweils eine Spur formatiert und aus dem gemeinsamen Segment das Step-In ausgeführt. Dabei ist die Sequenz nach Fertigstellung eines Spurinhaltes zweimal fast identisch vorhanden. Der einzige Unterschied besteht in der Spurlänge und der Dichte. Funktionell wirkt sich das nicht aus, doch ist es ein Schönheitsfehler, den sich ein versierter Programmierer nicht erlauben sollte.

Soft- und Hardwareänderungen

Das System-RAM wird im Atronic-Controller recht wenig genutzt. So könnte man Optionen wie RAM-Transfer, Verify abschalten, extern definiertes Interlace (alles wie bei CorComp) vorsehen. Denkbar, und im Grunde eigentlich gar nicht so abwegig, wäre die Option auf 42 Spuren pro Diskettenseite. Damit würden, entsprechende Laufwerke vorausgesetzt, 378 KByte pro DS/DD-Disk zur Verfügung stehen, die sogar noch mit dem Standard-DOS verwaltet werden könnten (die Sector-Bitmap erlaubt ja bekannterweise bis 400 KByte pro Disk). Oder aber, man formatiert mit 19 Sektoren pro Spur (statt 18 bei DD) und erreicht so bei 42 Spuren pro Seite 1596 Sektoren pro Disk, also 399 KByte pro Disk. Die Möglichkeiten sind vielfältig.

Auch sind die CRU-Bits 9,>C,>D,>E und >F nicht belegt. Zusammen mit dem System-RAM (als Index-Speicher) könnte man 80 Spur Laufwerke ggfs. auf 40 Spuren umschalten und dies beim Sektorzugriff in der Spur-Sektor-Berechnung auswerten.

Der mit CRU-Bit >B geschaltete Bereich des 16 KByte ROMs ist zudem in weiten Bereichen nicht programmiert, so daß hier etliche neue Routinen Platz finden könnten.

Besonderheiten im CorComp-ROM-Listing

Überflüssiges in der Formatierroutine

Das CorComp-DOS ist streckenweise sogar noch etwas umständlicher als das von Atronic. So wird nicht in einem Zug auf beiden Seiten formatiert, wenn zweiseitig gefordert war, sondern es wird erst die eine Seite formatiert, dann ein RESTORE ausgeführt, und erst dann auf der zweiten Seite gearbeitet. Nicht nur, daß das RESTORE unnötig Zeit kostet, es ist auch sonst nicht zu sehen, wieso dieses Verfahren gewählt wurde. Dazu kommt noch, daß bei CorComp sogar das Step-In nach einer Spur für SD und DD getrennt, also 2 Mal vorhanden ist. Zusammen mit solch vielsagenden Befehlen wie 'LI R0,>FFFF' gibt es noch andere Bereiche, die den Leser möglicherweise dazu veranlassen, sich erschrocken an den Kopf zu greifen.

Soft- und Hardwareänderungen

Neben der Beseitigung oben genannter Mißstände ist im Prinzip das Gleiche zu sagen wie bei Atronic. Zudem stehen durch den 9901 als CRU-Interface noch etliche CRU-Leitungen für Ein- und Ausgänge zur Verfügung. Wieso CorComp hier einen so teuren Chip anstatt 3 kleiner TTL-Latches eingesetzt hat, wird wohl von hier aus schwer zu erklären sein. Auf jeden Fall sitzt hier ein CRU-Interface, das in einem Floppy-Disk-Controller nur mit gebremstem Schaum arbeitet. Denkbar wären Drive-Sense-Eingänge wie beim TI-Controller. Damit würde der unvermeidliche I/O-ERROR 06 beim Zugriff auf nicht vorhandene Laufwerke wenigstens schneller kommen. Auch könnten direkt Index-Puls, Schreibschutz und andere Leitungen vom Laufwerk getestet werden, ohne erst umständlich den FDC mit einem nicht ausführbaren Befehl zu traktieren.

All das wäre durch ein paar mehr Leitungen auf der Platine fast zum Nulltarif möglich gewesen (der versierte Bastler kann das aber jederzeit mit ein paar Drähten nachholen!).

Irgendwie erinnert das an Texas Instruments, die ihre seinerzeit bahnbrechende 9900-CPU mit lächerlichen 256 Byte High-Speed RAM versahen und alles andere über Wait-States bremsten.

Der mit CRU-Bit >B geschaltete Bereich des 16 KByte ROMs ist, im Gegensatz zum Atronic-ROM, weitgehend mit Basic-Routinen, der Verwaltung des eigenen Titelbildes und der Manager-Laderoutine gefüllt, so daß hier kaum Platz für Eigenentwicklungen bleibt. Jedoch ist das eigene Titelbild verzichtbar, ebenso wie der Titelbild-Manager-Loader, der auch aus dem Basic aufgerufen werden kann. Hier fällt neuer Raum an, der für eigene Ideen nutzbar ist.

Besonderheiten im Myarc-ROM-Listing

Besonders undurchsichtig ist die Sektor-I/O-Routine programmiert. Durch die vielen Flags, die den Ablauf beeinflussen, ist kaum zu erkennen, was passiert. Grundsätzlich wird unterschieden zwischen Sektor lesen und schreiben, zwischen Transfer ins oder aus dem CPU-RAM und Transfer über VDP-RAM. Dabei führt ein Verify-Lauf immer ins ROM.

Hier heißt es im Zweifelsfall mehrfach lesen.

Unsauberkeiten in der Programmierung

Ein Lost-Data-Zustand, im Allgemeinen bewirkt durch das zeitkritische Polling-Verfahren, führt beim Myarc-DOS zu einer Wiederholung des Befehls ohne Reduktion der Retries. Zum einen sind ohnehin mit 5 nur sehr wenig Wiederholungschancen gegeben, zum anderen kann sich das unter Umständen zu einer Endlosschleife entwickeln. Im praktischen Betrieb konnte beobachtet werden, daß der Controller gelegentlich Pau-

sen von bis zu 30 Sekunden einlegte, in denen sich augenscheinlich nichts tat. Manchmal konnte diese Bedenkpause durch Wackeln am Jacket der Diskette unterbrochen werden. Es sollte daher bei einer Softwareänderung hier zuerst die Retry-Zahl verdoppelt und ein LostData in den Retry-Countdown aufgenommen werden.

Durch die willkürliche Verwendung des VDP-RAMs ergeben sich Unverträglichkeiten mit Software, die normgerechte VDP-Verwaltung voraussetzt.

Auf Level 2 ist hier nur mit viel Aufwand etwas zu ändern, bei Level 1 jedoch kann zumindest das Chaos im VDP-Stack, wo das PAD-RAM ab >83A2 gesichert wird, durch Beachten des Zeigers an >8366 vermieden werden.

Etwas unglücklich ist die Verwaltung mehrerer aufeinanderfolgender Zugriffe auf verschiedene Laufwerke gelöst. Alle 3 anderen Systeme merken sich die Nummer des zuletzt angesprochenen Laufwerks und können so einem neu zugeschalteten die notwendige Zeit zum Erreichen der Zugriffsbereitschaft lassen. Im Myarc-DOS wird erst bei einem Fehler durch Auslösen eines Interrupts eine kurze Pause eingelegt. Das ist nicht nur umständlich, sondern schlicht unsinnig.

Soft- und Hardwareänderungen

Im oberen ROM-Bereich, der mit CRU-Bit 3 geschaltet wird (dient gleichzeitig der Dichte-Umschaltung), ist noch ca. 1 KByte Platz. Auch das System-RAM kann, da als 8 Bit-RAM ausgeführt, extern oder für eigene Optionen wie etwa direkten RAM-Transfer genutzt werden. DasCRU-Interface ist jedoch komplett ausgelastet.

ROM-Listing TI-Controller

```
*
* Sektor Lesen/Schreiben
*
40E8  LI   R4,10           Anzahl der Versuche = 10
40EC  MOVB @>4630,@>50(R9) Fehlerbyte löschen
      BL   @>4496          Laufwerk anwählen
      BL   @>45F0          VDP-Zeiger auf Puffer der letzten Spur dieses LW
      CLR  R0
      MOVB @>FBFE(R15),R0  Letzte Spur lesen
4100  CI   R0,>D700        mit >28, dez 40 (invertiert) vergleichen
      JH   >410C           tatsächliche Spur ist kleiner - weiter
      BL   @>4524      Spurnr. ist zu groß - LW initialisieren, RESTORE
      SETO R0          Spurnummer 0 (invertiert!)
410C  MOVB R0,@>5FFA     Spurnummer ins Spurregister des FDC
      MOV  @>4A(R9),R1   Kopie der Sektornummer holen (wurde in >8350 übergeben)
      SBZ  7             Disk-Unterseite anwählen
      CLR  R7          Seitenindex
      CI   R1,720      Sektornummer über dem Maximalwert?
      JHE  >41B6       Nummer ist zu groß
      CI   R1,360      Größer als die höchste Nummer auf Seite 1?
      JL   >413E       Nein
      AI   R1,-719     Ja, Komplement bilden, Sektor ist auf der Unterseite!
      ABS  R1          Betrag
      CLR  R0          Rechenregister
      DIV  @>4632,R0   durch 9 teilen, Spur und relativen Sektor berechnen
      AI   R1,-8       Sektornummer korrigieren
      ABS  R1          Betrag
      SBO  7           Unterseite anwählen
      LI   R7,>0100    Seitenindex
      JMP  >414E       weiter
413E  CI   R1,1        Sektornummer 1?
      JH   >4148       größer
      BL   @>4524      RESTORE, Kopf auf Spur 0
4148  CLR  R0          Rechenregister 'Spur' löschen
      DIV  @>4632,R0   Sektornummer geteilt durch 9
414E  SWPB R0          errechnete Spur des gesuchten Sektors ins High-Byte
      INV  R0          auf FDC-Chip Bus anpassen (invertieren)
      BL   @>4614       VDP-Schreibadresse mit Wert in R2 setzen
      MOVB R0,@-2(R15)  Spurnummer in den Puffer der letzen Spur dieses LW
      MOVB R0,@>5FFE    und ins FDC-Datenregister (WRITE TRACK TO SEEK)
      SWPB R1          Sektornummer (relativ) ins High-Byte
      INV  R1          anpassen
      MOVB R1,@>5FFC   ins Sektorregister
      CB   R0,@>5FF2   stimmt die geforderte Spur mit der aktuellen überein?
      JEQ  >417A       Ja, Kopf nicht bewegen!
      BL   @>45CA      Nein, FDC-Kommandoroutine
      DATA >E100       SEEK TRACK
      BL   @>4482      FDC-Status holen, Befehlsende abwarten
      SLA  R0,13       SEEK ERROR (Spur nicht verifiziert)?
      JOC  >41B0       Ja, Fehler >11
417A  BL   @>45CA      Spur gefunden, neues FDC-Kommando
      DATA >3F00       READ ADDRESS, beliebiges Adressfeld lesen
      SBO  2           WAIT ENABLE BIT, erlaubt das Anhalten der CPU
4182  MOVB @>5FF6,R0   Spurnummer lesen
      LI   R6,4        4 Bytes blind lesen
      MOVB @>5FF6,R5   Seitennummer lesen
      INV  R5          korrigieren
4190  MOVB @>5FF6,R0   Sektor, Sektorlänge und CRC blind lesen, diese Bytes
      DEC  R6          werden nicht benötigt
      JNE  >4190       weiter
      BL   @>4480      Befehlsende abwarten (WAIT DISABLE)
      SLA  R0,13       RECORD NOT FOUND (kein Adressfeld gefunden)?
      JOC  >41BC       Ja, Fehler >21
      JLT  >41C2     CRC ERROR (Adressfeld defekt)! Fehler >22
      SLA  R0,2      LOST DATA (Lesefehler)?
      JOC  >41C8     Ja, Fehler >23
      CB   R7,R5     stimmt die Seitennummer mit der des AF überein?
```

```
      JEQ  >41CE     Ja, OK!
*
* Fehler-Routinen, >4590 erlaubt Wiederholung des Versuchs, >45AC nicht!
*
      BL   @>45AC    Fehler >06: - Seitenanwahl fehlgeschlagen
      DATA >0600
41B0  BL   @>4590    Fehler >11: - Seek Error, Spur nicht verifiziert
      DATA >1100
41B6  BL   @>45AC    Fehler >07: - Sektornummer zu groß
      DATA >0700
41BC  BL   @>4590    Fehler >21: - Record not found, kein Adressfeld gef.
      DATA >2100
41C2  BL   @>4590    Fehler >22: - CRC-Error, Adressfeld defekt
      DATA >2200
41C8  BL   @>4590    Fehler >23: - Lost Data, Lesefehler (schwerwiegend)
      DATA >2300
*
41CE  MOVB R1,@>5FFC     Sektornummer schreiben
      MOV  @>4E(R9),R2   VDP-Pufferadresse holen
      MOVB @>4D(R9),R0   I/O-Code lesen
      JEQ  >425C         >00, also Sektor schreiben
      BL   @>4614        VDP-Schreibadresse setzen
      BL   @>45CA        FDC-Kommando
      DATA >7700         READ SECTOR
      LI   R6,>0100      Anzahl Bytes im Sektor
      SETO R5            Index 'LESEN/VERIFY WAR FEHLERHAFT'
      SBO  2             FDC-Chip erlauben, die CPU anzuhalten
      MOVB @>4D(R9),R0   I/O-Code
      JNE  >4214         nicht Null, also lesen
*
* Sektordaten Verify
*
      CLR  R0            I/O-Code ist 0, nach Schreiben jetzt Verify
41F6  MOVB @>5FF6,R0     1 Byte aus dem FDC-Chip lesen
      AB   @>FBFE(15),R0 zum Komplement im Datenpuffer addieren (muß >FF sein!)
      CI   R0,>FF00      Richtig?
      JNE  >422E         Nein, Fehler!
      SZCB @>5FF6,R0       Nullen im >FF-Byte setzen
      SB   @>FBFE(R15),R0  Sollwert abziehen (muß >00 ergeben!)
      JNE  >422E           Nicht Null, Fehler!
      DECT R6            OK, alle durch?
      JNE  >41F6         Nein, weiter
      JMP  >422C         alle Bytes korrekt - weiter mit Löschung des Index R5!
*
* Sektorinhalt lesen
*
4214  MOVB @>5FF6,R0    ein Datenbyte von Disk holen
      INV  R0           korrigieren
      MOVB R0,@-2(R15)  ins VDP-RAM
      MOVB @>5FF6,R0    dto. nächstes Byte
      INV  R0
      MOVB R0,@-2(R15)
      DECT R6           alle durch?
      JNE  >4214        nein
*
422C  CLR  R5           Fehlerindex nach lesen
*
* Fehler-Einsprung bei Verify-Fehler
*
422E  BL   @>4480    Befehlsende abwarten
      SLA  R0,13     RECORD NOT FOUND (Sektor nicht korrekt erkannt)?
      JOC  >4244     Ja, Fehler >21
      JLT  >424A     CRC ERROR (Daten teilweise verloren)!
      MOV  R5,R5     Fehler beim Verify?
      JNE  >4256     Ja, Fehler >28
      SLA  R0,2      LOST DATA (Daten verspätet angeliefert/abgeholt)?
      JOC  >4250     Ja, Fehler >23
      B    @>4676    Operation erfolgreich, via VDP-Stack zurückspringen
*
```

```
* Fehler-Routinen, Wiederholungen sind erlaubt
*
4244  BL   @>4590    Fehler >21: - Sektor nicht korrekt erkannt
      DATA >2100
424A  BL   @>4590    Fehler >22: - CRC-Fehler im Sektor
      DATA >2200
4250  BL   @>4590    Fehler >23: - Datenverlust (schwerwiegend)
      DATA >2300
4256  BL   @>4590    Fehler >28: - Verify nicht erfolgreich
      DATA >2800
*
* Sektor schreiben
*
425C  BL   @>461E       VDP-Leseadresse mit R2 setzen
      BL   @>45CA       FDC-Kommando
      DATA >5700        WRITE SECTOR
      LI   R6,>0100     256 Byte im Sektor
      SBO  2               WAIT ENABLE
426C  MOVB @>FBFE(R15),R0  Datenbyte aus VDP lesen
      INV  R0              invertieren
      MOVB R0,@>5FFE       in FDC-Datenregister schreiben
      MOVB @>FBFE(R15),R0  nächstes Byte
      INV  R0              anpassen
      MOVB R0,@>5FFE     ins Datenregister
      DECT R6            alle Bytes?
      JNE  >426C       Nein
      BL   @>4480      Ja, Befehlsende abwarten
      SLA  R0,11       WRITE PROTECT (Disk schreibgeschützt)?
      JOC  >429A       Ja, Abbruch mit Fehler >34
      SLA  R0,2        RECORD NOT FOUND (Sektor nicht gefunden)?
      JOC  >42A0       Ja, Fehler >31
      SLA  R0,2        LOST DATA (Daten verspätet angeliefert)?
      JOC  >42A6       Ja, Fehler >33
      BL   @>461E      VDP-Leseadresse erneut setzen
      JMP  >41E0       weiter mit Verify
*
429A  BL   @>45AC    Fehler >34: - Disk ist schreibgeschützt, kein Retry
      DATA >3400
42A0  BL   @>4590    Fehler >31: - Sektor nicht gefunden
      DATA >3100
42A6  BL   @>4590    Fehler >33: - Datenverlust
      DATA >3300
*
* Diskette formatieren
*
42AC  CLR  @>4A(R9)     Index: Disk ist einseitig
      MOVB @>4C(R9),R8  Laufwerknummer
      SRL  R8,12        höchstes Nibble Null?
      JEQ  >42C4        Ja. OK
      C    R8,@>4630    >0001, also Option 1?
      JEQ  >42C4        Ja
      BL   @>45AC       Falsche Option!
      DATA >0700             Fehlercode >07
42C4  SZCB @>4638,@>4C(R9)   Laufwerknummer normieren
      CB   @>51(R9),@>4657   >02, zweiseitig?
      JNE  >42D6             Nein
      SETO @>4A(R9)          Ja, Disk ist zweiseitig zu formatieren
42D6  MOVB @>4630,@>50(R9)   Fehlerbit löschen
      BL   @>4496            Laufwerk (Nr. in >4C(R9)) zuschalten
      BL   @>4524            RESTORE, Kopf über Spur 0 positionieren
      CLR  R3                Spurnummer >00
42E6  MOV  @>4A(R9),@>4A(R9) Einseitig?
      JEQ  >42F8             Ja, Unterseite auslassen!
      SBO  7             Unterseite der Diskette anwählen
      LI   R7,>0100      Seitennummer >01
      BL   @>43AA        Spur im Puffer aufbauen und schreiben
42F8  SBZ  7             Oberseite anwählen
      CLR  R7            Seitennummer >00
      BL   @>43AA        Spur im Puffer aufbauen und schreiben
```

```
      BL   @>45CA       FDC-Kommandoroutine
      DATA >A500        STEP IN, Kopf eine Spurweite nach innen fahren
      BL   @>4482       Befehlsende abwarten
      AI   R3,>0100     Spur +1
      CB   R3,@>4D(R9)       alle geforderten Spuren?
      JNE  >42E6             Nein
      MOV  @>4A(R9),@>4A(R9) Einseitig?
      JEQ  >437A             Ja
      SBO  7             Nein, auf Unterseite schalten
*
* Prüfen, ob Unterseite korrekt formatiert wurde
*
      LI   R4,10        10 Versuche
      BL   @>4524       RESTORE, Kopf über Spur 0 bringen
      MOV  @>4E(R9),R2  Pufferadresse holen
      BL   @>4614       VDP-Schreibadresse mit R2 setzen
      BL   @>45CA       FDC-Kommandoroutine
      DATA >3F00        READ ADDRESS, Adressfeld lesen
      LI   R6,6         6 Byte lesen
      SBO  2            WAIT ENABLE
433A  MOVB @>5FF6,R0    1 Byte lesen
      INV  R0           invertieren
      MOVB R0,@-2(R15)  ins VDP-RAM
      MOVB @>5FF6,0     dto. nächstes Byte
      INV  R0
      MOVB R0,@-2(R15)
      DECT R6           fertig?
      JNE  >433A        Nein
      BL   @>4480       Befehlsende abwarten
      SLA  R0,13        RECORD NOT FOUND (kein Adressfeld da)?
      JOC  >4398        Ja, Fehler >21
      JLT  >439E        CRC ERROR (Adressfeld defekt)! Fehler >22
      SLA  R0,2         LOST DATA (Datenverlust)?
      JOC  >43A4        Ja, Fehler >23
      MOV  @>4E(R9),R2  Pufferadresse holen
      INC  R2           +1
      BL   @>461E       als Leseadresse setzen
      CLR  R0
      MOVB @>FBFE(R15),R0   Seitennummer lesen
      JEQ  >437A            Null, Oberseite!
      MOVB @>4D(R9),R0   Spuranzahl
      SLA  R0,1          mal 2 (doppelseitig)
      JMP  >4384            weiter
437A  MOVB @>4631,@>51(R9)  >01
      MOVB @>4D(R9),R0      Spuranzahl
4384  SRL  R0,8          ins High-Byte
      MPY  @>4632,R0     mal Sektoren pro Spur (9) ergibt Zahl aller Sektoren
      MOV  R1,@>4A(R9)      übergeben
      MOVB @>4633,@>4D(R9)  9 Sektoren pro Spur
      B    @>4676           Rücksprung via VDP-Stack
4398  BL   @>4590  Fehler >21: - Kein Adressfeld gefunden, Record not found
      DATA >2101   Code mit Index 'Formatieren'
439E  BL   @>4590  Fehler >22: - Adressfeld defekt, CRC-Fehler
      DATA >2201   Code mit Index 'Formatieren'
43A4  BL   @>4590  Fehler >23: - Datenverlust, Lost Data
      DATA >2301   Code mit Index 'Formatieren'
*
* Spurbild im Puffer aufbauen und Spur schreiben
*
43AA  MOV  R11,R8        Rückkehr sichern
      MOV  @>4E(R9),R2   Pufferadresse
      BL   @>4614        als Schreibadresse setzen
      LI   R6,22         Track-Header 22 Nullen, GAP 1
      CLR  R2            Sektorzähler
      JMP  >43C0
43BC  LI   R6,6            ADDRESS GAP
43C0  MOVB @>4630,@-2(R15)
      DEC  R6
      JNE  >43C0
```

```
      MOVB @>4639,@-2(R15) >FE, Adress-Mark-ID
      NOP                  VDP nicht überfahren
      MOVB R3,@-2(R15)   Spurnummer schreiben
      NOP
      MOVB R7,@-2(R15)   Seitennummer
      MOVB R3,R0         Spurnummer kopieren
      SRL  R0,8          ins Low-Byte
      SWPB R7            Seitennr. ins Low-Byte
      MPY  @>4635(R7),R0 je nach Seite multiplizieren
      SWPB R7            Seitennr. korrigieren
      A    R2,R1         Sektornummer addieren
      DIV  @>4632,R0             durch 9 teilen
      MOVB @>464F(R1),@-2(R15)  Sektornummer aus Interlace-Tabelle
      LI   R6,>FFEC              negativer Offset der Tafel
43F8  MOVB @>464E(R6),@-2(R15)  AF fertig und ID-GAP (GAP 2) schreiben
      INC  R6                   alle Bytes des GAPs?
      JNE  >43F8         Nein, GAP komplettieren
      LI   R0,>E5E5      Formatierungsdaten
      BL   @>4474        VDP-Fill
      DATA >0100             256 Bytes
      MOVB @>464E,@-2(R15)   >F7, CRC auslösen
      SETO R0                >FF schreiben, GAP 3
      BL   @>4474        Füllen
      DATA >002D         45 Mal
      INC  R2            Sektorzähler+1
      CI   R2,9          fertig?
      JNE  >43BC         nein
      BL   @>4474        ja, Spur-Postambel, GAP 4
      DATA >00E7         füllen
      LI   R4,3          3 Versuche pro Spur
442C  MOVB @>4E(R9),R2   Pufferadresse holen
      BL   @>461E        als Leseadresse setzen
      BL   @>45CA        FDC-Kommandoroutine
      DATA >0B00         WRITE TRACK
      LI   R6,>0CA3      Anzahl Bytes einer Spur
      SBO  2               WAIT ENABLE
4440  MOVB @>FBFE(R15),R0  aus VDP-RAM lesen
      INV  R0              invertieren
      MOVB R0,@>5FFE       ins Datenregister
      MOVB @>FBFE(R15),R0  nochmal
      INV  R0
      MOVB R0,@>5FFE
      DECT 6             Zaehler minus 2
      JGT  >4440         noch nicht fertig
      BL   @>4480    Befehlsende abwarten
      SLA  R0,11     WRITE PROTECT?
      JNC  >4464     Nein
      B    @>429A    Ja, Fehlersequenz des Sektor-I/O-Segments verwenden
4464  SLA  R0,4      LOST DATA?
      JNC  >4472     Nein
      DEC  R4        Ja, noch ein Versuch erlaubt?
      JNE  >442C     Ja
      BL   @>45AC    Nein, Fehler >33
      DATA >3300
4472  B    *R8       Eine Spur ist komplett, Ende
*
* Füllroutine
*
4474  MOV  *R11+,R6     Anzahl holen
4476  MOVB R0,@-2(R15)  R0 wiederholt ins VDP-RAM
      DEC  6            fertig?
      JNE  >4476        nein
      RT                Ja, Ende
*
4480  SBZ  2           Wait disable
4482  MOVB @>5FF0,R0   FDC-Statusregister lesen
      INV  R0          invertieren
      JLT  >4490       High-Bit gesetzt, DRIVE NOT READY, Index-Puls n.i.O.
      SRC  R0,9        Befehl noch nicht beendet?
```

```
      JOC  >4482       Ja, warten
      RT               Befehl beendet, Ende
*
4490  BL   @>45AC      Fehler >06: - Hardware-Fehler
      DATA >0600
*
* Gefordertes Laufwerk zuschalten
*
4496  MOV  R11,R7
      MOV  @>58(R9),R2  Start des File-Blocks im VDP-RAM
      AI   R2,-10       10 abziehen
      BL   @>461E          R2 als VDP-Leseadresse setzen
      MOVB @>FBFE(R15),R0  Nr. des zuletzt benutzten Laufwerks lesen
      CLR  R5              LW-Index
      CB   R0,@>4C(R9)  gleiches LW wie zuvor?
      JEQ  >44B2        Ja
      SETO R5           Nein, Index setzen
44B2  CLR  R0
      MOVB @>4C(R9),R0  Laufwerknummer
      JEQ  >451E        Null - Fehler!
```

