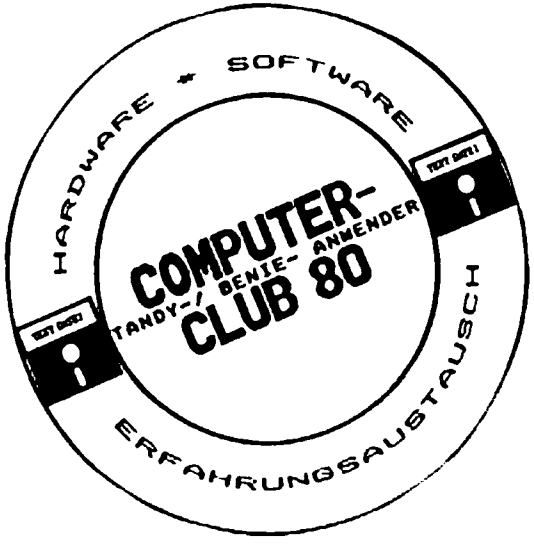


SONDERINFO

FLOPPY-CONTROLLER

1771
179X
279X

Die Programmierung der Floppy-Controller 1771, 179x, 279x

von Gerald Schröder

In den ersten TRS-80- und Genie-Systemen wurde der Floppy-Controller WD-1771 eingesetzt (WD=Western Digital, kann auch SAB=Siemens sein). Es folgte der 1791, der auch doppelte Schreibdichte verarbeiten konnte. Die letzten Systeme (Genie IIs und IIIs) setzten den 2791 ein, den vorläufigen Schlußpunkt dieser Serie. Auch in CP/M-Rechnern dürften diese Controller zu finden sein. Übrigens sind der 1791 und der 2791 Mitglieder einer Familie (179x bzw. 279x genannt) von mehreren Controllern, die sich nur in Kleinigkeiten unterscheiden. Auf die Unterschiede werde ich noch eingehen, auch darauf, daß es zwischenzeitlich eine Serie 179x-02 gab. Allerdings habe ich keine Unterlagen über den Ur-Großvater 1761 und einen 1781, so daß ich mich über die beiden ausschweigen muß.

Der Otto-Normal-Benuter des Newdos (oder GDOS oder CP/M) macht sich nun wenig Gedanken über die Programmierung dieser Teile. Sobald jemand (wie ich) ein CP/M-BIOS selber schreiben muß, sollte er sich damit beschäftigen. Ich habe das dadurch umgangen, daß ich einfach aus dem GDOS und einem anderen CP/M-BIOS die Floppy-Routinen abgeschrieben habe. Aber ich stellte mir die Frage: Was kann das Ding eigentlich alles? Und somit habe ich Grossers "DOS-Buch" genommen und mir Teile vom Datenblatt des 179x übersetzt. Nachdem der Artikel schon fertig war, bekam ich von Helmut noch die Datenblätter zu den Controllern 1771, 179x-02, (SAB-)279x und (WD-)279x-02 (wobei die beiden letzten sich nach den mir vorliegenden Daten scheinbar nicht unterscheiden). Dadurch ist der Artikel noch etwas gewachsen.

Das Ergebnis ist eine Anleitung zum Programmieren bzw. Benutzen dieser Floppy-Controller. Der Aufbau und Anschluß (also der Hardware-Krempel) ist mir dabei egal. Ich werde versuchen, alle Möglichkeiten zu beschreiben, die sich bieten, wenn auch einige durch die Hardware bei einigen Rechnern verbaut wurden (vor allem der DMA-Zugriff). Aber das kann ja geändert werden.

Da der Artikel ziemlich lang geraten ist, will ich kurz den Aufbau erklären. In Kapitel 1 wird gezeigt, wie man an den Controller überhaupt rankommt (unabhängig vom verwendeten Computer) und was er so leistet. In diesem Kapitel dürfen sich auch die Hardware-Freaks mal angesprochen fühlen, einige der Möglichkeiten der Controller besser auszunutzen.

Kapitel 2 liefert eine Anleitung zum Programmieren des Controllers und sagt, wie er darauf reagiert. Allerdings werde ich mich dabei auf den Endpunkt der Fahnenstange, den 2791, beschränken. Die Unterschiede zu den anderen werden angemerkt oder in Kapitel 4 beschrieben. Für den Profi sind eigentlich nur die Tabellen interessant, die sich jeder zum Programmieren auf's Laufwerk legen sollte (s. Anhang).

Die Formatierung einer Diskette ist in Kapitel 3 das Thema. Das mag trivial erscheinen (gebe ich einfach "FORMAT" ein), ist es aber nicht, sobald wir uns genau ansehen, was zu passieren hat und was neben den "normalen" Formaten noch möglich ist.

Kapitel 4 bringt dann die Unterschiede zwischen zwischen den einzelnen Controllern. Dort beschreibe ich auch, was sie nicht können. Evtl. solltet Ihr Euch den Teil für Euren Controller zuerst durchlesen, damit Ihr in Kapitel 2 schon wißt, ob Ihr das für Euren Controller überhaupt gebrauchen könnt.

Wie die Programmierung bei den TRS-80s und ihren Kompatiblen konkret aussieht, sage ich nicht. Weil das sehr gut im Grosser steht (Kapitel 1). Wer das in natura sehen will, sollte sich im Grosser im Kapitel 3 die Seiten 24 bis 31 bzw. im SYS0 4630h-47E2h ansehen.

1. Die Verbindung zum Prozessor

Gelegentlich muß ich hier näher auf die Hardware eingehen, aber das werde ich kurz halten. Für den Programmierer ist das nicht so wichtig. Interessant dagegen ist, daß

1. die Daten parallel übertragen werden (8 Bits),
2. der Controller den DMA-Betrieb unterstützt,
3. zur Ansteuerung des Controllers 4 Register zur Verfügung stehen,
4. nach Beendigung jedes Kommandos ein Interrupt ausgelöst wird,
5. der Controller einiges nicht macht bzw. kann.