ROM-Listing CorComp-Controller

```
*
* System-RAM 2114  Da es sich um ein 4-Bit-RAM handelt, ist je Byte nur das
*                  High-Nibble dekodiert, das Low-Nibble ist immer 0!
*
4000   BYTE 0   Puffer für zuletzt angesprochenes Laufwerk
*
4001   BYTE 0   PAD-RAM Save- und Verify-Index.
*               >10 Kein Verify
*               >20 Leseroutinen im PAD-RAM
*
4002   DATA 0   Letzte Spur auf Laufwerk 1
4004   DATA 0   Letzte Spur auf Laufwerk 2
4006   DATA 0   Letzte Spur auf Laufwerk 3
4008   DATA 0   Letzte Spur auf Laufwerk 4
*
400A   BYTE 0  Dichtemaske, wird im Low-Nibble HBy R5 verwaltet
*              Jedes gesetzte Bit bedeutet Single Density
*
400B   BYTE 0  I/O-Optionsbits
*              >10   Formatieren mit externem Interlace
*              >20   Transfer FDC <-> RAM
*              >40   Systemaufruf (CALL MANAGER), kein normaler Rücksprung
*
* Sicherungszone für das PAD-RAM
*
400C   DATA 0,0,0,0,0,0,0,0,0,0,0,0,0
*
*...
*
* VDP-Adresse (R0) setzen
*
4390   ORI  R0,>4000     Schreibadresse
4394   SWPB R0           Leseadresse
       MOVB R0,*R15
       SWPB R0
       MOVB R0,*R15
       ANDI R0,>3FFF     Registerinhalt korrigieren
       RT
*
* R1 (Wort) ins VDP-RAM schreiben
*
43A2   SWPB R1
       MOVB R1,@>8C00
       SWPB R1
       MOVB R1,@>8C00
       RT
*
*...
*
43CC   TEXT '(C) 1984 BY Millers Graphics'
*
43E8   DATA >0100,>0200,>0400,>0800  Test-Bitmuster für Dichte je Laufwerk
43F0   DATA >1000,>2000,>4000,>8000
43F8   DATA >0001,>0002,>0004,>0008
4400   DATA >0010,>0020,>0040,>0080
4408   DATA >0020,>0800              Bitmuster zur Transferrichtung
*
440C   DATA >0009  Sektoren pro Spur SD
440E   DATA >0012  Sektoren pro Spur DD

*
4410   DATA >0001,>0000
4414   DATA >04FC,>FFFE,>F74E,>01F7,>FFFF,>FFFF
4420   DATA >FFFF,>FFFF,>FFFF,>FF00,>0000,>0000
442C   DATA >00FB
*
```

```
* Sektor-Interlace Single Density (IL=3)
*
442E   BYTE >00,>07,>05,>03,>01,>08,>06,>04,>02
       EVEN
*
* ID-GAP Double Density
*
4438   BYTE 0,0,0,0,0,0,0,0,0,0,0
4443   BYTE >F5,>F5,>F5,>FB
*
* Sektor-Interlace Double-Density (IL=4)
*
4447   BYTE >00,>0B,>04,>0F,>08,>01,>0C,>05,>10
       BYTE >09,>02,>0D,>06,>11,>0A,>03,>0E,>07
       EVEN
*
* ROUTINE >10  Sektor lesen/schreiben
*
445A   MOV  R11,@>48(R9) R11 nach >8348
       BL   @>4618       PAD-RAM sichern, I/O-Routinen kopieren
       MOV  @>400A,R5    Dichte/Modus-Index lesen
       ANDI R5,>F0F0     relevante Bits maskieren
       MOV  R5,R0        kopieren
       SRL  R5,4         ins Low-Nibble kopieren
       SWPB R0           Rest ins High-Nibble High-Byte
       SOC  R0,R5        ehemaliges Low-Byte ins High-Nibble
       ANDI R5,>FF00     nur das High-Byte zählt
       BL   @>45F8       Step-Zeit des LW ins High-Nibble Low-Byte einbauen
       LI   R4,>000C     12 Versuche sind erlaubt
*
* Einsprung bei Retry
*
447E   MOVB @>4412,@>50(R9) Fehlerbyte ad hoc löschen
       BL   @>49E2          Laufwerk zuschalten
       CLR  R10             Register für Laufwerknummer löschen
       MOVB @>4C(R9),R10    Laufwerknummer holen
       SRL  R10,7           mal 2 ins Low-Byte
       COC  @>43E6(R10),R5  Dichtemaske in Bezug auf das LW prüfen
       JEQ  >449A           Bit gesetzt -> Single Density
       SBZ  >A           Bit nicht gesetzt -> DDEN-Leitung aktivieren
       JMP  >449C        weiter
449A   SBO  >A              Single Density
449C   MOV  @>4000(R10),R0  Letzte Spur auf diesem Laufwerk (Kopfstellung)
       ANDI R0,>F0F0        High-Nibbles maskieren
       MOV  R0,R1        zur Rechnung kopieren
       SLA  R1,4         in die Low-Nibbles
       SOC  R1,R0        Bits anpassen
       ANDI R0,>FF00     High-Byte isolieren
       CI   R0,>2700     Spur über 39?
       JLE  >44BA        Nein, Laufwerk wurde bereits benutzt
       BL   @>4A32       FDC-Kommando RESTORE, Kopf über Spur Null fahren
       CLR  R0           Spurnummer = 0
44BA   SBZ  7            Diskettenunterseite anwählen
       MOVB R0,@>5FFA    momentane Spur ins Spurregister des FDC-Chips
       MOV  @>4A(R9),R1  Sektornummer holen
       CLR  R0              Rechenregister löschen (32-Bit Division!)
       COC  @>43E6(R10),R5  Ist dem Laufwerk Doppelte Dichte zugeordnet?
       JEQ  >44D2           Nein

       DIV  @>440E,R0   Ja, durch >0012 teilen
       JMP  >44D6
44D2   DIV  @>440C,R0    SD, durch >0009 teilen
44D6   CI   R0,>0028     Spurnummer über 40?
       JL   >44E8        Nein, der betr. Sektor befindet sich auf der Oberseite
       LI   R3,>004F     Ergänzungswert
       S    R0,R3        Spurnummer davon abziehen
       JLT  >4586        negativ -> Sektornummer war zu groß
       SBO  7            Nummer O.K. -> Disk-Unterseite schalten
       MOV  R3,R0        Spurnummer kopieren
```

```
44E8   SWPB R0              ins High-Byte
       MOVB R0,@>4000(R10)  Spurpuffer aktualisieren
       SRC  R0,12           Bitpositionen anpassen
       MOVB R0,@>4001(R10)  und die restlichen Bits ablegen
       SRC  R0,4
       MOVB R0,@>5FFE    Nummer ins Sektorregister (hätte auch schon früher
*                        gemacht werden können!)
       SWPB R1           Sektornummer ins High-Byte
       MOVB R1,@>5FFC    ins Sektorregister
       CB   R0,@>5FF2    ist die Spur bereits aktuell?
       JEQ  >4514        Ja, Kopf bleibt da, wo er ist
       BL   @>4AA6       Nein, FDC-Kommando
       DATA >1C00        SEEK TRACK
       BL   @>49D0       Befehlsende abwarten
       SLA  R0,13        SEEK ERROR Bit herausschieben
       JOC  >4580        Gesetzt -> Seek Error!
4514   MOV  @>4E(R9),R2  Pufferadresse holen
       MOVB @>4D(R9),R0  ebenso den I/O-Code
       JEQ  >458C        0 -> schreiben
       COC  @>43F2,R5    RAM-Transfer-Bit aktiv?
       JEQ  >452C        Ja, VDP-Setup umgehen
       BL   @>4AE0       VDP-Schreibadresse setzen
       LI   R2,>8C00     Zeiger laden
*
* Verify-Einsprung
*
452C   BL   @>4AB2       FDC-Kommando
       DATA >8C00        READ SECTOR
       LI   R6,>0100     256 Byte sind Standard
       SOCB @>43F6,R5    Fehlerbit a priori setzen
       CLR  R0           überflüssig!
       SBO  >02          WAIT ENABLE
       MOVB @>4D(R9),R0  I/O-Code
       JEQ  >4548        Schreiben, also jetzt Verify
       BL   *R9          Routine im PAD aufrufen
       JMP  >454C        Weiter mit Fehlerprüfung
4548   BL   @>A(R9)      Verify-Routine steht 10 Byte hinter der Leseroutine
454C   SZCB @>43F6,R5    Fehlerbit löschen (wird nur beim normalen Ende gemacht)
4550   BL   @>49CE       Busy-Ende (Befehlsende) abwarten
       SLA  R0,13        Record-not-found-Bit gesetzt?
       JOC  >4568        Ja, Fehler >21
       JLT  >456E        CRC-Fehler -> Fehler >22
       COC  @>43F6,R5    Abbruch-Bit gesetzt?
       JEQ  >457A        Ja, Verify wurde abgebrochen! -> Fehler >28
       SLA  R0,2         Lost Data?
       JOC  >4574        Ja, Fehler >23
4564   B    @>46BE       Befehl zu Ende bringen
*
4568   BL   @>4A60       Fehler >21: Sektor nicht gefunden (RNF)
       DATA >2100
456E   BL   @>4A60       Fehler >22: CRC-Fehler (Prüfsumme)
       DATA >2200
4574   BL   @>4A60       Fehler >23: Lost Data
       DATA >2300

457A   BL   @>4A60       Fehler >28: Verify-Fehler
       DATA >2800
4580   BL   @>4A60       Fehler >11: Spur nicht verifiziert
       DATA >1100
4586   BL   @>4A78       Fehler >07: Programmparameter falsch
       DATA >0700
*
* Ausführung Sektor schreiben
*
458C   COC  @>43F2,R5    Transfer-Bit gesetzt?
       JEQ  >459A        Ja -> RAM-Transfer
       BL   @>4AF0       Nein, VDP-Leseadresse setzen
       LI   R2,>8800     Pointer auf VDPRD
459A   BL   @>4AB2       FDC-Kommando
```

```
       DATA >AC00        WRITE SECTOR
       LI   R6,>0100     256 Bytes
       SBO  >02          WAIT ENABLE
       BL   *R9          Routine im PAD-RAM ausführen
       BL   @>49CE       Busy-Ende
       SLA  R0,11        Write Protect?
       JOC  >45D6        Ja, Abbruch mit Fehler >34
       SLA  R0,2         Record not found?
       JOC  >45DC        Ja, Fehler >31
       SLA  R0,2         Lost Data?
       JOC  >45E2        Ja, Fehler >33
       MOVB @>4001,R2    Turbo-Option
       COC  @>43F0,R2    Letztes Bit im High-Nibble HB gesetzt?
       JEQ  >4564        Ja, Routine beenden, kein Verify!
       MOV  @>4E(R9),R2  Nein, Pufferadresse holen
       COC  @>43F2,R5    RAM-Transfer-Bit
       JEQ  >452C        Ja, Verify von dort
       BL   @>4AF0       Nein, VDP-Zugriff vorbereiten
       LI   R2,>8800     Zeiger
       JMP  >452C        nun Verify aus VDP-RAM
*
45D6   BL   @>4A78       Fehler >34: Disk ist schreibgeschützt!
       DATA >3400
45DC   BL   @>4A60       Fehler >31: Sektor nicht gefunden (RNF)
       DATA >3100
45E2   BL   @>4A60       Fehler >33: Datenverlust (LOST DATA)
       DATA >3300
*
45E8   TB   >12   Step-Einstellung
       TB   >16          LW 1
       TB   >15   Step-Einstellung
       TB   >0F          LW 2
       TB   >14   Step-Einstellung
       TB   >0E          LW 3
       TB   >13   Step-Einstellung
       TB   >11          LW 4
*
* Step-Timing Bits je nach LW in R5 einbauen
*
45F8   CLR  R10          Rechenregister vorbereiten
       MOVB @>4C(R9),R10   Laufwerknummer
       SRL  R10,6        mal 4 ins Low-Byte
       X    @>45E4(R10)  F2-Bit testen
       JNE  >460A        Schalter an, schneller
       ORI  R5,>0040     Schalter offen, langsam
460A   X    @>45E6(R10)  F1-Bit testen
       JNE  >4614        nicht aktiv
       ORI  R5,>0080     doch -> Code komplettieren
4614   SRL  R10,1        Laufwerknummer durch 2 (vergebene Mühe, die rufenden
*                        Routinen nutzen das nicht aus!)
       RT                Ende
*
* I/O-Routinen kopieren, PAD-RAM dabei sichern
*
4618   MOV  R11,R7         R11 sichern
       SOCB @>43F2,@>4001  Index, welche Routinen kopiert wurden
       LI   R6,5           5 Worte für die Leseroutine
       MOVB @>4D(R9),R0  I/O-Code
       JNE  >4634        nicht 0 -> lesen
       AI   R6,8           Schreiben -> 8 Worte mehr (incl. Verify)
       SZCB @>43F2,@>4001  Schreiben indizieren
4634   MOVB @>4C(R9),R0    Laufwerknummer
       JEQ  >4694          Null -> Fehler >07
       CB   R0,@>4414    gegen 4 vergleichen
       JH   >4694        zu groß -> auch nichts!
       MOV  R9,R2        R2 = >8300
       LI   R3,>400C     Zeiger auf den betreffenden Bereich des System-RAMs
*
4646   MOV  *R2+,R0      PAD-Inhalt nach R0
```

```
       MOV  R0,*R3+      zuerst die High-Nibbles
       SLA  R0,4         Rest verschieben
       MOV  R0,*R3+      und nun die High-Nibbles
       DEC  R6           alles durch?
       JNE  >4646        noch nicht
       MOV  R9,R2        PAD-Start wieder holen
       MOVB @>4D(R9),R0  I/O-Code
       JNE  >4664        Lesen
*
       LI   R3,>46A4     Schreib- und Verifyroutinenadresse
       LI   R6,>000D     13 Worte
       JMP  >466C        weiter
4664   LI   R6,>0005     5 Worte
       LI   R3,>469A     Adresse Leseroutine
466C   MOV  *R3+,*R2+    Routinen verschieben
       DEC  R6           Fertig?
       JNE  >466C        Nein
       MOVB @>400B,R8    Transfer-Index
       COC  @>43F2,R8    CPU-RAM-Transfer?
       JNE  >4692        Nein
       MOVB @>4D(R9),R0  Ja, Lesen oder schreiben?
       JNE  >468E        Lesen
       SOC  @>4408,*R9     Schreiben, im TS-Feld ein *R2+ erzeugen
       SOC  @>440A,@>A(R9) dto. im TD-Feld der Verify-Sequenz
       JMP  >4692
468E   SOC  @>440A,*R9   Lesen, im TD-Feld ein *R2+ erzeugen
4692   B    *R7          Ende
*
4694   BL   @>4A78       Fehler >07: z.B. Laufwerknummer falsch
       DATA >0700
*
469A   MOVB @>5FF6,*R2   Sektorleseroutine
       DEC  R6
       JNE  >469A
       RT
*
46A4   MOVB *R2,@>5FFE   Sektorschreibroutine
       DEC  R6
       JNE  >46A4
       RT
*
46AE   CB   @>5FF6,*R2   Datenvergleichsroutine
       JNE  >46BA
       DEC  R6
       JNE  >46AE
       RT
46BA   B    @>4550

*
* Ende des Sektor-I/O
*
46BE   LI   R6,>0005
       MOVB @>4001,R0    Welche Routinen wurden kopiert?
       COC  @>43F2,R0    Leseroutinen?
       JEQ  >46D0        Ja
       AI   R6,>0008     Nein, Schreibroutinen -> mehr ist wiederherzustellen!
46D0   ANDI R5,>4F00     Transfer- und Dichtebits in R5 isolieren
       MOVB R5,@>400B    ablegen
       SRC  R5,12
       MOVB R5,@>400A    dto. den Rest
       SRC  R5,4
       MOV  @>58(R9),R2  Adresse des Systembereichs im VDP-RAM
       AI   R2,>FFF6     auf Drive-Info-Block zeigen
       BL   @>4AE0       Schreibadresse
       MOVB @>4000,R0    Nummer des zuletzt benutzten Laufwerks
       SRL  R0,4         anpassen
       MOVB R0,@>8C00    in den Disk-Info-Block
       MOV  R9,R2        PAD-Start
       LI   R3,>400C
```

```
46FC   MOV  *R3+,R0      PAD wiederherstellen
       ANDI R0,>F0F0     irrelevante Nibbles löschen
       MOV  *R3+,R1      nächster Anteil
       ANDI R1,>F0F0     maskieren
       SRL  R1,4         anpassen
       SOC  R1,R0        zusammenflicken
       MOV  R0,*R2+      ins PAD-RAM
       DEC  R6           alle Bytes restauriert?
       JNE  >46FC        noch nicht
       COC  @>43F4,R5    welcher Art war der Routinenaufruf?
       JNE  >471E        Standard
       MOV  @>48(R9),R11 Nein, System-Aufruf (CALL MGR etc.)
       RT                Internes Ende
*
471E   B    @>567E       Standard-Exit
*
4722   DATA >F000
*
* Diskette formatieren
*
4724   CLR  R1
       MOVB @>4D(R9),R1  Geforderte Anzahl Spuren
       CI   R1,>2800     40?
       JLE  >473C        weniger
       SRL  R1,1         Mehr -> durch 2 teilen
       MOVB R1,@>4D(R9)      wieder zurück
       MOVB @>43EA,@>51(R9)  >02, egal was war - jetzt wird's Double Sided!
473C   MOVB @>51(R9),R1      Seitenzahl
       JEQ  >4748        Null, dann wird's >01
       CB   R1,@>43EA    Wert gültig?
       JLE  >474E            Ja
4748   MOVB @>43E8,@>51(R9)  >01 wenn's nicht >02 war
474E   SZCB @>4722,@>4C(R9)  Laufwerk normieren
       MOVB @>4D(R9),R1      Spuranzahl
       MOVB @>43E9,@>4D(R9)  I/O-Code Schreiben
       BL   @>4618           Routinen ins PAD-RAM kopieren
       MOVB R1,@>4D(R9)      R1 ist unverändert, Spurnummer zurück
       MOV  @>400A,R5    Optionen
       ANDI R5,>F0F0     relevante Bits maskieren
       MOV  R5,R0        Kopie für Rechnung
       SRL  R5,4         Low-Nibble bilden
       SWPB R0           High-Nibble hoch
       SOCB R0,R5        Byte erzeugen
       ANDI R5,>FF00     Optionsbits und Dichtemaske im HByte
       BL   @>45F8       Step-Zeit ermitteln
       BL   @>49E2       Laufwerk schalten
       SBZ  7            Default Seite 0
       BL   @>4A32       Restore Drive
       CLR  R10              Spur-Index
       CB   @>50(R9),@>43EA  Dichte >02?
       JNE  >4798            Nein
       SBZ  >A           Ja, doppelte Dichte schalten
       B    @>489C       In DD-Formatierroutine springen
*
* Disk in Single Density formatieren
*
4798   SBO  >A           SD schalten
       CLR  R3           Spurnummer
       MOV  @>4E(R9),R2  Pufferadresse VDP
       BL   @>4AE0       als Schreibadresse setzen
       CLR  R2           Sektornummer
       SETO R0           >FF
       BL   @>49C2       Index-GAP
       DATA >000C        12 Bytes, nicht 22 wie üblich!
47AE   CLR  R0           >00
       BL   @>49C2       Füllen
       DATA >0006          6 Bytes
       MOVB @>4417,@>8C00  >FE, AMID
       NOP
```

```
       MOVB R3,@>8C00    Spurnummer
       NOP
       MOVB R10,@>8C00   Seitenindex
       COC  @>43F0,R5    Interlace aus ROM?
       JEQ  >47D6               Nein
*
* Befehlsausführung testen
*
49CE   SBZ  >02          Wait-Disable
49D0   MOVB @>5FF0,R0    Status-Byte lesen
       JLT  >49DC        Not Ready!
       SRC  R0,9         FDC noch BUSY?
       JOC  >49D0        Ja, warten
       RT                Ende
*
49DC   BL   @>4A78       Fehler melden
       DATA >0600        >06, Hardware-Fehler
*
* Laufwerk zuschalten
*
49E2   MOV  R11,R7
       SZCB @>43F6,R5   Index 'altes Laufwerk'
       MOVB @>4000,R0   letztes Laufwerk
       SRL  R0,4         rechtsbündig im High-Byte
       CB   R0,@>4C(R9)  wie neues Laufwerk?
       JEQ  >49F8        Ja
       SOCB @>43F6,R5    Nein, vermerken
49F8   MOVB @>4C(R9),R0  Laufwerknummer holen
       SLA  R0,4         auf 4-Bit-RAM anpassen
       MOVB R0,@>4000  in Puffer übernehmen
       SRL  R0,12      rechtsbündig im Low-Byte
       LI   R2,>0080   Walking-Bit
       SLA  R2,0       entsprechend schieben
       AI   R12,8      CRU-Basis auf Drive-Select
       LDCR R2,4       LW anschalten
       AI   R12,-8     Basis reparieren
       SBZ  8          LW 4 aus
       CI   R0,4       eben das gefragt?
       JNE  >4A1E      Nein
       SBO  8          Ja, zuschalten
4A1E   COC  @>43F6,R5  anderes Laufwerk?
       JNE  >4A30      Nein, fertig
       LI   R0,>0C80   Ja, Umschaltpause
4A28   SWPB R5          Dummy-Befehl
       SWPB R5
       DEC  R0          Schluß?
       JNE  >4A28       Noch nicht!
4A30   B    *R7         Aber jetzt
*
4A32   MOV  R11,R8      Rückkehr sichern
       BL   @>4AA6      FDC-Kommando
       DATA >0800       RESTORE
4A3A   MOVB @>5FF0,R0   Statusregister lesen
       JLT  >49DC       NOT READY
       SRC  R0,9        FDC noch Busy?
       JOC  >4A3A       Ja, warten
       MOVB @>5FF0,R0   Status nochmal holen (überflüssig wegen Circular Shift!)
       SLA  R0,6        TRK 00 aktiv?
       JOC  >4A50       Ja, Spurnummer auf Null setzen
       B    @>49DC      Fehler >06
*
* Laufwerk konnte justiert werden (Spur Null korrekt erreicht)
*
4A50   CLR  R6           Spurnummer = 0
       MOVB @>4C(R9),R6  Laufwerknummer
       SRL  R6,7               als Wort-Zeiger (mal 2)
       MOV  @>4412,@>4000(R6)  >00 in Spurpuffer
       B    *R8                Ende
*
```

```
* Fehlerbehandlung mit Retries
*
4A60   DEC  R4           Fehlerzähler runter
       JEQ  >4A78        Null -> Vorbei!
       CLR  R1           Rechenregister
       MOVB @>4C(R9),R1  Laufwerknummer
       SRL  R1,7           mal 2 ins Low-Byte
       XOR  @>43E6(R1),R5  betreffende Dichte umschalten
       BL   @>4A32         RESTORE, Kopf über Spur Null
       B    @>447E       weiter
*
* Fehler ohne Retry-Möglichkeit
*
4A78   MOV  *R11+,R0     Fehlercode holen
       MOVB R0,@>50(R9)  an die rufende Software weitergeben
       CI   R0,>0600     Hardware-Fehler?
       JNE  >4AA2        Nein
       BL   @>4AB2       Ja, FDC-Chip rücksetzen
       DATA >D000        FORCE INTERRUPT
4A8A   MOVB @>5FF0,R0    Statusregister lesen
       SRC  R0,9         BUSY-Bit herausschieben
       JOC  >4A8A        Chip ist noch nicht bereit, warten
       CLR  R6           Rechenregister
       MOVB @>4C(R9),R6  Laufwerknummer
       SRL  R6,7         mal 2 als Wort-Zeiger
       LI   R0,>FFFF      Au weia...
       MOV  R0,@>4000(R6) Spurnummer zu >FF machen, erzwingt Restore beim
*                         nächsten Zugriff auf dieses Laufwerk
4AA2   B    @>46BE       PAD-RAM wiederherstellen etc.
*
* FDC-Kommandoroutine
*
4AA6   MOV  R5,R0      Step-Maske und Optionsbits kopieren
       SLA  R0,2       in Position
       ANDI R0,>0300   normieren (Options-Bits stören hier)
       SOC  *R11+,R0   Kommando einbauen
       JMP  >4AB4      weiter
4AB2   MOV  *R11+,R0   Kommando übernehmen
4AB4   MOVB @>5FF0,R6  Status-Register
       SLA  R6,1       Ready ins Carry
       SBZ  1          Motor-Strobe
       SBO  1
       JNC  >4ACC      LW ist bereit
       LI   R6,>80E8   Pausenzähler
4AC4   SWPB R5         Dummy zum Erhalt von R5
       SWPB R5
       DEC  R6         Pause um?
       JNE  >4AC4      Nein
4ACC   MOVB R0,@>5FF8  Op-Code in Kommandoregister
       SBO  3          Head-Load
       SWPB R5         Pause vor nächstem Zugriff
       SWPB R5         (Platzverschwendung!)
       SWPB R5
       SWPB R5
       SWPB R5
       SWPB R5
       RT              Ende
*
* VDP-Schreibadresse mit Wert in R2 setzen
*
4AE0   MOVB @>E5(R9),*R15  Low-Byte R2
       ORI  R2,>4000       High-Byte anpassen
       MOVB R2,*R15      High-Byte setzen
       ANDI R2,>3FFF     Befehlsbit wieder löschen
       RT
*
* VDP-Leseadresse mit Wert in R2 setzen
*
4AF0   MOVB @>E5(R9),*R15  Low-Byte R2
```

```
       ANDI R2,>3FFF       Lesebefehl garantieren
       MOVB R2,*R15      High-Byte nachschieben
       RT                Ende
*
* Fortsetzung des Power-Up-Link
*
4AFC   AI   R12,8
       LDCR @>4412,5     Laufwerke alle aus
       AI   R12,-8
       SBZ  1            Motoren anwerfen
       SBO  1
       MOVB @>4A88,@>5FF8  Interrupt-Code als Software-Reset
       LI   R0,>0F0F       Preset-Werte für's System-RAM
       LI   R6,>4000   Beginn der RAM-Zone
       MOV  R0,*R6+    Letztes Laufwerk 0, Verify an
       SLA  R0,4       ab jetzt >F schreiben
       MOV  R0,*R6+    Laufwerk 1 nicht justiert
       MOV  R0,*R6+    dto. Laufwerk 2
       MOV  R0,*R6+    dto. Laufwerk 3
       MOV  R0,*R6+    dto. Laufwerk 4
       SRL  R0,4       wieder >0 schreiben
       MOV  R0,*R6     Dichte komplett DD, Transfer in VDP, ROM-Interlace
       BL   @>49CE           auf Abarbeitung des Interrupts warten
       MOVB @>4412,@>50(R9)  Fehlerbyte auf >00
       B    @>567E           Ende
*
* ...
*
* Beginn der Operationen zur Rückkehr ins untere ROM
*
567E   MOV  @>8366,R11   VDP-Stackpointer
       MOVB @>83F7,*R15  Low-Byte R11 in VDPWA
       ANDI R11,>3FFF    maskieren
       MOVB R11,*R15     High-Byte in VDPWA
       INCT @>8366       Zeiger korrigieren
       MOVB @>8800,R11   Low-Byte der Rückkehradresse
       SWPB R11
       MOVB @>8800,R11   High-Byte der Rückkehradresse
       B    @>5F78       ROM ausblenden etc.
*
* ...
*
* Übergang vom unteren ins obere ROM
*
5F70   SBO  >0B            Oberes ROM anschalten
       MOV  @>4080(R1),R1  Zeiger zur Routine aus Liste
       B    *R1            anspringen
*
* Rücksprung ins untere ROM
*
5F78   SBZ  >0B   Oberes ROM aus
       RT         Ende (R11 aus VDP-Stack)
```