Der erste Punkt dürfte kaum weiter überraschen, nur wird damit ein "Handshaking" nötig, d.h. der Controller muß über irgendeine Extra-Leitung sagen, wann er Daten holen oder abgeben möchte. Die einfache (und beim TRS-80 samt Nachfolgern genutzte) Möglichkeit ist, ständig den Status abzufragen (wie, dazu weiter unten mehr) und sobald das Bit 1 (DRQ=Data Request) gesetzt wird, die Daten abzuholen bzw. zu übergeben. Dieses Verfahren wird "Polling" genannt.

Mir gefällt die zweite Möglichkeit besser: Ein DMA-Controller (DMA=Direct Memory Access) übernimmt die Übertragung direkt aus dem bzw. in das RAM, während der Prozessor sich langweilt oder andere Sachen macht (Wurzel aus 2 auf 10 Stellen genau berechnen oder solchen Quatsch). Der Floppy-Controller verfügt dazu über eine Extra-Leitung, sinningerweise auch DRQ genannt, die dasselbe leistet wie das DRQ-Bit im Statusbyte, aber direkt an einen DMA-Controller angeschlossen werden kann. Damit hätten wir dann drei Prozessoren (bzw. einen Hauptprozessor und zwei Controller) im System, was den TRS-80-Leuten wohl zu aufwendig war und schätzungsweise nur bei entsprechend aufwendiger Programmierung drumherum was bringt (ansonsten "langweilt" sich der Proz eh' nur rum). Aber die Möglichkeit ist vorhanden, wenn a) sich ein DMA-Controller im System befindet oder b) der HD-64180 als Haupt-Prozessor benutzt wird (der hat nämlich schon zwei DMA-Controller "on chip", werden also kostenlos mitgeliefert).

So, damit hätten wir Punkt zwei auch abgehakt. Kommen wir zu Punkt drei, den Registern. Davon haben wir 4 zur Verfügung, was bedeutet, daß nur 4 I/O-Ports oder (Memory-Mapped-) Speicheradressen dafür verbraten werden. Zur Adressierung werden nur 2 Bit benötigt (Adressen 00, 01, 10, 11). Wo diese 4 Adressen nun wirklich liegen, bleibt dabei dem Hardware-Entwickler überlassen. Die Bedeutung der Register:

00 Status- und Kommando-Register
01 Track-Register
10 Sektor-Register
11 Daten-Register

Die letzten drei Register dürften leicht zu verstehen sein: Das Track-Register nimmt eine Track-Nummer von 0 bis 244 auf, das Sektor-Register eine Sektor-Nummer von 0 bis 244. Beim Schreiben wird dem Controller mitgeteilt, auf welchem Track das Laufwerk gerade steht bzw. welcher Sektor als nächstes bearbeitet werden soll. Beim Lesen werden die aktuellen Werte ausgegeben. Bis jetzt werden nur Track-Nummern bis 80 und Sektor-Nummern bis ca. 40 benutzt. Die Laufwerke können also noch etwas leistungsfähiger werden, als sie es heute schon sind. 244 (F4h) ist eine Grenze, die durch die Format-Prozedur bestimmt wird (s. Kapitel 3). Das Daten-Register dient zum Übertragen der Daten zwischen Prozessor (oder DMA-Controller) und Floppy-Controller (u.a. ist auch der nächste zu suchende Track ein Datum, s. das Seek-Kommando). Nur das erste Register (Kommando und Status) ist etwas "Besonderes": Wenn etwas in dieses Register geschrieben wird, nimmt der Floppy-Controller das Byte als Kommando entgegen (sowas wie "Schreibe

Sektor" usw.), beim Lesen gibt er aber seinen momentanen Status zurück, wobei die Bedeutung der einzelnen Bits stark davon abhängt, welches Kommando vorher gegeben wurde. Dieses Register hat also zentrale Bedeutung für den gesamten Betrieb und jedesmal, wenn im folgenden etwas von "Kommandos geben" oder "Status abfragen" steht, ist dieses Register im Spiel.

Kommen wir zu Punkt vier. Sobald der Floppy-Controller ein Kommando erhalten hat (durch Schreiben eines Bytes in Register 00), fängt er an zu werkeln; einzige Ausnahme: beim Kommando "Force Interrupt" hört er auf, das vorherige Kommando auszuführen. Nachdem das Kommando beendet ist (ob zur allgemeinen Zufriedenheit oder mit einem Fehler, ist dabei egal) wird ein Interrupt ausgelöst; dafür gibt es wie für DRQ auch eine eigene Leitung (hier: zum Prozessor oder einem Interrupt-Controller), genannt INTRQ (Interrupt Request). Diese Möglichkeit wird im TRS-80 auch nicht genutzt, kann aber eingestellt werden. Was für einen Sinn könnte das Ding haben? Nun, wie schon gesagt, kann der Proz weiterarbeiten, während der Floppy-Controller ein Kommando ausführt, z.B. einen Sektor an den DMA-Controller weitergibt. Aber der Proz sollte irgendwann darüber informiert werden, daß das Kommando ausgeführt wurde und ob Fehler aufgetreten sind. Dann wird ein Interrupt ausgelöst und der Proz kann sich ansehen, was inzwischen passiert ist, während er sich mit der Wurzel aus 2 herumgeschlagen hat.

Auf zum letzten Punkt: Was leistet der Floppy-Controller nicht? Das hängt nun ganz von dem Controller ab. Zum Beispiel kann der 1771 nur SD (=Single Density, einfache Schreibdichte) lesen oder schreiben und kann mit DD (=Double Density, doppelte Schreibdichte) nichts anfangen. Die Nachfolger können teilweise zwischen SD und DD umschalten, was aber über eine Extra-Logik (und Extra-Ports oder -Adressen) dem Controller mitgeteilt werden muß. Wie dies passiert? Das weiß nur der jeweilige Hardware-Entwickler.

Außerdem kennt der 1771 keine DS-Disketten (=Double Sided, doppelseitig), während die Nachfolger teilweise überprüfen können, ob ein Track auf Seite 1 steht und einige sogar die Seite 1 des Laufwerks direkt anwählen können. Ansonsten muß dies eine Extra-Logik übernehmen, die wie bei der Anwahl von SD/DD natürlich wieder Ports/Adressen verbrät.

Eine Komplikation gibt es noch: Auf dem Markt sind heutzutage 8-, 5.25- und 3,5-Zoll-Laufwerke. Früher wurde einfach zwischen 8 Zoll und 5.25 Zoll unterschieden und die 3.5-Zoll-Drives zu den 5.25-Zöllern gesteckt. Für den Controller gibt es nur einen Unterschied: Die 8-Zoll-Laufwerke haben eine höhere Übertragungsrate (KBit/Sekunde) als die kleineren Laufwerke. Deshalb muß bei einigen Controllern der Takt zwischen 2 Mhz (8") und 1 Mhz (5.25") umgeschaltet werden, einige verlangen außerdem noch eine Benachrichtigung darüber, was nun anliegt, während einige mit 2 Mhz Takt zufrieden sind und ihn intern durch zwei teilen, wenn sie mit 5.25" arbeiten sollen. Der Grund für die Unterschiede ist darin zu suchen, daß je nach Controller verschiedene ZusatzChips (Data Separator und Write Precompensation) nötig sind oder deren Funktionen direkt in den Controller übernommen wurden. Aber nun kommt der Hammer: Irgendwann fiel einem Hersteller ein, daß mit der höheren Übertragungsrate auch auf 5.25"- und 3.5"-Laufwerken mehr gespeichert werden kann. Und schon hatten wir 8-Zoll-Laufwerke in der Größe 5.25" und 3.5", die heute bis 1.2 MB bzw. 1.44 MB aufnehmen können. Wenn im folgenden von 5.25" die Rede ist, bezeichnet dies einfach die niedrigere Übertragungsrate (Kapazitäten bis 720 KB sind typisch dafür, sowohl bei 5.25"- als auch bei 3.5"-Laufwerken).

Welches Diskettenlaufwerk soll der Controller nun ansprechen, wenn mehrere vorhanden sind? Dem Floppy-Controller ist das vollkommen schnurzpiepe. Auf den ersten Blick mag das etwas verwundern, aber so können theoretisch (vom FDC her) beliebig viele Laufwerke angeschlossen werden oder mehrere Disketten-Controller, z.B. für jedes Laufwerk einer, in dem System rumhängen. Im TRS-80 waren drei Laufwerke maximal vorgesehen, in den Nachfolgern wurde diese Zahl teilweise auf vier erhöht. Ich finde diese Einschränkung etwas kurzsichtig. Auf jeden Fall geht dafür bestimmt ein Extra-Port oder eine Memory-Mapped-I/O-Adresse verloren.

2. Die Kommandos für den Controller und seine Reaktion

Ein Kommando sollte nur an den Controller gegeben werden, wenn er nicht beschäftigt ist (Not Busy, Status-Bit 0 = 0). Einzige Ausnahme ist das Kommando "Force Interrupt", das immer gegeben werden kann. Welche Kommandos kann der Floppy-Controller nun ausführen? Entsprechend den Datenblättern der Floppy-Controller lassen sich die Kommandos in vier Klassen einteilen:

I. Kopf-Positionierung: Diese Kommandos setzen den Schreib-/Lesekopf des Laufwerks auf verschiedene Tracks.
II. Sektor-Zugriffe: Hier werden einzelne Sektoren gelesen oder geschrieben.
III. Adress- und Track-Zugriffe: Diese Kommandos dienen zum Lesen einer Adresse (ID-Feld) und dem Lesen oder Schreiben eines Tracks.
IV. Force-Interrupt: Hier gibt es nur das gleichnamige Kommando, das meist zum Unterbrechen anderer Befehle dient.

Im folgenden werden die einzelnen Kommandos der verschiedenen Klassen kurz erklärt. Eine Zusammenfassung in Tabellenform befindet sich im Anhang.

Wenn Zeiten angegeben werden (z.B. wie lange der Kopf zum Positionieren braucht), beziehen sich diese auf 8"! Bei 5.25" (bzw. bei der niedrigen Übertragungsrate) müssen die Zeiten jeweils verdoppelt werden! Hier gleich die erste Rechen-Übung: Nach jedem Kommando (bzw. dem Schreiben in das Kommando-Register) muß eine bestimmte Zeit gewartet werden, bevor der Status zum ersten Mal ausgelesen werden darf (sonst gibt's nur Müll). Das Status-Bit 0 ist nach 12 (SD) bzw. 6 (DD) Mikrosekunden OK. Die restlichen 7 Bits können frühestens nach 28 (SD) bzw. 14 (DD) Mikrosekunden (sinnvoll) abgefragt werden. Wie gesagt, verdoppeln sich im 5.25"-Modus die Zeiten.

Außerdem werden die beim 2791 möglichen Kommandos beschrieben. Unterschiede zu anderen Controllern werden in Kapitel 4 abgehandelt.

2.1 Klasse I: Positionieren des Schreib-/Lese-Kopfes

Alle Kommandos dieser Klasse dienen dazu, den Schreib-/Lesekopf über einem bestimmten Track der Diskette zu positionieren. Von der Step-Rate hängt es ab, wie schnell der Kopf bewegt wird; das Laden des Kopfes an die Disketten-Oberfläche (zum Schreiben oder Lesen) kann von diesen Kommandos gesteuert werden, wird aber ansonsten beim Schreiben oder Lesen automatisch vorgenommen.

Die Step-Rate r1/r0 hängt vom angeschlossenen Laufwerk-Typ ab. Die meisten Laufwerke sind so gut, daß sie eine Step-Rate von 3 Millisekunden vertragen (also r1/r0 = 00).

Die Flags Head Load und Verify sollten zusammen betrachtet werden. Das Flag h legt fest, ob der Kopf am Anfang des Kommandos auf die Oberfläche gesenkt wird; v gibt an, daß am Ende (nach einer Kunstpause von 15 ms) verglichen werden soll, ob der richtige Track erreicht wurde; gleichzeitig wird auch hier der Kopf geladen (sonst könnte die Track-Nr. nicht gelesen werden). Die Möglichkeiten:

h	v	Wirkung
0	0	der Kopf wird abgehoben, wenn er vorher geladen war
1	0	der Kopf wird am Anfang geladen
0	1	der Kopf wird am Ende geladen und der Vergleich durchgeführt
1	1	der Kopf wird am Anfang geladen und der Vergleich durchgeführt

Leider ist das nur die Theorie, denn in der Praxis scheren sich die Laufwerke kaum um das h-Flag, sondern laden den Kopf, wenn die Diskette eingelegt oder die Klappe geschlossen wurde, oder wenn der Motor anspringt (TRS-80 und Kumpanen) oder was-weiß-ich. Und das Verify geht in die Hose, wenn die logischen (in den Sektoren vermerkten) Track-Nummern nicht den physikalischen entsprechen (z.B. die Diskette beginnt mit dem logischen Track 1, der physikalisch trotzdem meist Track 0 heißt).