ROM-Listing Atronic-Controller

```
4000 bis 400F      DATA 0        16 Byte Nullen um Fehlfunktionen zu vermeiden
                                 falls versehentlich Bit >B an ist
4010 bis 49EF      DATA >FFFF    nicht programmiert

4A00 bis 4AFF      System-RAM des Controllers (2114). Pro Byte ist nur das Low-
                   Nibble verfügbar!

   4A00            Nummer des zuletzt angesprochenen Drives
   4A02            Letzte Spur auf LW 1
   4A04            Letzte Spur auf LW 2
   4A06            Letzte Spur auf LW 3
   4A08            Letzte Spur auf LW 4
   4A0A bis 4A18   keine Verwendung
   4A1A            Density-Bitmap für Sektor-I/O
   4A1B bis 4A1F   keine Verwendung
   4A20 bis 4A53   Puffer für den Inhalt des PAD-RAMs während der Sektor-
                   I/O-Routinen die wegen des Timings den 16 Bit-Bus
                   brauchen!
   4A54 bis 4AFF   keine Verwendung

Vektoren für den Übergang vom unteren in das obere ROM

4B00   DATA >484D  Nicht zugewiesen, der Bereich ist leer.
4B02   DATA >4B16  Fortsetzung des Powerup-Links.
4B04   DATA >4BDE  Sektor Schreib/Lese-Operationen.
4B06   DATA >4F30  Diskette formatieren.
4B08   DATA >51CC  Fortsetzung von CALL MANAGER.

4B0A bis 4B14      DATA 0   Leer

Fortsetzung des Power-Up-Link.

4B16   AI   R12,8          CRU-Basis hochschalten
       LDCR @>4BB6,5       Drive-Select- und Side-Select-Leitungen auf Ground.
       AI   R12,-8         Alte Basis (>1100) wiederherstellen
       SBZ  1
       SBO  1              Drive-Motoren anwerfen (Monoflop triggern).
       MOVB @>4EC6,@>5FF8  FDC-Chip mit >D0 rücksetzen.
                           Force Interrupt, Terminate with no Interrupt.
       MOV  @>58(R9),R2    Aus >8358 die VDP-Adresse des Volume-ID Blocks holen.
       AI   R2,-10         auf die VDP-Startadresse der DISK DRIVE INFO zeigen.
       BL   @>4E7C         VDP-Adresse mit Wert in R2 zum Schreiben vorbereiten.
       LI   R0,7
4B3C   MOVB R0,@-2(R15)    R15 enthält VDPWA da GPLWS, VDPWA-2 = VDPWD !!
       DEC  R0             Disk-Info und 3 folgende Bytes löschen.
       JNE  >4B3C          noch nicht fertig!
       BL   @>4B94           FDC Wait disable, warten bis BUSY-Zustand beendet.
       MOVB @>4BB6,@>50(R9)  Wert in >8350 (HBy) löschen.

System-RAM 2114 initialisieren.

       CLR  R0           Werte vorbereiten
       SETO R2
       LI   R6,>4A00     Anfangsadresse des 4 Bit RAM
       MOV  R0,*R6+      Letztes Drive = 0
       MOV  R2,*R6+      Letzte Spur auf LW1 = >FF. Das zwingt die Sektor-
       MOV  R2,*R6+      Routine zum Initialisieren der aktuellen Spur.
       MOV  R2,*R6+
       MOV  R2,*R6+      alle 4 Laufwerke.
                         Der VDP-Puffer ab >3EEB ist normalerweise nur bis LW 3
                         ausgelegt, er wird aber nicht benutzt.
       MOV  R0,@>4A1A    Density-Bitmap komplett auf Double Density.

Meldung in das Titelbild setzen.
```

```
       LI   R0,>01A6       Bildschirmposition der Meldung.
       BL   @>4BA6         VDP-Adresse mit Wert in R0 zum Schreiben vorbereiten.
       LI   R0,>4B80       Pointer auf den Text
       LI   R2,>14         Länge des Textes
4B74   MOVB *R0+,@-2(R15)  Daten sukzessive in VDPWD schreiben
       DEC  R2             fertig?
       JNE  >4B74          nein - weiter
4B7C   B    @>5F68         ROM ausblenden und Rückkehr aus DSR.

4B80   TEXT '  DISK SYSTEM READY '

4B94   SBZ  2              FDC Wait disable.
4B96   MOVB @>5FF0,R0      FDC Status lesen.
       JLT  >4BA2          High-Bit gesetzt - Drive not ready!
       SRC  R0,9           Low-Bit gesetzt ?
       JOC  >4B96          Ja - noch Busy, weiter warten.
       RT                  Low-Bit auf Null - Ende.
4BA2   B    @>50B8         Fehler - Abbrechen!

VDP-Adresse mit Wert in R0 zum Schreiben setzen.

4BA6   ORI  R0,>4000       Schreib-Operation ankündigen.
       SWPB R0             Low-Byte zuerst.
       MOVB R0,*R15        in VDPWA
       SWPB R0
       MOVB R0,*R15        High-Byte zuletzt.
       RT                  Ende.

4BB4   DATA >0000
4BB6   DATA >0001,>0002,>0004,>0008
4BBE   DATA >0010,>0020,>0040,>0080
4BC6   DATA >0100,>0200,>0400,>0800
4BCE   DATA >1000,>2000,>4000,>8000

4BD6   DATA >0012          Anzahl der Sektoren pro Spur bei Double Density.
4BD8   DATA >0009          Anzahl der Sektoren pro Spur bei Single Density.
4BDA   DATA >0004
4BDC   DATA >0200          Step-Timing Index (0000 bei  6 ms)
                                             (0100 bei 12 ms)
                           Fabrikeinstellung:(0200 bei 20 ms)
                                             (0300 bei 30 ms)
Sektor Lesen/Schreiben.

4BDE   MOVB @>4A1A,R5      alte Density-Bitmap in R5, R5 dient als Statuspuffer.
       ANDI R5,>0F00       relevante Bits maskieren.
       BL   @>4E0A         Laufwerk prüfen, Sektor I/O-Routine ins PAD setzen.
       LI   R4,10            Anzahl der zulässigen Versuche für Sektor-I/O.
       MOVB @>4BB6,@>50(R9)  >8350 (HBy) löschen (kein Fehler).
       BL   @>4DB0           Laufwerk-Nr. prüfen, Select-Leitung setzen.
       CLR  R7
       MOVB @>4C(R9),R7    LW-Nummer aus >834C holen.
       SRL  R7,7           LW-Nr. mal 2 im Low-Byte.
       MOV  @>4A00(R7),R0  Spur-Nr. des zuletzt benutzten LW holen.
       ANDI R0,>0F0F       Low-Nibs isolieren,
       MOV  R0,R1          kopieren,
       SLA  R1,12          High-Nib vom Low- ins High-Byte,
       AB   R1,R0          Low-Nib aufaddieren,
       ANDI R0,>FF00       und High-Byte isolieren.
       CI   R0,>2700       über die letzte Spur (ggfs. 1. Zugriff auf das LW).
       JLE  >4C1E          Nein - weiter.
       BL   @>4D4E         Ja - Kopf über Spur 0 fahren (Restore).
       CLR  R0             Spur-Index zu Null setzen.
4C1E   MOVB R0,@>5FFA      geforderte Spur-Nr. in das FDC-Reg. Daran erkennt
                           der Controller die richtige Position über der Spur!
       CLR  R0
       MOV  @>4A(R9),R1    in >834A wurde die Sektor-Nr. kopiert.
       COC @>4BC4(R7),R5   ist das dem LW entsprechende Bit im Status gesetzt?
       JEQ  >4C36          Ja - dann ist dem LW Single Density zugeordnet!
       SBZ  >A             Nein - Double Density schalten!
```

```
       DIV  @>4BD6,R0      Wert in R0 (Sektor-Nr.) durch >12 teilen.
                           Nach der Division steht in R0 die Spur, in R1 der
                           Sektor in dieser Spur.
       JMP  >4C3C          weiter.
4C36   SBO  >A             Single Density schalten.
       DIV  @>4BD8,R0      durch >9 teilen.
4C3C   SBZ  7              Diskettenoberseite anwählen.
       SZC  @>4BD0,R5      Seitenindex im Statusregister löschen.
       CI   R0,40          Wurde eine Spur über 40 berechnet?
       JL   >4C58          Nein
       LI   R2,79          möglicherweise Seite 2.
       S    R0,R2          79 minus Spur-Nr. gibt Komplement, Spur auf Seite 2.
       JLT  >4D18          weniger als Null - Sektor-Nr. war zu hoch.
       SBO  7              Spur-Nr. i.O., Diskettenrückseite anwählen.
       SOC  @>4BD0,R5      Seitenindex setzen.
       MOV  R2,R0          Spur-Nr. in R2.
4C58   SWPB R0             Spur-Nr. ins Low-Byte.
       MOVB R0,@>4A00(R7)  Spur-Puffer aktualisieren.
       SRC  R0,4
       MOVB R0,@>4A01(R7)  Byte komplett ablegen.
       SLA  R0,4           Wort reparieren.
       MOVB R0,@>5FFE      Spur-Nr. für Seek in das Datenregister.
       SWPB R1             Sektor-Nr. ins High-Byte.
       MOVB R1,@>5FFC      Sektor-Nr. in Sektor-Register für später.
       CB   R0,@>5FF2      Track Register lesen, Spur schon erreicht?
       JEQ  >4C84          Ja - weiter.
       BL   @>4D82         Nein - SEEK, Load Head, Verify on Track.
       DATA >1C00          FDC-Kommando Typ I, mit Stepping Rate verODERn!
       BL   @>4B96         warten bis Busy beendet, Abbruch wenn Drive not ready
       SLA  R0,13          Status prüfen.
       JOC  >4D12          SEEK-Error - Abbruch, Retry möglich!
4C84   MOVB R1,@>5FFC      Sektor-Nr. für Verify in das Sektor Register.
       MOV  @>4E(R9),R2    PAB-Buffer Adresse holen.
       MOVB @>4D(R9),R0    Schreib-Lese Opcode holen.
       JEQ  >4CDA          wenn 00 dann schreiben.
       BL   @>4E7C         Adresse des PAB-Buffers setzen.
       MOV  R15,R2         VDPWA kopieren.
       AI   R2,-2          VDPWD erzeugen.
4C9C   BL   @>4D74         READ SECTOR, Single Record, Compare Side, 30 ms Delay
                           Enable Side Compare.
       DATA >8600          Kommandotyp II
       LI   R6,256         Byteanzahl für Sektor-R/W.
       SOCB @>4BD4,R5      High-Bit setzen.
       SBO  2              Wait enable.
       CLR  R0
       MOVB @>4D(R9),R0    Schreiben oder Lesen?
       JNE  >4CBC          wenn nicht Null, dann lesen.
       CLR  R0
       BL   @>A(R9)        Geschrieben wurde schon, jetzt Verify (Offset A!).
       JMP  >4CBE
4CBC   BL   *R9            entspricht BL  @>8300!
4CBE   SZCB @>4BD4,R5      High-Bit wieder löschen.
4CC2   BL   @>4B94         Wait-Disable etc., Verify-Error-Entry.
       SLA  R0,13          Record not found?
       JOC  >4D24          Ja - Abbruch!
       JLT  >4D2A          CRC-Error - auch Ende!
       COC  @>4BD4,R5      High-Bit gesetzt?
       JEQ  >4D36          Verify-Fehler!
       SLA  R0,2           Lost-Data (Timing-Fehler des Programms)?
       JOC  >4D30          Ja
       B    @>4EDA         R5 zurück nach >4A1A, PAD reparieren etc.  ENDE.

Sektor schreiben.

4CDA   BL   @>4E86       VDP-Adresse mit R2 zum Lesen setzen.
       MOV  R15,R2
       AI   R2,>FBFE     VDPRD herstellen.
       BL   @>4D74       WRITE SECTOR, Single Record, Compare Side, 30 ms Delay,
                         Enable Side Compare.
```

```
       DATA >A600        Kommandotyp II.
       LI   R6,256       Byte Count.
       SBO  2            Wait enable.
       BL   *R9          Sektor schreiben.
       BL   @>4B94       Wait disable etc.
       SLA  R0,11        Bit S6 prüfen.
       JOC  >4D3C        Fehler, Write Protect!
       SLA  R0,2         Bit S4 prüfen.
       JOC  >4D42        Fehler, Record not found!
       SLA  R0,2         Bit S2 prüfen.
       JOC  >4D48        Lost Data, Timing Fehler!
       MOV  @>4E(R9),R2  Buffer-Adresse holen.
       BL   @>4E86       VDP-Adresse mit R2 setzen (Lesen)
       MOV  R15,R2
       AI   R2,>FBFE     VDPRD erzeugen.
       JMP  >4C9C        Sektor lesen zum Verify benutzen.

Fehlerroutinenaufrufe    Aufruf wenn:

4D12   BL   @>4E98       Alle Retries fehlgeschlagen oder SEEK-Error.
       DATA >1100
4D18   BL   @>4E98       Sektor-Nr. zu gross.
       DATA >0700
4D1E   BL   @>4EB6       Drive-Nr. falsch.
       DATA >0700
4D24   BL   @>4E98       READ-Error, Record-Not-Found.
       DATA >2100
4D2A   BL   @>4E98       READ-Error, CRC-Error.
       DATA >2200
4D30   BL   @>4E98       READ-Error, Lost Data.
       DATA >2300
4D36   BL   @>4E98       Verify fehlgeschlagen.
       DATA >2800
4D3C   BL   @>4EB6       WRITE PROTECTED.
       DATA >3400
4D42   BL   @>4E98       WRITE-Error, Record not found.
       DATA >3100
4D48   BL   @>4E98       WRITE-Error, Lost Data.
       DATA >3300

Subroutinen

Kopf über Spur Null positionieren.

4D4E   MOV  R11,R8       R11 sichern.
       BL   @>4D82       RESTORE, Load Head at beginning, no Verify on Track!
       DATA >0800        Kommandotyp I, mit Stepping-Rates verODERn.
       BL   @>4B96       warten auf Busy-Ende.
       BL   @>4D66              Status ein Mal prüfen ob Spur 0 erreicht ist.
       MOV  @>4BB4,@>4A00(R7)   Spur-Puffer loeschen.
       B    *R8                 Ende.

Auf Hardware-Error prüfen, Track 0 nicht erreicht

4D66   MOVB @>5FF0,R0    FDC-Status lesen.
       SLA  R0,6         Bit S2 prüfen, TRK 00.
       JOC  >4D72        OK - weiter.
       B    @>50B8       Error-Exit mit I/O-Error 06! Kopf nicht über Spur 0!
4D72   B    *R11         Ende.

FDC-Kommando Typ II (Read/Write Sector)

4D74   MOV  *R11+,R0     Data holen.
       COC  @>4BD0,R5    Seite 2?