Der Kopf wird (theoretisch) abgehoben, wenn a) ein Kommando mit h=0 und v=0 kommt oder b) während 15 Disk-Umdrehungen nach dem letzten Kommando kein neues gegeben wird.

Zum Verify (v=1): Der Controller versucht 5 Disk-Umdrehungen lang, das ID-Feld eines Sektors zu lesen. Wenn er es schafft (d.h. eine korrekte CRC (Checksumme) in einem ID-Feld findet), wird die gefundene Track-Nr. mit der aus dem Track-Register verglichen. Stimmen sie überein, wird das Kommando ohne Fehler beendet. Stimmen sie nicht überein oder findet er 5 Umdrehungen lang kein ordentliches ID-Feld, wird ein "Seek Error" im Status-Register angezeigt und das Kommando beendet. Übrigens ist das Beenden eines Kommandos immer mit einem Interrupt verbunden, was ich ab jetzt nicht mehr wiederholen werde.

Bei den Step-Kommandos gibt es noch das Update-Flag. Wenn es gesetzt ist, wird das Track-Register beim Steppen in- oder dekrementiert. Ansonsten bleibt es unverändert.

2.1.a Restore (=Seek Track 0)

Dieses Kommando bewirkt, daß der Kopf über Track 0 positioniert und das Track-Register auf 0 gesetzt wird. Wenn das Laufwerk nach 255 Steps immer noch nicht Track 0 erreicht hat, bricht der Controller die Sache mit der Fehlermeldung "Seek Error" ab (physikalisch Track 0, ganz außen).

2.1.b Seek

Im Track-Register muß sich die aktuelle Track-Nr. und im *Daten-Register* die gewünschte Track-Nr. befinden. Der Controller steppt dann automatisch solange, bis die gewünschte Track-Nr. erreicht ist. Achtung: Die logische Track-Nr. muß nicht der physikalischen entsprechen! Dann gibt es eine "falsche" Fehlermeldung, wenn "Verify" gewählt wurde!

2.1.c Step

Dieses Kommando führt einen Step in die Richtung aus, in die beim letzten Mal gesteppt wurde.

2.1.d Step In

Der Kopf wird in in Richtung auf Track 80 zu bewegt, und zwar um einen Schritt.

2.1.e Step Out

Der Kopf wird in Richtung auf Track 0 zu bewegt, wieder um einen Schritt.

2.2 Klasse II: Schreiben und Lesen eines Sektors

Bei beiden Kommandos wird zuerst der Kopf an die Diskettenoberfläche geladen. Wenn das Delay-Flag e gesetzt ist, folgt eine Warteschleife von 15 ms, bevor auf die Bestätigung durch das Laufwerk gewartet wird (e sollte gesetzt sein, warum auch immer).

Dann sucht der Controller 5 Disk-Umdrehungen lang nach dem richtigen Sektor. Dazu liest er die ID-Felder, vergleicht die Track-Nr. mit dem Track-Register, die Sektor-Nr. mit dem Sektor-Register und prüft die CRC-Bytes. Wenn das Side-Compare-Flag c gesetzt ist, vergleicht der Controller auch die Seitennummer im ID-Feld mit dem Side-Flag s. Wurde nicht der richtige Sektor gefunden, endet das Kommando mit einem "Record Not Found Error".

Nun wird der Sektor gelesen oder geschrieben. Sollte das Multiple-Record-Flag m gesetzt sein, inkrementiert der Controller das Sektor-Register und fängt von vorne an (ohne das Laden des Kopfes). Beendet wird das Kommando erst dann, wenn a) das Kommando "Force Interrupt" gegeben wird oder b) die Sektor-Nummer zu groß wird und der Controller mit einem "Record Not Found Error" abbricht.

2.2.a Read Sector

Nach der CRC-Prüfsumme im ID-Feld muß innerhalb der nächsten 30 Bytes (Single Density) bzw. 43 Bytes (Double Density) die Daten-Adreßmarke (Data Address Mark) folgen, sonst wird der Fehler "Record Not Found" ausgegeben (s. Kapitel 3: Formatierung).

Wenn ein Datenbyte abholbereit ist, setzt der Controller das DRQ-Bit (im Status-Byte) bzw. die DRQ-Leitung. Der Prozessor (oder der DMA-Controller) muß nun das Datum aus dem Daten-Register lesen. Tut er das nicht, bevor das nächste Byte abholbereit ist, setzt der Controller die Fehlermeldung "Lost Data".

Am Ende des Sektors steht noch eine Prüfsumme (CRC), die der Controller überprüft. Stellt sich ein Fehler heraus, setzt er die Fehlermeldung "CRC Error". Zuletzt gibt der Controller in Bit 5 des Status-Bytes bekannt, welche Adreßmarke er gefunden hat: 1 bedeutet "Deleted Data Mark", 0 "Data Mark".

2.2.b Write Sector

Wenn der richtige Sektor gefunden wurde, überspringt der Controller zuerst 11 Bytes (Single Density) bzw. 22 Bytes (Double Density). In dieser Zeit muß der erste DRQ vom Prozessor oder DMA-Controller beantwortet worden sein, sonst bricht der Controller mit der Fehlermeldung "Lost Data" ab. Ansonsten schreibt der Controller jetzt 6 bzw. 12 Null-Bytes und dann die Adreßmarke (Bit 0=1: Deleted, Bit 0=0: Data Mark). (s. Kapitel 3).

Nun werden die Daten-Bytes geschrieben. Wenn ein DRQ nicht rechtzeitg vom Prozessor (DMA-Controller) wahrgenommen wird, schreibt der Controller ein Null-Byte und setzt die Fehlermeldung "Lost Data", bricht aber nicht ab!