       JNE  >4D8C        Nein
       SOC  @>4BCC,R0    Ja - Data mit >0800 verodern.
       JMP  >4D8C

FDC-Kommando Typ I (Seek, Restore, Step)
```

```
4D82   MOV  *R11+,R0     Data holen.
       SOCB @>4BDC,R0    Je nach Step-Time unterschiedliche Werte in >4BDC!
       JMP  >4D8C

Allgemeine FDC-Kommandoroutine

4D8A   MOV  *R11+,R0     Data holen.
4D8C   MOVB @>5FF0,R6    Status lesen.
       SLA  R6,1         High-Bit gesetzt, Drive not ready?
       SBZ  1
       SBO  1            Motor anwerfen.
       JNC  >4DA4        Bit nicht gesetzt, Drive ready!
       LI   R6,>7530     Schleife.
4D9C   SRC  R5,4         Verzögerung, um den Motor hochlaufen zu
       SRC  R5,4         lassen.
       DEC  R6           Fertig?
       JNE  >4D9C        Nein.
4DA4   MOVB R0,@>5FF8    Data in FDC-Chip.
       SBO  3            Head-Load.
       SRC  R5,8
       SRC  R5,8
       B    *R11

Subroutine Laufwerk zuschalten

4DB0   SZCB @>4BD4,R5    High-Bit loeschen.
       MOVB @>4A00,R0
       ANDI R0,>0F00     Letztes LW, das benutzt wurde.
       CB   R0,@>4C(R9)  ist es das gewünschte LW?
       JEQ  >4DC6        Ja.
       SOCB @>4BD4,R5    Nein - New-Drive Index setzen.
4DC6   CLR  R0
       MOVB @>4C(R9),R0  aktuelle LW-Nr. holen.
       JEQ  >4D1E        Fehler, LW-Nr.=0!
       MOVB R0,@>4A00    aktuelles LW ist jetzt das letzte Drive.
       SRL  R0,8         ins Low-Byte.
       C    R0,@>4BDA    mit 4 vergleichen.
       JH   >4D1E        Fehler, LW-Nr.>4!
       LI   R2,>0080     Bitmuster für CRU
       SLA  R2,0         mit R0 links schieben
       AI   R12,8        CRU-Basis hoch, um die unteren Bits nicht zu ändern.
       LDCR R2,3         Drive-Select Leitung aktivieren.
       AI   R12,-8       Basis restaurieren.
       CI   R0,4         Laufwerk 4?
                         Hier steht in der Version 1.0  CI R2,4 weshalb das LW 4
                         nicht erkannt wird.
       JNE  >4DF4        nicht LW 4!
       SBO  8            Drive-Select separat für LW 4!
       JMP  >4DF6
4DF4   SBZ  8            LW 4 ggfs. abschalten.
4DF6   COC  @>4BD4,R5    High-Bit gesetzt, neues Laufwerk?
       JNE  >4E08        nein - weiter.
       LI   R0,>0BB8     Schleife, Umschaltpause
4E00   SRC  R5,4
       SRC  R5,4
       DEC  R0
       JNE  >4E00
4E08   B    *R11         Ende.

4E0A   CLR  R7
       MOVB @>4C(R9),R0  LW-Nr. holen.
       JNE  >4E16        nicht Null -OK!
4E12   B    @>4D1E       Fehler, falsches LW!
4E16   CB   R0,@>4BDB    mit 4 vergleichen.
       JHE  >4E12        zu hoch, Fehler!
       LI   R6,>A        Anzahl der Bytes der Leseroutine.
       MOVB @>4D(R9),R0  Schreib/Lese-Code.
       JNE  >4E2C        nicht 00, lesen.
```

```
       SETO R7           Schreib-Flag.
       LI   R6,>1A       Anzahl der Bytes der Schreibe-Routine(n).
4E2C   LI   R3,>4A20     Adresse des Puffers im 2114.
       MOV  R9,R0        PAD-Basis kopieren.
       MOV  R6,R1        Byteanzahl wird zweimal gebraucht!

4F2A   DATA >484D,>F000,>F700

Diskette formatieren

4F30   CLR  R5
       SOC  @>4BCE,R5          R5= >1000
       MOVB @>4D(R9),R8        Anzahl der Spuren in R8
       MOVB @>4BB4,@>4D(R9)    Löschen
       SZCB @>4F2C,@>4C(R9)    LW-Nr. maskieren.
       BL   @>4E0A             LW-Nr. prüfen und Sektor-I/O Routinen ins PAD.
       MOVB R8,@>4D(R9)        Spuranzahl ins Low-Byte zusammen mit LW-Nr.
       CLR  R7
       MOVB @>4C(R9),R7        LW-Nr. in R7
       SRL  R7,7               mal zwei und ins Low-Byte.
       MOVB @>50(R9),@>52(R9)  Density-Code kopieren.
       CB   @>51(R9),@>4BC8    Seitenzahl mit >02 vergleichen
       JNE  >4F68              einseitig
       SETO @>4A(R9)           FAC als Double Sided Index verwenden
4F68   MOVB @>4BB4,@>50(R9)    ursprünglichen Dichte-Code löschen
       BL   @>4DB0             Drive-Select Leitungen setzen
       BL   @>4D4E             Restore, Kopf über Spur Null.
       CLR  R3                 Spurzähler löschen
4F78   MOV  @>4A(R9),@>4A(R9)  FAC prüfen ob gesetzt.
       JEQ  >4F8A              FAC nicht gesetzt - Single Sided!
       SBO  7                  Side-Select auf 'Opposite'!
       LI   R10,>0100          Seitenindex für den Spuraufbau
       BL   @>4F4E             Initialisiere eine Spur
4F8A   SBZ  7                  Side-Select 'Upper'
       CLR  R10                Seitenindex auf 'Seite 1'
       BL   @>4F4E             eine Spur initialisieren
       BL   @>4D82             Step in, update track register, load head
       DATA >5800              no verify on destination track.
                               Kommandotyp I, mit stepping rates verODERn.
       BL   @>4B96             auf Status-Bit warten
       AI   R3,>0100           Spurzähler + 1 im High-Byte
       CB   R3,@>4D(R9)        Alle Spuren ?
       JNE  >4F78              Nein - weiter.
       BL   @>4D4E             Restore, Kopf zurück.
       MOVB @>4D(R9),R0        Spuranzahl in R0
       SRL  R0,8               ins Low-Byte
       CB   @>4BC8,@>51(R9)    Seitenzahl mit >02 vergleichen
       JNE  >4FBA              Single Sided!
       SLA  R0,1               Double Sided, hat die doppelte Spuranzahl!
4FBA   CB   @>4BC8,@>52(R9)    Density mit >02 vergleichen
       JNE  >4FCE              Single Density!
       MPY  @>4BD6,R0          Double Density, mit >12 malnehmen!
       MOVB @>4BD7,@>4D(R9)    Anzahl der Sektoren pro Spur
       JMP  >4FD8
4FCE   MPY  @>4BD8,R0          mit >09 malnehmen, Single Density!
       MOVB @>4BD9,@>4D(R9)    Sektoren pro Spur =>09
4FD8   MOV  R1,@>4A(R9)        Gesamtzahl der Sektoren in FAC.
       LI   R6,>1A
       B    @>4EF8             PAD restaurieren und Ende.

Subroutine zum Formatieren einer Spur

4FE4   MOV  R11,R8
       MOV  @>4E(R9),R2        PAB-Buffer Adresse holen
       BL   @>4E7C             Adresse setzen
       CB   @>4BC8,@>52(R9)    Density-Code
       JNE  >4FFA              Single Density!
       B    @>510C             Double Density!
```

```
Subroutine zum Formatieren einer Spur Single Density

4FFA   LI   R6,>16           GAP 1, am Anfang der Spur stehen 22 Nullen
       SBO  >A               SD anzeigen
       CLR  R2               Sektor-Zähler
       JMP  >5008            beim 1. Sektor 22 Nullen
5004   LI   R6,6             ADDRESS GAP, alle weiteren Sektoren haben 6 Nullen!
5008   MOVB @>4BB6,@-2(R15)  Nullen schreiben
       DEC  R6               fertig?
       JNE  >5008            nein
       MOVB @>50C9,@-2(R15)  >FE schreiben, AMID
       NOP
       MOVB R3,@-2(R15)      Spur-Nr. schreiben
       NOP
       MOVB R10,@-2(R15)     Seiten-Nr. der Diskette (0 - Lower, 1 - Upper)!
       MOVB R3,R0            Spur-Nr. kopieren
       SRL  R0,8             ins Low-Byte
       SWPB R10              Seiten-Nr. ins Low-Byte.
       MPY  @>50C5(R10),R0   Seite 0 - Multiplikation mit @>50C4 (=0006)!
                             Seite 1 - Multiplikation mit @>50C6 (=0003)!
       SWPB R10              Seitenindex korrigieren
       A    R2,R1            Sektor-Zähler zu R1 addieren
       DIV  @>4BD8,R0           teilen durch >0009
       MOVB @>50DF(R1),@-2(R15) Sektor-Nr. aus der Interlace-Tabelle.
       LI   R6,-20
5040   MOVB @>50DE(R6),@-2(R15) ID-GAP aufbauen
       INC  R6
       JNE  >5040
       LI   R0,>E5E5           Inhalt des Datenbereichs eines Sektors
       BL   @>50AC
       DATA >0100              Wert in R0 256 mal schreiben.
       MOVB @>50DE,@-2(R15)    >F7, CRC-Auslöser.
       SETO R0
       BL   @>50AC
       DATA >002D              45 mal >FF anhängen
       INC  R2                 nächster Sektor
       CI   R2,9               alle 9 bearbeitet?
       JNE  >5004              nein
       BL   @>50AC
       DATA >00E7              noch 231 mal >FF als GAP 4.
       LI   R4,3               beim Formatieren sind 3 Retries erlaubt.
5074   MOVB @>4E(R9),R2        PAB-Buffer Adresse holen
       BL   @>4E86             Adresse setzen
       BL   @>4D8A             Spur ist bereit, geschrieben zu werden
       DATA >F400              Write track, 30 ms delay, Kommandotyp III.
       MOV  R15,R2             VDPWA kopieren
       AI   R2,>FBFE           VDPRD erzeugen
       LI   R6,>0CA3           Anzahl der Bytes einer Spur in SD!
       SBO  2                  Wait enable
       BL   *R9                Werte in FDC schieben
       BL   @>4B94             Wait disable, warten
       SLA  R0,11              Write protect?
       JNC  >509C              Nein.
       B    @>4D3C             Doch - Abbruch!
509C   SLA  R0,4               Lost data?
       JNC  >50AA              Nein - Schreibversuch war O.K.!
       DEC  R4                 Fehlversuch, Retry möglich?
       JNE  >5074              Ja!
       BL   @>4EB6             Nein - Fehler!
       DATA >3300
50AA   B    *R8                Ende.

Subroutine VDP-Puffer mit gleichen Bytes füllen

50AC   MOV  *R11+,R6           Data holen
50AE   MOVB R0,@-2(R15)        R0 schreiben
       DEC  R6
       JNE  >50AE
       B    *R11              Ende.
```

```
Fehlercodeübergaberoutinen

50B8   BL   @>4EB6
       DATA >0600              Hardware-Error!
50BE   BL   @>4EB6
       DATA >0700              Software-Error!

Data's für den Spuraufbau

50C4   DATA >0006,>0003        seitenabhängige Multiplikatoren.
50C8   BYTE >F0,>FE            AMID

Bytes für ID-GAP

50CA   BYTE >01,>F7,>FF,>FF,>FF,>FF,>FF,>FF
50D2   BYTE >FF,>FF,>FF,>FF,>FF,>00
50D8   BYTE >00,>00,>00,>00,>00,>FB,>F7

Sektor-Interlace Liste Single Density (IL=3)

50DF   BYTE 0,7,5,3,1,8,6,4,2

50E8   DATA 0,0,0,0,0,0         Teil von GAP 2
50F4   BYTE 0
50F5   BYTE >F5,>F5,>F5,>FB     DAM

Sektor-Interlace Liste Double Density (IL=4)

50F9   BYTE 0,11,4,15,8,1,12,5,16,9,2,13,6,17,10,3,14,7

510B   BYTE 0

Subroutine zum Formatieren einer Spur Double Density

510C   CLR  R2                  Sektor-Zähler
       SBZ  >A                  Double Density
       LI   R0,>4E4E            Sync-Muster
       BL   @>50AC              GAP 1
       DATA >0020               32 mal >4E schreiben
511A   LI   R6,-15
511E   MOVB @>50F8(R6),@-2(R15)  ADDRESS-GAP
       INC  R6
       JNE  >511E
       MOVB @>50C9,@-2(R15)      >FE, AMID
       NOP
       MOVB R3,@-2(R15)          Spur-Nr.
       NOP
       MOVB R10,@-2(R15)         Seiten-Index
       MOVB @>50F9(R2),@-2(R15)  Sektor-Nr. aus der Interlace-Tabelle.
       NOP
       MOVB @>4BC6,@-2(R15)      >01, Sektorlänge
       NOP
       MOVB @>4F2E,@-2(R15)      >F7, CRC für AF
       LI   R0,>4E4E             GAP 2
       BL   @>50AC
       DATA >0016
       LI   R6,-16
515E   MOVB @>50F9(R6),@-2(R15)  GAP 2 und DAM
       INC  R6
       JNE  >515E
       LI   R0,>E5E5             Leerdaten
       BL   @>50AC
       DATA >0100
       MOVB @>4F2E,@-2(R15)      >F7, CRC für Sektor
       LI   R0,>4E4E
       BL   @>50AC               DATA GAP, GAP 3
       DATA >001C                28 Bytes
       INC  R2                   nächster Sektor
```

```
       C    R2,@>4BD6           gleich >0012?
       JL   >511A               nein - weiter

GAP 4 schreiben

       BL   @>50AC
       DATA >00BE
       LI   R4,3                3 Retries
5194   MOVB @>4E(R9),R2         PAB-Buffer
       BL   @>4E86              neu setzen
       BL   @>4D8A              Write track, 30 ms delay.
       DATA >F400               Kommandotyp III
       MOV  R15,R2
       AI   R2,>FBFE
       LI   R6,>190E            Anzahl der Bytes pro Spur bei DD!
       SBO  2                   Wait enable
       BL   *R9                 Datentransfer
       BL   @>4B94              Wait disable, warten
       SLA  R0,11               Write protect?
       JNC  >51BC               Nein.
       B    @>4D3C              Ja - Abbruch!
51BC   SLA  R0,4                Lost data?
       JNC  >51CA               Nein, OK!
       DEC  R4                  noch Retries?
       JNE  >5194               ja
       BL   @>4EB6              nein!
       DATA >3300
51CA   B    *R8                 Ende.
```