Am Ende des Sektors wird die vom Controller intern errechnete Prüfsumme (CRC, 2 Bytes) auf die Diskette geschrieben, gefolgt von einem FEh.

2.3 Klasse III: Adresse lesen, Track lesen oder schreiben

2.3.a Read Address

Der Controller sucht sich das nächste ID-Feld, liest es und gibt es an den Prozessor aus. Es handelt sich um sechs Bytes, die der Proz sich nach dem jeweiligen DRQ abholen muß. Die Bedeutung der Bytes:

Byte Nr.	Bedeutung
1	Track-Nr.
2	Seiten-Nr. (nur Bit 0)
3	Sektor-Nr.
4	Sektor-Länge (nur Bits 1/0: 00=128, 01=256, 10=512, 11=1024 Byte)
5	CRC 1 (Prüfsumme)
6	CRC 2 "

Die Prüfsumme wird auch vom Controller überprüft und er setzt die Fehlermeldung "CRC Error", wenn sie nicht ok ist. Außerdem wird die Track-Nr. auf jeden Fall in das Sektor-Register geschrieben. Die kann das Programm mit der "echten" im Track-Register vergleichen.

2.3.b Read Track

Ein kompletter Track (von Indexloch bis Indexloch) wird eingelesen, mit allem Drum und Dran (Gaps usw., s. Kapitel 3). Es wird keine Prüfung der CRCs vorgenommen. Beim Lesen können Fehler auftreten! Dieses Kommando eignet sich nicht zum Kopieren einer Diskette, sondern nur zum Analysieren.

2.3.c Write Track

Ein kompletter Track (von Indexloch bis Indexloch) wird geschrieben. Wenn das erste DRQ nicht rechtzeitig beantwortet wird, endet das Kommando sofort mit einem "Lost Data Error". Wenn später ein Byte nicht rechtzeitig angeliefert wird, schreibt der Controller ein Null-Byte und setzt den Fehler "Lost Data". Adreßmarken und CRCs werden vom Controller auf spezielle Kommandos im Datenstrom hin geschrieben. Näheres dazu im Kapitel 3 "Formatierung einer Diskette".

2.4 Klasse IV: Force Interrupt

Dieses Kommando unterbricht normalerweise andere Kommandos und erzeugt einen Interrupt, wobei der Zeitpunkt von den Bits 3-0 abhängt: Bit 3=Interrupt sofort, Bit 2=beim Index-Loch, Bit 1=Übergang von Ready auf Not-Ready, Bit 0=Übergang von Not-Ready auf Ready. Es können auch mehrere gesetzt werden. Ist keins gesetzt, wird kein Interrupt erzeugt. Wenn Bit 3 gesetzt war (Kommando D8h), muß danach das Kommando D0h (kein Interrupt) folgen, sonst gibt es beim nächsten Kommando Probleme!

Dieses D8h ist gut zu gebrauchen, wenn der Controller im Interrupt-Betrieb läuft und nun ein Kommando abgebrochen werden soll: Mit D8h wird die Interrupt-Routine sofort angesprungen und man erfährt dann, was nicht geklappt hat. Dann muß aber ein Kommando der Interrupt-Routine D0h sein.

Das Status-Byte wird nicht geändert, wenn vorher ein Kommando ausgeführt wurde (vorher Busy=1). Nur Busy wird gelöscht. Wenn der Controller vorher nichts zu tun hatte, wird das Status-Byte neu gesetzt, wobei die Bits die Bedeutung wie bei den Kommandos der Klasse I haben (s. Tabelle 3a).

Nach einem Force Interrupt muß 8 Mikrosekunden (DD) bzw. 16 Mikrosekunden (SD) gewartet werden, bevor das nächste Kommando gegeben werden darf, sonst gibt es Ärger! Nicht vergessen: Bei 5.25" verdoppeln sich die Werte!

3. Die Formatierung einer Diskette

Ich möchte vorausschicken, daß ich meine Erkenntnisse nur durch Lesen und Nachdenken erworben, aber nicht am "lebenden Objekt" überprüft habe. Ich kann also keine Garantie für die folgenden Informationen übernehmen. Leider sind die Angaben, wie die Formatierung zu geschehen hat, in den Datenblättern etwas dürftig und teilweise widersprüchlich. Ich habe deshalb hier etwas meine Phantasie spielen lassen.

Die Einteilung einer Diskette beginnt bei den Tracks, die jeweils einen "Kreis" auf der Diskettenoberfläche bilden. Normalerweise können heutige Laufwerke bis zu 40 oder 80 Tracks bearbeiten. Nun kann ein Track aber sowohl auf der Vorder- als auch auf der Rückseite einer Diskette stehen, wenn das Laufwerk zwei Köpfe hat. Deshalb ist noch eine Seiten-Kennung (0 oder 1) nötig.

Jeder Track umfaßt (meistens) mehrere Sektoren. Jeder Sektor bekommt eine Nummer und muß eine bestimmte Länge haben. Der 2791 unterstützt Sektor-Längen von 128, 256, 512 oder 1024 Bytes, d.h. soviele Daten-Bytes können in einem Sektor gespeichert werden. Der "echte" Sektor auf der Diskette enthält außer den Daten eine Menge Zusatzinformationen, so daß er sehr viel länger ist; in den Datenblättern wird er in zwei Felder geteilt: das ID-Feld (ID=Identify), das alle Infos über Track-Nr, Sektor-Nr., Seiten-Nr. und Anzahl der Datenbytes enthält, sowie das Datenfeld, das aus den Datenbytes besteht. Dazu kommen bei jedem Feld Prüfsummen und Adreßmarken sowie einige GAPs (Lücken).