ROM-Listing MYARC DDCC-1

```
*
* Vektortafel
*
4000   DATA 0,0    nicht zugewiesen
*
       DATA >4010  CALL FILES, Direktmodus
       DATA >405A  CALL FILES über SBR >16
       DATA >0000  nicht zugewiesen
       DATA >43E4  Sektor-I/O extern (über DSRLNK)
       DATA >41B0  Formatieren
       DATA >4786  Sektor-I/O intern
*
* CALL FILES im Direktmodus
*
4010   MOV  R11,R10        Rückkehr sicherstellen
       MOV  @>FF4C(R4),R1  >832C liefert VDP-Adresse im Eingabepuffer
       AI   R1,7           plus 7, hinter FILES
       BL   @>40A8     Wort an dieser Adresse nach R0 holen
       CI   R0,>C801   String in Klammern, Länge 1?
       JNE  >404E      Nein, ignorieren
       INCT R1         Ja, nächstes Wort
       BL   @>40A8     lesen
       AI   R0,>CF4A   plus Offset
       CI   R0,>0900   über 9?
       JH   >404E          Ja, nicht erlaubt
       MOVB R0,@>FF6C(R4)  >834C
       BL   @>4062         Pufferzonen anpassen
       MOVB @>FF70(R4),R0  >8350
       JNE  >404E
       A    @>40A6,@>FF4C(R4)      >832C plus 12, Befehlsende
       SZCB @>FF62(R4),@>FF62(R4)  >8342 löschen
*
404E   MOV  R10,R11
4050   LI   R7,>1E03     Op-Code SBZ 3
       LI   R8,>045B     Op-Code RT
       B    R7           ROM ausblenden und Rückkehr
*
* CALL FILES über Level 1 Routine ausführen
*
405A   MOV  R11,R10      Rückkehr sichern
       BL   @>4062       Zuweisung vornehmen
       JMP  >404E        Ende
*
VDP-Pufferzonen anpassen - Diese Routine kümmert sich nicht um andere Systeme, wel-
che evtl. das VDP-RAM belegen. Damit ist ein Absturz dieser Systeme mit diesem Con-
troller unvermeidlich, wenn auf diesem Wege mehr VDP-RAM allokiert werden soll!
*
4062   MOVB @>FF6C(R4),R0  >834C, Files-Anzahl
       SRL  R0,8           ins Low-Byte
       JEQ  >40A2        Null ist unzulässig
       CI   R0,>0009     über Neun?
       JGT  >40A2        auch nicht erlaubt
       MOVB @>508D,R9    momentane Einstellung
       SRL  R9,8           ins Low-Byte
       MOV  @>FF90(R4),R1  Wert aus @>8370 (VDPHI) lesen
407A   C    R0,R9          beide angeglichen?
       JLT  >4088        Nein, alter Wert zu hoch
       JEQ  >4090        beide gleich - Ende
4080   AI   R1,>FDFA     VDPHI reduzieren
       INC  R9           entsprechend mehr Files erlaubt
       JMP  >407A        weiter
4088   AI   R1,>0206     518 Bytes pro File-Daten-Block
       DEC  R9           weniger Files erlaubt
       JMP  >407A        weiter
*
4090   SWPB R9           neue Files-Anzahl
```

```
       MOVB R9,@>508D      in Puffer
       MOV  R1,@>FF90(R4)  Neues VDPHI
       CLR  R0             Fehlercode >00
409C   MOVB R0,@>FF70(R4)  in @>8350
       RT                  Ende
*
40A2   SETO R0     Fehlercode >FF
       JMP  >409C  übergeben in @>8350
*
40A6   DATA >000C
*
* Wort aus VDP-RAM lesen
*
40A8   MOVB @3(R4),*R15  Low-Byte R1 setzen
       NOP
       MOVB R1,*R15      High-Byte nachschieben
       NOP
       MOVB @>FBFE(R15),R0   High-Byte lesen
       SWPB R0               umdrehen
       MOVB @>FBFE(R15),R0   Low-Byte lesen
       SWPB R0               Korrektur
       RT                Ende
*
* VDP-Adresse mit Offset setzen
*
40C0   A    *R11+,R1     Offset aus DATA holen
40C2   SWPB R1
       MOVB R1,@>8C02
       SWPB R1
       MOVB R1,@>8C02
       RT
*
40D0   DATA >0009,>0100
40D4   DATA >0005        Retries
40D6   DATA >0400
40D8   DATA >0028        Spuranzahl beim Formatieren
40DA   DATA >467C
40DC   DATA >FBFE,>D0D4
40E0   DATA >02F7,>0311,>3401
40E6   DATA >12FF,>2000
*
* Adreßfeld lesen, Ermittlung der Formatierung, DRQ wird ignoriert!
*
40EA   DATA >C000        FDC-Kommando READ ADDRESS
40EC   SBZ  3             DD ggfs. schalten
       MOVB @>83D4,@>5F01 Befehl an FDC
       LI   R11,>0002     ca. 2 Sekunden
40F8   SETO R10           Preset
40FA   TB   0        Interrupt, Befehl beendet?
       JEQ  >4106    Ja
       DEC  R10      Nein, warten
       JNE  >40FA
       DEC  R11
       JNE  >40F8
4106   SBO  3        SD und ROM an
       RTWP          Ende
*
* Warteschleife auf Befehlsabarbeitung
*
410A   LI   R9,>0003     Standard-Preset
410E   SETO R10          Schleife
4110   TB   0            Interrupt?
       JEQ  >416A        Ja, Befehl beendet
       DEC  R10
       JNE  >4110
       DEC  R9
       JNE  >410E
411C   LI   R6,>0600     Fehlercode >06
       BL   @>46C4         PAD-RAM restaurieren
```

```
       C    @>83F6,@>40DA  GPLWS R11 = >467C?
       JNE  >4132
       CI   R2,>0003     Laufwerk 4?
       JNE  >4136        Nein
4132   SBZ  1
       JMP  >413A
4136   BL   @>46C4       PAD-RAM restaurieren
413A   CLR  R9           Kein Fehler für PAB
       MOVB R10,@>8350   auch keiner sonstwie
       JEQ  >4150        Ja
       LI   R9,>C000     >03 in den oberen Bits, Illegale Operation
       CB   @>40E4,R10   >34, Write Protect?
       JNE  >4150        Nein
       LI   R9,>2000       Ja, >01 in den oberen 3 Bits, Schreibschutz
4150   CB   @>506A,@>40E3  >F7? (Interner Op-Code)
       JNE  >415C          Nein, Workspace nicht verändern
       LWPI >83E0        Ja, GPLWS laden
415C   MOV  @>5074,R11   Rückkehradresse holen
       CI   R11,>4832    Interner Sektor-I/O?
       JEQ  >416A        Ja, innerhalb des oberen ROMs zurückkehren
       B    @>4050       Ende
416A   RT
*
* VDP-Füllroutine
*
416C   MOV  *R11+,R1     Anzahl
416E   MOVB R0,@>8C00    Füllmuster
       DEC  R1           alle?
       JNE  >416E        Nein
       RT                Ende
*
* Sektor (mit DAM) aufbauen
*
4178   MOVB @>40DC,@>8C00   >FB, DAM
       LI   R0,>E5E5        Leerdaten
       LI   R1,>0100     256 Bytes
       JMP  >416E        füllen
*
* Adreßfeld aufbauen
*
4188   MOVB @>40DD,@>8C00   >FE, IDAM
       MOV  @>5046,R5
       MOVB @>508D(R5),@>8C00  Spurnummer
       MOVB R8,@>8C00          Seitenindex
       MOVB @>500F,@>8C00      Sektornummer, R7 LBy
       MOVB @>40E5,@>8C00    >01, Länge
       MOVB @>40E1,@>8C00    >F7, CRC-Auslöser
       RT
*
* Einsprung Formatieren
*
41B0   MOV  R11,@>5074
       LWPI >5000        Workspace im System-RAM
       BL   @>4716       PAD-RAM vorbereiten
       LI   R10,>0700
       CB   @>834D,@>40D9   Spuranzahl = >28 (.40)?
       JNE  >4136           ungleich - Fehler (sehr unflexibel!!)
       BL   @>4698       Laufwerk zuschalten
       CLR  R4                Zähler für Gesamtsektorsumme
41CE   MOVB @>40D7,@>508E(R2)  >00 in Spurnummernpuffer
       MOVB @>836C,R9          Step-Code?
       JNE  >41E2           Langsam
       MOVB @>40D4,@>5F01  >00, Restore schnell
       JMP  >41E8
41E2   MOVB @>40E0,@>5F01  >02, Restore langsam
*
* Innere Schleife mit Spurinkrement
*
41E8   BL   @>410A       Ende Restore abwarten
```

```
       CLR  R8           Seitenindex
41EE   CLR  R7           Sektornummer
       MOV  @>5034,R1    VDP-Adresse
       BL   @>40C0       VDP-Adresse zum
       DATA >4000          Schreiben setzen
       CB   @>5036,@>40E0  >02, Double Density?
       JEQ  >424E          Ja
*
* Spur SD im Puffer aufbauen
*
       CLR  R0           >00
       BL   @>416C       füllen
       DATA >0016        22 Bytes
420A   BL   @>4188       Adreßfeld aufbauen
       SETO R0           >FF
       BL   @>416C       ID-GAP, GAP 2
       DATA >000B        11 Bytes
       CLR  R0           >00
       BL   @>416C       ID-GAP cont.
       DATA >0006        6 Bytes
       BL   @>4178         Sektor aufbauen
       MOVB @>40E1,@>8C00  >F7, CRC-Auslöser
       SETO R0
       BL   @>416C       DATA-GAP, GAP 3
       DATA >002D        45 Bytes >FF
       CLR  R0           >00
       BL   @>416C       ADDRESS-GAP aufbauen
       DATA >0006        6 Bytes
       AI   R7,>0007     Sektornummer weiter
       CLR  R6           Rechenregister
       DIV  @>40D0,R6    durch 9 teilen
       MOV  R7,R7        Rest?
       JNE  >420A        ja, weiter
       SETO R0           >FF, GAP 4
       AI   R4,>0009     Sektorzähler weiter
       JMP  >42A6        Spur ist fertig zum Schreiben
*
* Spur DD im Puffer aufbauen
*
424E   LI   R0,>4E4E     GAP 1
       BL   @>416C       aufbauen
       DATA >0032        50 Bytes >4E
       CLR  R0           >00
       BL   @>416C       ADDRESS GAP
       DATA >000C        12 Bytes
       LI   R0,>F5F5     AMID-Kennung
       BL   @>416C       aufbauen
       DATA >0003        3 Bytes
       BL   @>4188       AF mit AMID aufbauen
       LI   R0,>4E4E
       BL   @>416C       ID-GAP, GAP 2
       DATA >0016        22 Bytes
       CLR  R0           plus Lücke
       BL   @>416C       >00
       DATA >000C        12 Bytes
       LI   R0,>F5F5     DAM vorbereiten
       BL   @>416C
       DATA >0003        3 mal wiederholen
       BL   @>4178         Sektor aufbauen
       MOVB @>40E1,@>8C00  >F7, CRC-Auslöser
       AI   R7,>0007       Interlace über modulo-Funktion
       ANDI R7,>000F     Low-Nib stehen lassen
       JNE  >424E        weiter wenn kein Overflow
       LI   R0,>4E4E     GAP 4 Daten
       AI   R4,>0010     Sektorzähler weiter
*
* Pufferinhalt schreiben, Spur formatieren
*
42A6   BL   @>416C       GAP 4 aufbauen
```

```
       DATA >02BC          Anzahl Bytes
       MOVB @>508E(R2),R11 Spurnummer holen
       JNE  >42BC          nicht Null
       MOVB @>40DF,@>5F01  >D4, Force Interrupt, unmittelbarer Interrupt
       BL   @>410A         warten
42BC   LI   R9,>4616     Adresse VDP-Transferroutine
       BL   @>4770       verschieben
       LI   R10,>83C4    letztes Wort der FDC-Schreibroutine
       LI   R9,>43D4     Füllroutine
42CC   MOV  *R9+,*R10+   anhängen
       CI   R9,>43E4     alle?
       JNE  >42CC        Nein
       INC  @>83A8       Befehl in Schreibroutine von Sektor- in Spur schreiben
       SBZ  2            Unterseite (Seite 0)
       MOV  R8,R8        welche Seite wird formatiert?
       JEQ  >42E0        unten
       SBO  2            Nein, oben. Anschalten
42E0   MOV  @>5034,R1      VDP-Adresse
       BL   @>40C2         setzen
       CB   @>40E0,@>5036  >02, Double Density?
       JNE  >42FE          Nein
       LI   R10,>17E6    Ja, Spurlänge DD effektiv (GAP 4 durch Füllroutine!)
       MOV  R10,@>83F4   R10 GPLWS
       BLWP @>460E       Double Density Schreibroutine
       JMP  >430A        weiter
42FE   LI   R10,>0BE3    Spurlänge SD ohne GAP 4 (s.o.)
       MOV  R10,@>83F4   R10 GPLWS
       BLWP @>4612       SD-Routineneinsprung
430A   MOVB @>5F01,R10   Statusregister lesen
       ANDI R10,>4000    Write-Protect isolieren
       JEQ  >431A        nicht aktiv
       SRL  R10,4        doch, in >0400 für Fehlercode wandeln
       B    @>4136       Abbruch
*
431A   CB   @>5037,@>40E0  >02, Double Sided?
       JNE  >4372          Nein
       MOV  R8,R8        Ja, schon Seite 2?
       JNE  >432E        Ja, dann weiter
       LI   R8,>0101     Nein, Seitenindex
       B    @>41EE       weitermachen
*
* Auf korrekte Formatierung Seite B prüfen
*
432E   MOV  @>5046,R5      Laufwerknummer (???)
       MOVB @>508E(R2),R0  Spur Null auf diesem LW?
       JNE  >4372          Nein, dann keinen neuen Test machen
       CB   @>40E0,@>5036  >02, Double Density?
       JEQ  >4346          Ja
       BLWP >477E        Adreßfeld lesen SD
       JMP  >434A
4346   BLWP @>4782       Adreßfeld lesen DD
434A   MOV  @>83F4,R11   R10 GPLWS
       JNE  >4354        R10 ist bis zur GAP 4 Sequenz gekommen!
       B    @>411C       Nein, I/O-Fehler >06 melden
4354   MOVB @>5F01,R5    Statusregister lesen
       SLA  R5,4         Record not found?
       JNC  >4372          Nein, OK
       MOVB @>40E5,@>5037  >01, nur einseitiges Formatieren möglich
       AI   R4,-9          Gesamtsektorzahl korrigieren
       CB   @>40E0,@>5036  Double Density?
       JNE  >4372          Nein
       AI   R4,-7        Ja, nochmal 7 ab gibt 16 weniger
*
4372   CLR  R6             Rechenregister
       MOVB @>508E(R2),R6  Spurprotokoll
       AI   R6,>0100       plus 1
       CB   R6,@>5033    Spuranzahl erreicht?
       JNE  >43B4        Nein
       BL   @>46C4       PAD-RAM restaurieren
```

```
       MOV  R4,@>834A    Gesamtsektoranzahl übergeben
       LI   R8,>0900       9 SPS SD
       CB   @>40E0,@>8350  >02, Double Density?
       JNE  >439A          Nein
4396   LI   R8,>1000       Ja, 16 SPS (gleichzeitig SEEK-Code!)
439A   MOVB R8,@>834D    SPS übergeben
       SLA  R2,2           Laufwerk mal 4
       MOV  R4,@>5058(R2)  Gesamtsektorzahl aufheben
       MOV  @>8350,R4      Dichte und (ggfs. korrigierte) Seiten
       SWPB R4             umdrehen
       MOV  R4,@>505A(R2)  umgedreht ablegen
       CLR  R10            kein Fehler
       B    @>413A       Ende
*
* Nächste Spur anfahren mit SEEK ohne VOT (Verify-On-Track)
*
43B4   MOVB R6,@>508E(R2)  R6 in Spurprotokoll (wurde bereits inkrementiert)
       MOVB R6,@>5F07      und ins Datenregister
       MOVB @>836C,R6    Step-Code
       JNE  >43CA          langsam
       MOVB @>4398,@>5F01  >10, Seek schnell
       JMP  >43D0
43CA   MOVB @>40E6,@>5F01  >12, Seek langsam
43D0   B    @>41E8         weiter mit warten auf Ausführung
*
* Füllroutine für's Formatieren, bis Interrupt kommt
*
43D4   MOVB @>8800,R10   letzten Wert aus VDP
43D8   MOVB R10,@>5F07   ins Datenregister
       TB   0            INTREQ?
       JNE  >43D8    nein, weiter, gnadenlos
       SBO  3        SD und ROM an
       RTWP          Schluss
*
* Einsprung Sektor-I/O
*
43E4   MOV  R11,@>5074  Globaler R11-Puffer
       BL   @>4716      PAD-RAM vorbereiten
       MOV  R2,R6       LW-Nummer
       SLA  R6,2        mal 4
       AI   R6,>505A    Zeiger auf Seiten/Dichte-Protokoll
       MOV  *R6,R4      Eintrag lesen
       MOVB @>40D7,R4   >00, Seite egal, Interrupt-Flag dafür einbauen
       MOV  R3,R3       Sektor Null?
       JNE  >442C       Nein
       MOV  R0,R0       Lesen von Disk?
       JEQ  >442C       Nein, Schreiben auf Disk
*
* Retry-Einsprung,Sektor lesen oder Laufwerk-Erstzugriff
*
4402   CLR  R6       Spurnummer = 0
       CLR  R7       spätere Sektornummer
       CLR  R5       wird nicht benutzt!
       SBZ  2            Seite 0 (unten)
       MOVB R12,R4       Interrupt-Flag aktiv
       BL   @>4698       Laufwerk zuschalten
       MOVB @>836C,R9      Step-Flag
       JNE  >441E          Langsam
       MOVB @>40D0,@>5F01  >00, RESTORE, Step 00
       JMP  >4424
441E   MOVB @>40E0,@>5F01  >02, RESTORE, Step 10
4424   MOV  R2,R9          Laufwerknummer kopieren
       AI   R9,>508E     Offset der Spurnummerntafel addieren
       JMP  >449A        weiter mit Übergabe der Nummer
442C   MOV  R2,R6        Laufwerknummer kopieren
       SLA  R6,2         mal 4
       AI   R6,>5058     plus Tafel 'Sektorobergrenze'
       MOV  *R6+,R8      maximale Sektoranzahl+1 holen
       JNE  >4440          nicht null - OK
```

```
       MOVB @>40E7,@>836A  >FF, Laufwerk und Parameter nicht initialisiert
       JMP  >4402          wieder von vorne
4440   LI   R10,>0700    Index 'Fehler >07'
       C    R3,R8        geforderter Sektor nicht zu gross?
       JL   >444C        Ja, kleiner - OK
       B    @>4136       Sektornummer zu hoch, I/O-Error 7
444C   BL   @>4698       Laufwerk zuschalten
       LI   R8,>0010       16 SPS
       CB   @>5009,@>40E0  >02, DD?
       JEQ  >4460          Ja
       LI   R8,>0009       Nein, SD, 9 SPS
*
452A   MOV  R0,R0        Schreiben auf Disk?
       JEQ  >4532        Ja
       AI   R1,>C000       Nein, Pufferzeiger korrigieren (für Verify)
4532   MOVB @>500F,@>5F05  Sektornummer ins Sektorregister
       MOVB R6,@>5F03      Spurnummer ins Spurregister
       MOVB R8,@>5F07      erster Wert im Datenregister
       CB   @>5009,@>40E0  >02, DD?
       JEQ  >454E          Ja
       BLWP @>4612       SD-I/O-Routine
       JMP  >4552
454E   BLWP @>460E       DD-I/O-Routine
4552   MOVB @>5F01,R10   Statusregister lesen
       COC  @>40D6,R10   >0400, LOST DATA?
       JEQ  >44DE        Ja, wiederholen
       ANDI R10,>1800    RNF oder CRC?
       JEQ  >45C6        Nein, keiner davon
*
* Fehler bei Sektor-I/O
*
4562   MOVB @>836A,R8    Fehler - Erstzugriff?
       JNE  >4584          Ja, bereits indiziert
       MOVB @>40E7,@>836A  Nein, >FF - jetzt nachholen
       MOVB @>836D,R9      Retries
       AI   R9,>FF00     minus 1 im High-Byte
       MOVB R9,@>836D    zurück
       JEQ  >45AC        Null - Ende
       MOV  R0,R0        I/O-Typ
       JEQ  >45A0        schreiben
       MOV  R3,R3        Sektor Null?
       JNE  >45A0          Nein
4584   CB   @>5009,@>40E0  >02, DD?
       JEQ  >45BE          Ja, jetzt mit SD probieren
*
* Dichte auf DD umschalten, Retries reduzieren
*
       MOVB @>836D,R9    Retry-Zähler holen
       AI   R9,>FF00     1 abziehen
       MOVB R9,@>836D    Ende bei Null
       JEQ  >45AC
       MOVB @>40E0,@>5009  >02, ab jetzt mit DD probieren
45A0   B    @>4402         Retry-Einsprung
*
45A4   SRL  R10,4         Fehlercode
       AI   R10,>3000     plus Kennung 'Schreiben'
       JMP  >45BA         übergeben
45AC   SRL  R10,4
       MOVB R10,R10       Kein Fehler?
       JNE  >45B6         doch
       AI   R10,>0200     hier muß schon was fehlerhaft sein!
45B6   AI   R10,>2000     Code plus Kennung 'Lesen'
45BA   B    @>4136        Äbergeben und Schluss
*
* Dichte auf SD wechseln, Retries □-1nicht□-0 reduzieren
*
45BE   MOVB @>40D7,@>5009  >00, auf Single Density umschalten
       JMP  >45A0          weiter mit Retry
*
```

```
* Sektor-I/O war fehlerfrei, Fortsetzung
*
45C6   MOV  R0,R0        I/O-Typ
       JNE  >45FA        Lesen, fertig
       MOVB @>5F01,R10   Statusregister lesen
       ANDI R10,>4000    Write Protect testen
       JNE  >45A4        Gesetzt - Schreibschutz!
       SETO R0           Nein, umschalten auf Lesen
       SETO @>5048       Index RAM-Transfer (für Dummy-Lesen in ROM!)
       CLR  R1             Pufferadresse ist >0000 (Konsolen-ROM)!
       MOVB @>40E7,@>836B  >FF, Verify-Index
       B    @>444C         weiter mit 'Laufwerk anschalten'
*
45E6   MOVB @>836B,R8  Verify-Durchlauf?
       JNE  >45F4      Ja, fertig
       MOV  R3,R3      Sektor 0?
       JNE  >45F4      Nein
       BL   @>46F0     Laufwerkdefaults bei jedem Zugriff auf Sektor 0 setzen!
*
45F4   CLR  R10        Kein Fehler
       B    @>4136     Fehlerbehandlung für Rückkehr mit benutzen
*
45FA   MOVB @>836A,R8  Parameter neu zu setzen?
       JEQ  >45E6      Nein, es ging mit den alten!
       BL   @>46F0         Laufwerkparameter setzen (Seiten, Dichte, MaxSec)
       MOVB @>40D7,@>836A  >00, keine neue Anpassung
       B    @>442C         nochmal von vorne anfangen
*
460E   DATA >83E0,>83A4  Context-Switch Double Density I/O
4612   DATA >83E0,>83A6  Context-Switch Single Density I/O
*
* Datentransferroutinen, liegen im PAD-RAM ab >83A2
*
4616   DATA >A0F0     FDC-Kommandos Write Sector, Write Track
       SBZ  3              DD und ROM aus (ggfs. überspringen)
       MOVB @>83A2,@>5F01  korrekten Code in FDC
4620   TB   1              DRQ?
       JEQ  >462E     Ja
       TB   0         Interrupt?
       JEQ  >4638     Ja
       JMP  >4620     Nein, weiter auf DRQ warten
462A   TB   1         DRQ?
       JNE  >462A          Nein
462E   MOVB @>8800,@>5F07  Ja, bedienen
       DEC  R10            alle Bytes geschrieben?
       JNE  >462A     Nein, nächsten DRQ abwarten
4638   JMP  >4652     Hier wird beim Formatieren die Füllroutine angehängt!
*
463A   DATA >8000     FDC-Kommando Read Sector
*
463C   SBZ  3               ggfs. DD schalten
       MOVB @>83A2,@>5F01  FDC-Code übergeben
4644   TB   0               INTREQ
       JEQ  >4654     Ja, vielleicht Schreibschutz
4648   TB   1         DRQ?
       JNE  >4644          Nein, Interrupt testen
       MOVB @>5F07,@>8C00  DRQ bedienen
4652   JMP  >4648          kein Zähler, Interrupt macht das
4654   SBO  3         SD schalten und ROM an
       RTWP           Ende
*
4658   DATA >A000     FDC-Kommando Write Sector
*
465A   SBZ  3              ggfs. DD schalten
       MOVB @>83A2,@>5F01  FDC-Code übergeben
4662   TB   1              DRQ?
       JEQ  >4670     Ja, bedienen
4666   TB   0         INTREQ?
       JEQ  >4678     Ja, Ende
```

```
       JMP  >4662     weiter pollen
466C   TB   1         DRQ?
       JNE  >4662         Nein
4670   MOVB *R11+,@>5F07  Daten aus CPU-RAM auf Disk
       DEC  R10           alle?
       JNE  >466C     Nein
4678   JMP  >4694     Schluß
*
467A   DATA >1000
467C   DATA >8000     FDC-Kommando Read Sector
*
467E   SBZ  3              DD und ROM aus
       MOVB @>83A2,@>5F01  FDC-Code
4686   TB   0              INTREQ?
       JEQ  >4694          Ja, Ende
468A   TB   1             DRQ?
       JNE  >4686          Nein
       MOVB @>5F07,*R11+  Ja, DRQ bedienen
       JMP  >468A         und weiter
4694   SBO  3         Ende, ROM an
       RTWP           Schluß
*
* Laufwerk zuschalten
*
4698   SBZ  4        LW 1 aus
       SBZ  5        LW 2 aus
       SBZ  6        LW 3 aus
       SBZ  7        LW 4 aus
       MOV  R12,R10  Basis kopieren
       AI   R12,8    auf Bit 4
       A    R2,R12   Laufwerknummer dazu
       A    R2,R12   nochmal
       SBO  0        dieses Laufwerk anschalten
       MOV  R10,R12  alte CRU-Basis
       AI   R12,14   auf Bit 7
       S    R2,R12   minus Laufwerk
       S    R2,R12   nochmal
       TB   0        Laufwerk mit kurzer Step-Zeit?
       JEQ  >46BE       Ja
       MOVB R12,@>836C  Nein, vermerken
46BE   MOV  R10,R12     Basis restaurieren
       SBO  1        Motoren an
       RT            Ende
*
* PAD-RAM restaurieren
*
46C4   MOV  R11,R9       R10 darf nicht verändert werden
       MOV  @>5038,R1    Stackpointer
       INC  R1           plus 1 auf Datenzone
       BL   @>40C2       Leseadresse setzen
       LI   R6,>83A2     Zieladresse der gesicherten Daten
46D4   MOVB @>8800,*R6+  aus VDP verschieben
       CI   R6,>8400     Ende erreicht?
       JNE  >46D4        noch nicht
*
       LI   R8,>5030     Pufferzone im System-RAM
       LI   R6,>834A     Zieladresse
46E6   MOV  *R8+,*R6+    verschieben
       CI   R6,>8352     alle?
       JNE  >46E6        Nein
       B    *R9          Ja, Ende
*
* Laufwerk Standard-Werte setzen
*
46F0   LI   R10,>470E    Zeiger DS/DD
       MOV  R2,R9        LW-Nr kopieren
       SLA  R9,2         mal 4
       AI   R9,>5058       plus Offset Parametertafel
       CB   @>40E0,@>5009  DD?
```

```
       JEQ  >4708          Ja
       LI   R10,>4712    Nein, SD-Daten
4708   MOV  *R10+,*R9+   Defaults in Protokollzone
       MOV  *R10,*R9
       RT                Ende
*
470E   DATA >0500,>0202  Defaults DS/DD
4712   DATA >02D0,>0201  Defaults SS/DD
*
* PAD-RAM sichern und Routinen vorbereiten
*
4716   MOV  @>5046,R2    Laufwerknummer
       DEC  R2           korrigieren als Zeiger
       CLR  @>836A         Index 'Laufwerkparameter gültig' & 'Kein Verify'
       MOV  @>40D4,@>836C  Step-Code und Retries
       LI   R6,>834A       Adressfeldroutine kommt hier hin
       LI   R8,>5030     Pufferzone
472E   MOV  *R6+,*R8+    verschieben
       CI   R6,>8352     alles?
       JNE  >472E        noch nicht
       LI   R6,>3EF4     Start Stackbereich
       SB   @>508D,R6    Files-Anzahl als Vielfaches von 256 davon abziehen
       MOV  R6,*R8       nach >5038 als Stackpointer
       MOV  R11,R9       Rückkehr gewährleisten
       MOV  R1,R8        Pufferadresse nicht verändern
       MOV  R6,R1        VDP-Stackpointer
       INC  R1           Korrektur
       BL   @>40C0       als Schreibadresse sichern
       DATA >4000        Offset bewirkt Indizierung
       LI   R6,>83A2     Hier kommen später die Transferroutinen hin
4752   MOVB *R6+,@>8C00  PAD ins VDP-RAM schaffen
       CI   R6,>8400     Ende erreicht?
       JNE  >4752        Nein
       LI   R1,>40EA     Adressfeldroutine
       LI   R6,>83D4     kommt hier hin
4764   MOV  *R1+,*R6+    verschieben
       CI   R6,>83F4     30 Bytes
       JNE  >4764        noch nicht
       MOV  R8,R1        Pufferzeiger wieder an alten Platz
       B    *R9          Ende
*
* FDC-Routinen kopieren, Routinentyp (R9) extern definiert
*
4770   LI   R10,>83A2    Hier fangen sie alle an
4774   MOV  *R9+,*R10+   verschieben
       CI   R10,>83C6    und hier hören sie alle auf
       JNE  >4774        wenn sie komplett sind
       RT                Schluß
*
477E   DATA >83E0,>83D8  Single Density AF-Routine
4782   DATA >83E0,>83D6  Double Density AF-Routine
*
* Interner Sektor-I/O
*
4786   MOV  @>506A,R0    I/O-Code
       CB   R0,>40D1     >09?
       JEQ  >4838        Ja
       MOV  R11,R9
       CI   R2,>57FE
       JEQ  >487A
       MOV  @4(R3),R5
       SRL  R0,2
       JOC  >47A4
       SRL  R0,1
       JOC  >47EE
47A4   CLR  @4(R3)
       MOV  R5,R6
       SLA  R6,1
       JNC  >47CA
```

```
       ANDI R5,>7FFF
       MOV  R5,@>5006    Sektornummer in SYSRAMWS R3
       JEQ  >47CA          Sektor null!
       CLR  @>5048         VDP-Transfer
       MOV  @>5056,@>5002  Pufferadresse in späteres R1
       BL   @>4828         Sektor schreiben
       BL   @>481E         Fehlercode aus GPLWS R10 oder SYSRAMWS R9 holen
47CA   MOV  @2(R3),@>5006  späteres R3, Sektornummer
       MOV  R3,R5          Datenpufferadresse kopiern
       AI   R5,>0012     plus Offset
       MOV  R5,@>5002    Pufferadresse, Zieladresse der Daten
       SETO @>5048       RAM-Transfer
       BL   @>4828         Sektor-I/O, interner Einsprung
       CB   @>40E8,@>5012  R9 im SYSRAMWS gleich >20?
       JEQ  >47EE          Ja
       BL   @>481E       Fehlercode ermitteln
47EE   MOV  @>506A,R2
       CB   @>40D6,R2
       JEQ  >4806
       CLR  *R3
       MOVB R10,@>8350   Fehlercode übergeben
       AB   R10,@>506B
4802   MOV  R9,R11
       JMP  >487A
*
4806   SLA  R2,12
       JNC  >4814
       CLR  @6(R3)
       MOVB @>40D7,*R3
       JMP  >4802
4814   SLA  R2,4
       JOC  >4802
       CLR  @>5070
       JMP  >4802
*
481E   MOVB R10,R10      R10 = >00?
       JNE  >4826        Nein
       MOVB @>5012,R10   Ja, R9 holen
4826   RT
*
* Subroutine 'Sektor schreiben'
*
4828   LWPI >5000        SYSRAMWS
       CLR  R0           I/O-Modus Schreiben
       BL   @>43E4       normale Routine verwenden
4832   LWPI >83E0        Das auf diese Adresse zeigende R11 indiziert den
       RT                internen Aufruf!!
*
4838   CLR  R1
       MOVB @>1E(R3),R1
       SRC  R1,13
       AB   @3(R4),R1
       MOV  @>24(R3),R0
       SWPB R0
       COC  @>40D6,R1
       JNE  >485E
       C    @6(R3),R0
       JNE  >487E
       CB   *R3,@>22(R3)
       JNE  >487E
       JMP  >4874
485E   MOV  @2(R3),R9
       JNE  >486E
       C    @>5070,@>40D2
       JL   >487E
       JMP  >4874
486E   C    @>5070,R0
       JL   >487E
4874   AI   R1,>0100
```

```
4878   MOV  R10,R11
487A   B    @>4050       Ende
*
487E   MOVB R1,@>5072
       JMP  >487A        Ende
*
4884   TEXT '(C)MYARC1984'
*
4890   Ab hier wiederholt um 1 erhöhte Bytes
       z.B. >90,>91,>92,>93,>94...
*
* ...
*
5000   Start des System-RAMs. Volle 2KByte 8-Bit RAM
*
* ...
*
5030 - 5037      Kopie des PAD-Bereiches >834A - >8351
5038             VDP-'Stackpointer'
*
* ...
*
5046             Laufwerknummer
504A - 5053      Filenamenpuffer
*
* ...
*
5058   Sektormaximum Laufwerk 1
505A   Seiten- und Dichtecode Laufwerk 1
505C   Sec-Max LW 2
505E   SD-Code LW 2
5060   Sec-Max LW 3
5062   SD-Code LW 3
5064   Sec-Max LW 4
5066   SD-Code LW 4
5068   R11-Puffer
*
* 506A bis 508C  PAB-Kopie
*
  506A - 5072    Die ersten 9 Bytes des PAB
  5073           Namenlänge (Gerät + Datei)
*
5074   Wird auch als R11-Puffer benutzt
*
  5074 - 508C    Geräte und Dateinamenpuffer
*
508D   Files-Anzahl (1-9)
508E   Aktuelle Spur Laufwerk 1
508F   Aktuelle Spur Laufwerk 2
5090   Aktuelle Spur Laufwerk 3
5091   Aktuelle Spur Laufwerk 4
*
5092 - 5191      Puffer für Sektorinhalt
*
* ...
*
53BE - 53C7      Puffer für Disknamen
53C8 - 54C7      Puffer für FDS des Files
54C8 - 57FF      >00
5800 - 5EFF      >FF - Nicht belegt
5F00 - 5FFF      FDC-Bereich, alle ungeraden Adressen belegt, alle 8 Byte
*                wiederholt
```