Etwas sollte hier noch einmal klargestellt werden: Dem Controller ist es ziemlich egal, wie eine Diskette aussieht, solange sie einigen kleinen Konventionen genügt, damit er die enthaltenen Bytes richtig interpretiert. Es ist also keinesfalls nötig, daß eine Diskette homogen ist, wie es uns die Angabe "80/DS/DD mit Sektorlänge 1024" weismachen will. Es kann sein, daß nebeneinanderliegende Tracks verschiedene Schreibdichten aufweisen, mal ein Track ausgelassen wurde, auf den Tracks Sektoren mit unterschiedlicher Länge stehen usw. So ist es denkbar, daß der Track Nr. 12 (physikalisch) die Nummer 120 trägt und nur einen Sektor enthält, der 128 Byte lang ist, die Nummer 150 und eine Kennung für die Rückseite hat, obwohl er auf der Vorderseite steht.

Zur Formatierung einer Diskette wird das Kommando "Write Track" benutzt. Der Controller erwartet dann, daß ihm ein kompletter Track, Byte für Byte, übergeben wird, so daß er ihn auf die Diskette schreiben kann. Allerdings schreibt er nicht jedes Byte so auf die Diskette, wie es übergeben wird, sondern einige spezielle Bytes erkennt er als Befehle und schreibt etwas anderes. Diese Bytes dürfen deshalb nicht als Daten oder Sektor-Nummer o.ä. benutzt werden, weil der Controller sonst durcheinander kommen würde. Die gefährlichen" Bytes tragen die Nummern F5h bis FEh.

Diese Daten sind dem Controller zu übergeben:
(modifizierte Backus-Naur-Form)

```
Track ::= GAP4(Density)        (Anfangslücke, abhängig von der Schreibdichte)
          FCh                  (Index-Markierung)
          GAP1(Density)        (Lücke nach Index-Markierung)
          {Sektor}             (die Sektoren)
          {Füllbyte(Density)}  (Auffüllen bis zum Trackende)

Sektor::= GAP3(Density)        (Lücke am Anfang)
          FEh                  (ID-Adreß-Marke)
          TrackNr              (die Track-Nummer)
          SeitenNr             (die Seiten-Nummer)
          SektorNr             (die Sektor-Nummer)
          SektorLänge          (die Kennung der Sektor-Länge)
          F7h                  (CRC erzeugen)
          GAP2(Density)        (Lücke zwischen ID- und Daten-Feld)
          Adreßmarke           (Adreßmarke)
          {Data}128*2**SektorLänge
                               (soviele Daten-Bytes, wie bei SektorLänge
                                angegeben)
          F7h                  (CRC erzeugen)
          [GAP1(Density)]      (Lücke am Ende des Sektors, kann wegfallen)

Density      ::= SD | DD       (einfache/doppelte Schreibdichte)
Füllbyte(SD) ::= FFh           (Füllbyte bei Single Density)
Füllbyte(DD) ::= 4Eh           (Füllbyte bei Double Density)
TrackNr      ::= 00h | 01h | ... | F4h | FFh
SeitenNr     ::= 00h | 01h
SektorNr     ::= 00h | 01h | ... | F4h | FFh
SektorLänge  ::= 00h | 01h | 02h | 03h
              (für 128,  256,  512, 1024  Bytes/Sektor)
Adreßmarke   ::= DeletedDM | DataMark
DeletedDM    ::= F8h
DataMark     ::= FBh
Data         ::= 00h | 01h | ... | F4h | FFh  (meist wird E5h benutzt)

GAP4(SD)     ::= {Füllbyte}16                 (Lücke bei Single Density)
GAP4(DD)     ::= {Füllbyte}16 [{F6h}3]        (Lücke bei Double Density)
GAP3(SD)     ::= {Füllbyte}10 {00h}4          (usw.)
GAP3(DD)     ::= {Füllbyte}24 {00h}8  {F5h}3  (oder 16 Füllbytes?)
GAP2(SD)     ::= {Füllbyte}11 {00h}6          (genau 6 Nullen!)
GAP2(DD)     ::= {Füllbyte}22 {00h}12 {F5h}3  (genau 12 Nullen und 3xF5h!)
GAP1(SD)     ::= {Füllbyte}16
GAP1(DD)     ::= {Füllbyte}32                 (oder 16?)
```

Legende:

x ::= y	x wird ersetzt durch y
x(z)	x ist abhängig von z
xxh	Byte xx (in Hexadezimal) einsetzen
(qqq)	qqq ist ein Kommentar
x | y	x oder y
[x]	setze x keinmal oder einmal ein
{x}	setze x keinmal, einmal oder beliebig oft ein
{x}Wert	setze x hier "Wert"-mal ein, z.B. {00h}5 = setze C0h fünfmal ein

Interessant sind vor allem die "GAP"s. Das sind Lücken, die an strategisch wichtigen Stellen eingebaut werden müssen. Warum? Darüber schweigen sich die Datenblätter aus. Es scheint so, als ob dort die Synchronisation stattfindet. Der dafür "verschwendete" Platz sollte möglichst klein gehalten werden, damit viele Sektoren auf den Track passen, bei gleichzeitig möglichst schnellem und fehlerfreien Zugriff. Die Datenblätter weisen die oben angegebenen Werte als Minimalwerte aus. Es kann also damit herumexperimentiert werden. Z.B. besteht GAP4 in einem Beispiel in einem Datenblatt (für DD) aus 80x4Eh, 12x00h und 3xF6h und die Index-Markierung fehlt oft ganz. Auf jeden Fall sollten immer die 3xF5h (GAP3 und GAP2) geschrieben werden, denn die scheinen immens wichtig zu sein (warum auch immer); ebenso die 6/12 Nullen in GAP2 (s. auch Read/Write Sector!).

Im Grosser steht auf den Seiten 1-17 und 1-18, welches Format das Newdos 80 benutzt. Grosser sagt dort, daß die Größe der GAPs von der der Anzahl und Größe der Sektoren abhängt. In einem alten Format-Programm von TCS habe ich jedenfalls wilde Zahlen gefunden, die teilweise auch unter den als Minimum angegebenen Werten lagen und gerade für GAP2, wo sie exakt gelten sollten (nach dem Datenblatt), mit steigender Sektorgröße in die Höhe schossen (1024 Bytes/Sektor in DD: 88x4Eh, 48x00h, 3xF5h).

An Sektoren können soviele erzeugt werden, wie eben auf den Track passen. Wie kann festgestellt werden, ob der Track zu lang war? Ganz einfach: Dann erzeugt der Controller beim "Write Track" keine DRQs mehr, obwohl noch Daten zu übertragen sind. Zu kurz darf die Sache dagegen immer sein. Der Rest des Tracks sollte dann mit den Füllbytes aufgefüllt werden (solange der Controller noch mit DRQs nach Futter verlangt), damit es keine Probleme beim späteren Lesen gibt (da steht sonst ja nur Müll).

Jeder Sektor enthält die Angaben, auf welchem Track und welcher Seite er steht. Es besteht also die Möglichkeit, hier Mist zu bauen, indem zum Beispiel jeder Sektor eine andere Tracknummer bekommt usw. Das stört den Controller wenig, nur beim Arbeiten muß man dann später ziemlich aufpassen. Die Sektorlänge ist auf die vier Größen (binär) 00, 01, 10 und 11 für die Sektorlängen 128, 256, 512 und 1024 beschränkt. Es müssen auf jeden Fall (bei "Data") soviele Datenbytes geschrieben werden, wie der Sektor gemäß "SektorLänge" lang sein soll (Vorsicht! Keine Bytes von F5h bis FEh; die dürfen erst bei "Write Sector" auftauchen!). Deshalb diese seltsame Rechenformel ("**" soll "hoch" heißen).

Da die Bytes F5h bis FEh besondere Bedeutung haben, sind sie als Track-, Sektor- und Seitennummer sowie als Sektorlängenkennung und Datum (im Datenfeld) tabu! Hier eine Tabelle, die zusammenfaßt, was der Controller mit welchen Bytes macht:

Byte	Bedeutung bei Single Density	Double Density	
00-F4, FF	00-F4 bzw. FF schreiben		
F5	nicht erlaubt	A1* schreiben, CRC-Init	
F6	nicht erlaubt	C2* schreiben	
F7	2 CRC-Bytes erzeugen		
F8-FB	schreiben, CRC-Init	schreiben	(Data Address Mark)
FC	schreiben (")	"	(Index Address Mark)
FD	" (")	"	(frei)
FE	" , CRC-Init	schreiben	(ID Address Mark)

Die Bytes 00h-F4h und FFh werden einfach auf die Diskette geschrieben, in DD auch noch die Bytes F8h-FEh. Mit F8h-FBh sowie FEh (SD) bzw. nur mit F5h (DD) wird vom Controller der Prüfsummenzähler gesetzt, der dann beim Empfang von F7h zwei CRC-Bytes (Prüfsumme, Cyclic Redundancy Check) auf die Diskette schreibt. Die Bytes A1* und C2* (nur in DD) sind nicht die normalen Bytes dieser Art. Scheinbar kann der Controller an ihnen feststellen, wo er sich gerade befindet, weil hier das Taktsignal verändert wurde (Synchronisation).

4. Die anderen Controller

Erstmal eine Übersicht über alle mir (von den Datenblättern her) bekannten Controller und ihre Eigenschaften:

Familie -->		179x-01				179x-02						279x(-02)			
Eigenschaft	1771	91	92	93	94	91	92	93	94	95	97	91	93	95	97
Double Density		x		x		x		x		x	x	x	x	x	x
inv. Datenbus	x	x	x			x	x			x		x		x	
PLL-Daten-Sep.												x	x	x	x
Write Precomp.		x	x	x	x	x	x	x	x	x	x	x	x	x	x
Side Compare		x	x	x	x	x	x	x	x	x	x	x	x	x	x
Side Select										x	x			x	x
2 Mhz -> 1 Mhz												x	x		

Fangen wir bei dem 1771 an. Vorher gab es scheinbar noch einen 1761, über den ich aber keine Unterlagen habe.

1771

Die Zeit, bis der Kopf sich auf die Diskette gesetzt hat, wird hier mit 10 ms (statt 15) angegeben. Die Step-Raten betragen 6/6/10/20 ms (alles bei 8"). Der Controller hat zwar einen internen Daten-Separator (der die vom Laufwerk gelieferten Daten entschlüsseln kann), aber es wird ein externer empfohlen.

Bei den Kommandos ergeben sich folgende Unterschiede:

```
 Read Sector  1 0 0 m b e 0 0
 Write Sector 1 0 1 m b e a1 a0
  b=1: Sektor-Länge 128/256/512/1024 Bytes (wie normal bei allen späteren)
  b=0: Sektor-Länge von 16 bis 4096 Bytes, 01=16, 02=32 usw., 00=4096, die
       Längen F7h bis FEh sind nicht erlaubt
  a1 und a0 geben Daten-Adreßmarke an:
     0 0 = FBh, normal
     0 1 = FAh, Benutzer-definiert
     1 0 = F9h, Benutzer-definiert
     1 1 = F8h, deleted
     (alle späteren Controller kennen nur FBh und F8h)
  e=1: 10 ms Delay
 Read Track  1 1 1 0 0 1 0 s*
 Write Track 1 1 1 1 0 1 0 0
  s*=0: Synchronisation zu den Adreßmarken durchführen
  (es fehlt Flag e)
```

Beim Lesen eines Sektors muß die Daten-Adreßmarke spätestens 28 Bytes nach dem Ende des ID-Feldes gefunden werden. Im Statusregister zeigen die Bits 5 und 6 die gefundene Adreßmarke an (Bit 5=a_0, Bit 6=a_1).

Beim Formatieren muß GAP2 17 Bytes lang sein, wovon mindestens die letzten 6 Bytes 00h sein müssen. 00h sind statt FFh als Füllbytes erlaubt. Vor jeder Adreßmarke muß mindestens ein Nullbyte stehen. Empfohlen wird, daß jedes GAP mindestens 17 Bytes lang ist und mindestens 6 Nullen am Ende enthält. Die Index-Adreßmarke kann (wie bei allen anderen Controllern auch) fehlen.

Über den direkten Nachfolger des 1771, den 1781, habe ich wieder keine Daten.

179x-01

Die Nachfolger-Serie 179x-01 besteht aus 4 Controllern (1791/2/3/4). Zwei davon können Double Density schreiben und lesen. Zwei sind mit einem "wahren" Datenbus ausgerüstet (normal ist scheinbar der invertierte, bei dem ein 0-Signal eine logische 1 anzeigt). Alle Controller sind hier bereits mit einer "Write Precompensation" ausgerüstet (was immer das auch sein mag). Sie benötigen aber alle einen externen Daten-Separator.