Zugriffsadressen des FDC im Myarc-Diskcontroller

Register	Funktion	
Adresse	Lesen	Schreiben

5F01	Statusregister	Kommandoregister
5F03	Sektorregister	Sektorregister
5F05	Spurregister	Spurregister
5F07	Datenregister	Datenregister

*

5F09 ff Wie an >5F01, siehe Hinweis

RAM-Disks

Als kleine Zugabe, quasi in letzter Minute, finden sich hier einige Anmerkungen zum Thema RAM-Disk.

Mit dem "Einbruch" der Horizon RAM-Disk, in den ersten Exemplaren noch mit einer eher vorsintflutlichen DSR ausgestattet, die von sämtlichen Standards unbeeinflußt war, ist dieses Speichermedium vor allem wegen der enormen Geschwindigkeit des Datentransfers heute ein essentieller Faktor bei der 'vernünftigen' Arbeit am TI.

Die Vorgänger waren jedoch schon länger am Markt. Das begann mit der Foundation 128K-Karte, die jedoch nur eine Art Bank-Switching aufwies, mit dem man sich mehr schlecht als recht den Speicher für eigene Assemblerprogramme erweitern konnte.

Sie diente sowohl der CorComp-RAM-Disk als auch der von Mechatronic als Vorbild, die als erste ein umgeschaltetes 32K-RAM zur Simulation einer Floppy-Disk aufwiesen.

Das zentrale Problem des CorComp-Geräts war das aller CorComp-Karten:

eine CorComp-Erweiterung im System war genug, mehrere konnten einige PE-Boxen aufgrund der unzulässigen Busbelastung zu sonderbarem Verhalten veranlassen. Über die DSR sei nur soviel gesagt, daß sie ähnliche Holzhammermethoden wie die von Myarc benutzte (Autor war Galen Reed) und somit im Zusammenspiel mit bestimmten ROS-Versionen anderer Karten regelmäßig für Probleme sorgte.

Die Mechatronic-DSR war mehr ein Notlaufsystem, über das hier nicht weiter ausgeführt werden soll. Der Autor ist (zu seinem Glück) unbekannt.

Hardware-Konzepte

Hier existieren im wesentlichen 2 Varianten: CPU-RAM Bank-Switching und DSR-RAM Bank-Switching. Die Horizon-Karte wendet das zweite Verfahren an, alle anderen Karten das erste. Aus verschiedenen Gründen ist das Horizon-Konzept als einziges geeignet, die Daten per Batterie puffern zu lassen. Der CorComp-Versuch mit dem externen Netzteil kann nur als untauglich bezeichnet werden.

Alle RAM-Disks simulieren softwareseitig ein Diskettenlaufwerk. Dazu dient ein Betriebssystem mit Namen ROS (RAM-Disk Operating-System).

CPU-RAM Bank-Switching

Die schaltungstechnisch einfachste Lösung ist die, den ganzen 32K-Speicher der Speichererweiterung mehrfach vorzusehen und aus einer DSR, die ja im Bereich >4000 nicht umgeschaltet wird, wenn sie erst einmal gewählt wurde, den passenden Bereich der RAM-Disk anzuschalten.

In dem Moment stehen in der 32K-Erweiterung nicht mehr die Anwenderprogramme sondern die Sektorinhalte eines bestimmten Bereiches der RAM-Disk. Diese werden dann ins VDP-RAM verschoben bzw. mit den dort abgelegten Daten neu gefüllt.

Ein RAM-RAM-Transfer, wie ihn neue ROS-Versionen der Horizon bieten, ist mit diesem Konzept nicht realisierbar. Ferner ist es nicht möglich, bei Anwendung einer 16bit-32K eine RAM-Disks dieser Art zu nutzen, da es unmöglich ist, die Standard 16bit-32K-Schaltungen zu deaktivieren und somit die Bankumschaltung möglich wird. Die Hardware-Details sollen hier nicht näher erläutert werden.

DSR-RAM Bank-Switching

Die verträglichste Lösung für die Implementation eines externen Speichers führt über eine Pipe (einen Zugriffskanal) im DSR-Bereich.

Da dieser nur bei Erkennen der korrekten Gerätenamen aktiviert wird, ist es prinzipiell möglich, soviele RAM-Disks in der Box zu haben, wie es freie CRU-Adressen gibt, ohne daß sich diese, von den Laufwerknummern einmal abgesehen, ins Gehege kommen.

Software-Konzepte

RAM-Disks verfügen i.d.R. über jede Menge Speicher; die Horizon sogar über ein DSR-RAM, was neben dem leichten ROS-Austausch auch noch einen eleganten Workspace liefert.

Damit besteht keine Notwendigkeit, das VDP-RAM mitzubenutzen, abgesehen von den normalen PABs und deren Datenpuffern. Im entsprechenden Absatz dieses Kapitels wird kurz auf die Probleme der parallelen Nutzung des VDP-RAMs eingegangen.

Laufwerkumleitung

Viele Programme für den TI scheinen aus einer Zeit zu stammen, als Diskettenlaufwerke mit Gold aufgewogen wurden. Dementsprechend erlauben sie nur Laufwerksnummern von 1 bis 3 oder 4 (das sind schon die etwas progressiveren...).

Wie weiter unten erläutert wird, ist es daher nötig, mittels einer entsprechenden Schaltung die RAM-Disk beim Absuchen der CRU-Seiten vor den Diskcontroller zu setzen, de facto also auf CRU-Basis >1000. Hier kann die RAM-Disk dann die Laufwerke 1 bis 4 simulieren, wobei allerdings der Zugriff auf den Diskcontroller nur noch mit Umwegen möglich ist.

Hier wurden von TI selbst aber auch von Drittanbietern von Hard- und Software etliche Fehler begangen.

So prüft jede normale DSR z.B. bei einem Interrupt oder etwa die TI-RS-232 beim normalen DSRLNK ab, ob sie denn gemeint ist. Wenn nicht, dann wird das Flag für die Fortsetzung der Suche gesetzt und die DSR verlassen.

Nicht so beim Diskcontroller.

Bei ihm wird die im PAB verfügbare Laufwerksnummer auf die controllerinterne Maximalzahl verglichen und gleich ein Fehler gemeldet. Alle phantasielosen 'Abschreiber' der TI-Disk-DSR haben diesen Fehler übernommen. Das Ergebnis ist, daß RAM-Disks über der CRU-Adresse >1100 meist nicht oder unzureichend funktionieren, wenn sie auf Sektorebene angesprochen werden, egal ob sie nun Laufwerksnummern über 4 bzw. 3 aufweisen oder nicht.

Partitionierung

Mit der Verfügbarkeit hoher Speicherkapazitäten zu relativ niedrigen Preisen stieg die RAM-Disk-Kapazität stufenweise an, bis theoretische Speichergrößen jenseits des Megabytes realisierbar wurden. Damit stellte sich das Problem, wie man diese Speichermenge mit dem TI-DOS sauber verwalten kann.

Da wie o.a. die Grenze der Diskettenkapazität 400 KByte beträgt, und man im Sinne der Softwarekompatibilität mit den leider recht vielen unsauber programmierten Tools und Anwenderprogrammen tunlichst nicht über die 360K hinausgehen sollte, muß der Speicherraum einer derart großen RAM-Disk in mehrere kleine Laufwerke mit der angeführten Maximalkapazität aufgeteilt werden.

Diesen Vorgang, bei dem bestimmte, zusammenhängende Speicherbereiche einer RAM-Disk jeweils einem anderen virtuellen Laufwerk zugeordnet werden, nennt man Partitionierung.

Namenszugriff

Das TI-DOS erlaubt den Zugriff auf Daten durch Kombination des Dateinamens mit dem Namen der Diskette, auf der sich die betreffenden Dateien befinden. Das DOS sucht dann zuerst die Diskette anhand des Namens und dann die Datei. Intern wird natürlich der bei der Suche ermittelte korrekte Laufwerksbezeichner benutzt, doch merkt das der Anwender meist nicht.

RAM-Disk-Betriebssysteme sind nun so aufgebaut, daß sie beim Zugriff über den Disknamen die Anfrage dann weitergeben, wenn sie den Namen nicht in der RAM-Disk-Partitionstafel gefunden haben. Alle anderen Diskcontroller laufen bis Laufwerk 3 bzw. 4 und meckern dann (s.o.).

Mit diesem Verfahren kann trotz Umleitung einer Diskettenstation z.B. auf DSK1 für das Originale E/A-Modul, noch auf eine Diskette im Laufwerk 1 zugegriffen werden, wenn nur die RAM-Disk 1 anders benannt wird.

Konflikte bei der Parallelnutzung des VDP-RAMs

Das VDP-Konzept des VDP-Area-Links soll vermeiden, daß derartiges passiert. Leider ist es nicht zum Allgemeingut geworden, wie dieser VDP-Area-Link dazu benutzt werden kann, jeder DSR ihr eigenes 'Reich' im VDP-RAM zuzuweisen.

Manche RAM-Disk DSRs 'pfriemeln' recht mutig im Bereich des Diskcontrollers herum, andere lassen 'die Finger weg'.

Beides ist nicht ganz so, wie es sein könnte.

Im Interesse einer konsistenten und vor allem kompatiblen Verwaltung aus diversen Programmiersprachen, sollte ein offenes File immer den reservierten Bereich der File-Puffer belegen. Da neben dem Namen auch die Laufwerksnummer zum File gehört, die auch Teil des Namensvergleichs ist, gibt es nur dann Probleme, wenn RAM-Disk und Diskcontroller auf dem selben physischen Laufwerk eine Datei gleichen Namens offen halten. Die gängigen DSRs für Diskcontroller melden 'File already open', wenn so etwas passiert und greifen

munter darauf zu. Um also das VDP-RAM 'mehrbenutzerfähig' zu machen, müssen diese Fehler von der Programmseite restriktiver behandelt werden.

Weiterhin müßte jede DSR so geändert werden, daß sie die Daten eines angeforderten File-Abschnitts immer liest, auch wenn ihr die entsprechenden Pointer sagen, daß der Abschnitt schon im Datenpuffer steht.

Zudem muß eine sog. Updated-Data-Buffer-Area, besonders die von Sektor 0, nach Ende jedes File-I/Os rückgeschrieben werden.

Unter dieser Bedingung können RAM- und Floppy-Disk-DSR das VDP-RAM gemeinsam nutzen.

Zugriffsgeschwindigkeiten

Eine RAM-Disk ist schnell - das schnellste derzeit denkbare Speichermedium, nicht nur für den TI. Der Grund liegt auf der Hand: zum einen ist der TI-99/4A konzeptionell nicht DMA-fähig, zum anderen gehen sämtliche anderen externen Medien wie Floppy- und Hard- bzw. Optical-Disks den Umweg über einen Cache-Speicher, der doch irgendwann mal ins CPU-RAM übertragen werden muß, was eine RAM-Disk ja ausschließlich tut.

Wesentlich schlimmer, wenn es darum geht, diese potentielle Geschwindigkeit zu nutzen, macht sich ein langsames File-System bemerkbar.

Beim TI ist dies die satzorientierte Datei. Hier überwiegt der Verwaltungsaufwand für die Pointer, Buffer, Offsets etc., die das flexible Verwalten von Files ermöglichen, so daß eine RAM-Disk im Mittel bei Lese- und Schreibzugriffen (gemischt) nur etwa 30% schneller als eine DS/DD-Diskette ist. Mitverantwortlich für das Tempo ist auch der dauernde Umweg über das VDP-RAM, von dem ja nicht nur der PAB sondern auch der Datenpuffer betroffen ist.

Neuere ROS-Versionen erlauben durch Erweiterung des I/O-Codes den direkten RAM-RAM-Transfer, was signifikante Vorteile bringen kann.

Programme jedoch, die üblicherweise nicht zugängliche Daten im VDP-RAM suchen, laufen dann nicht mehr, da der Zeitgewinn bei einem Duplizieren ins VDP-RAM verloren ginge.

Harddisks

An Speicherkapazität sind Harddisks den RAM-Disks natürlich überlegen, nicht jedoch in Bezug auf die Zugriffsgeschwindigkeit.

Da Harddisks mechanische Systeme sind (zumindest zur Zeit), vergeht eine u.U. signifikante Zeitspanne, bis die Daten verfügbar sind - von Disk-Duplexing, Mirroring und Elevator-Seeking oder SCSI, ESDI und DCPs spricht man beim TI natürlich nicht.

Zudem sind sämtliche I/O-Kanäle des TI an die CPU gebunden, so daß diese nicht entlastet wird (mal abgesehen von den seriellen I/O-Chips) und es nur auf eher niedrige Datenraten bringt.

Hier soll nicht über Sinn oder Unsinn von Festplatten an einem 64K-Rechner wie dem TI oder dem 9640 geredet werden, vielmehr soll gezeigt werden, worin der prinzipielle Unterschied zwischen Disketten (Floppy- bzw. Flexible-Disks) und Harddisks (Fixed Disks) besteht.

Mechanischer Aufbau

Festplatten bestehen, wie der Name schon sagt, aus fest montierten Platten, die sandwichartig übereinander angeordnet sind.

Die Datenträger bestehen aus einer antimagnetischen Aluminiumlegierung und sind beidseitig mit einer magnetisierbaren Schicht ähnlich den Floppies beschichtet (natürlich viel feiner strukturiert).

Die höhere Speicherkapazität wird durch mehrere Faktoren erreicht:

- Eine höhere Rotationsgeschwindigkeit der Platten (~3000rpm),
- Geringer Abstände des über die Platten fliegenden Kopfes,
- Mehrere Platten,
- Schmalere Spuren und extrem kleine Köpfe, Kopfspalte,
- Optimierte Codierungstechniken.

Die höhere Rotationsgeschwindigkeit bewirkt eine Frequenzgangverbesserung, was die Flußwechselrate nach oben bringt. Zudem erhöht sich so die Datenübertragungsrate. Hierzu eine kleine Rechnung:

Eine Floppy-Disk, in MFM beschrieben, weist 18 Sektoren in einer Spur auf, von denen jeder 256 Byte enthält. Die Drehzahl beträgt 300 rpm, also 5 Umdrehungen pro Sekunde.

Dadurch werden 4,5 Kilobyte in 0,2 Sekunden, also maximal 22,5iKilobyte pro Sekunden gelesen bzw. geschrieben, wenn kein Verify stattfindet - mit Verify halbiert sich die Rate beim Schreiben.

Eine MFM-Harddisk verfügt über 17 Sektoren à 512 Byte pro Sektor und rotiert mit 3000 rpm, also 50 Umdrehungen pro Sekunde. Pro Sekunde werden also 25 mal 17 mal 1 Kilobyte übertragen, was eine Rate von 425 Kilobyte pro Sekunde ergibt.

Beide Rechnungen legen einen 1:1 Interleave zugrunde.

Die Kopfsteuerung der Harddisks wurde in den vergangenen Jahren mehrfach überarbeitet.Es finden sich Konzepte wie bei Floppies (Spannbandstepper) oder Voice-Coils und Linearmotoren.

Aufgrund der geringeren Toleranzen ist es jedoch nicht so einfach, eine Spur einfach so wiederzufinden, zudem man immer auch die Wärmeausdehnung aller Komponenten beachten muß.

Hierzu verfügt jede Harddisk normaler Technologie über eine komplett separierte Plattenseite, auf der ausschließlich Spurinformationen für die Positioniereinheit (Servo) aufgezeichnet sind. Neben dem normalen Steppen kann hier anhand eines Referenzpegels eine Nachjustierung (Feinjustage) erfolgen, falls einige Parameter aufgrund Alterung oder Wärmeeinfluß verschoben sind.

Ansteuerung

Zusätzlich zu den für Floppies üblichen Signalen können Harddisks einfache Befehle ausführen wie z.B. das Recalibrate anhand der o.a. Servo-Spuren. Dazu verfügt der, inzwischen veraltete, Shugart-Bus, der sich bei Harddisks als Seagate-Norm ST506 präsentiert, über zusätzliche Leitungen. Zudem hat jede Harddisk ihr eigenes Datenkabel.

Bei der Softwareansteuerung muß beachtet werden, daß neben Spur und Seitennummer nun auch eine Plattennummer benötigt wird.

Praktisch wird mit Zylinderbereichen, Zylindernummern und Kopfnummern sowie Sektornummern gearbeitet.

Zudem ist sich jede Festplatte 'bewußt', daß ihre Fehlerrate aufgrund der höheren Speicherdichte in Bereiche kommen kann, wo es kritisch wird. Daher wird als Analogon zum CRC-Byte bei Floppies ein sog. ECC-Byte (Error-Correction-Code) hinzugefügt, über den die Festplatte selbst fehlerkorrigierend (bis 5 Bit) tätig werden kann. Ist jedoch die ECC-Correction-Span überschritten, tritt der Fehler endgültig auf.

Ob dies ein Fortschritt zum gnadenlosen CRC-Fehler bei Floppies ist, kann nicht so einfach bewertet werden, doch ist mit ECC-Test immerhin eine Früherkennung möglich, wenn man dem Harddiskcontroller das ECC-Byte verbietet.

Handling

Aufgrund der sehr engen Toleranzen bei Kopf, Platten und Servo reagieren Festplatten sehr empfindlich auf Stöße, Staub und Temperaturschwankungen. In keinem Fall sollte man das Gehäuse einer Festplatte öffnen, da dann Staub eindringt, der unweigerlich beim nächsten Start Kopf und Platten irreparabel beschädigt. Es soll aber Leute geben, die ihre vermeintlich defekte Festplatte öffneten, ein wenig beim Steppen zuguckten, sie wieder zuschraubten und anschließend steif und fest behaupteten, sie ginge nun gar nicht mehr, weil sie ja vorher schon nicht funktionierte…

Im stehenden Zustand verkraften Harddisks schon mal einen leichten Stoß - im Betrieb hat dies i.d.R. einen Head-Crash zufolge, nach dem man die Kiste am besten wegwirft. Daher gilt: im Betrieb befindliche Geräte, die eine Festplatte beinhalten, in keinem Fall bewegen!