Die Kommandos und Reaktionen entsprechen denen des oben beschriebenen 2791. Auch hier kann die Index-Markierung beim Formatieren weggelassen werden, aber in SD darf nur FFh als Füllbyte benutzt werden (in DD sowieso nur 4Eh). Ab hier kennen die Controller übrigens doppelseitige Disketten und liefern (zumindest) einen automatischen Vergleich, ob der Sektor auf der richtigen Seite steht. Aber die Controller selbst können dem Laufwerk (noch) nicht mitteilen, welche Seite zu benutzen ist.

179x-02

Diese Neuauflage der ersten Serie führt zwei neue Controller ein (1795/7). Die beiden können dem Laufwerk mitteilen, welche Seite angewählt wurde. Außerdem prüfen sie automatisch, ob auch die im ID-Feld eingetragene Seitennummer der angewählten Seite entspricht. Dieser Vergleich kann nicht abgeschaltet werden! Außerdem können sie die Bedeutung des Sektorlängen-Bytes etwas verändern. Übrigens ist bei diesen beiden die Null als Füllbyte in SD wieder erlaubt.

Folgende Kommandos ändern sich somit:

Read Sector	1	0	0	m	L	e	U	0
Write Sector	1	0	1	m	L	e	U	a_0
Read Address	1	1	0	0	0	e	U	0
Read Track	1	1	1	0	0	e	U	0
Write Track	1	1	1	1	0	e	U	0

L=0: Sektor-Längen=256/512/1024/128 (was immer das auch soll)
L=1: Sektor-Längen=128/256/512/1024 (normal)
U=0: Seite 0 angewählt (und mit Seiten-Bit abgeglichen)
U=1: Seite 1 angewählt (und mit Seiten-Bit abgeglichen)

Kommen wir zum Schlußpunkt, der Serie

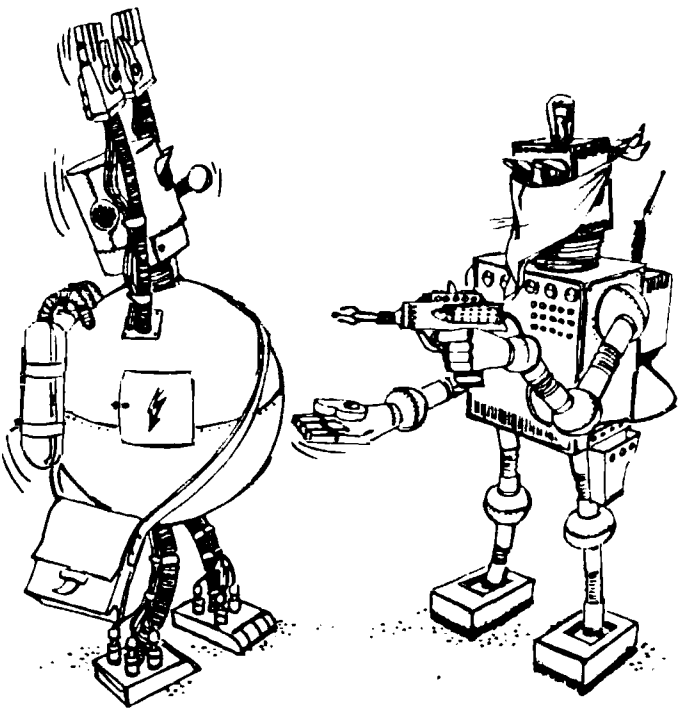
279x (-02)

Ich habe zwei Datenblätter vorliegen (eins von Western Digital und eins von Siemens) und mir scheint, daß die Siemens SAB-279x genau den WD-279x-02 entsprechen, nur daß Western Digital die Dinger noch in einem anderen Gehäuse anbietet (diese Quadrat-Form, "quad pack"). Auf jeden Fall haben die Dinger endlich einen (richtigen) Daten-Separator eingebaut, der "Phase Lock Loop Data Separator" genannt wird. Bei den anderen muß das Ding extra aufgebaut werden (bzw. es gibt sonst z.B. den WD-9216 dafür).

Bis auf eine Ausnahme scheinen die Dinger sonst der 179x-02-Familie zu entsprechen: Während der 2795 und der 2797 wie ihre Vorgänger eine Seiten-Auswahl ermöglichen, können der 2791 und der 2793 intern den angelegten 2 Mhz-Takt auf 1 Mhz herunterteilen. Vorher mußte immer eine externe Logik dies besorgen, wenn zwischen 8" und 5.25" umgeschaltet werden sollte. Hier ist dies dann nicht nötig (wohl aber noch bei den 2795/7). Außerdem müssen alle vier Controller darüber informiert werden, ob 8" oder 5.25" gewünscht wird, denn sonst arbeitet der Daten-Separator nicht richtig. Warum das nicht gleich alles zusammengepackt wurde, entzieht sich meiner Kenntnis. So sind jedenfalls beide Zweige (1/3 und 5/7) nichts Halbes und nichts Ganzes. Vielleicht gibt es ja inzwischen die Serie 379x. Darüber weiß ich aber nichts.

Literatur:

Hartmut Grosser: Das DOS-Buch, Aachen 1985, bes. Anhang A und Kapitel 1
Western Digital:
FD 1771-01 Floppy Disk Formatter/Controller
FD 179X-01 Floppy Disk Formatter/Controller Family
FD 179X-02 Floppy Disk Formatter/Controller Family
WD 279X-02 Floppy Disk Formatter/Controller Family
WD 9216-00/9216-01 Floppy Disk Data Separator - FDDS
Siemens:
SAB 279X Floppy Disk Formatter/Controller Family (1983)



»GELD HER ODER BATTERIE!«

(

Anhang: Tabellen

Tabelle 1: Kommando-Übersicht

Klasse	Kommando-Name	Bits und Flags 7	6	5	4	3	2	1	0	Bemerkung
I	Restore	0	0	0	0	h	v	r1	r0	= Seek Track 0
I	Seek	0	0	0	1	h	v	r1	r0	Suche Track
I	Step	0	0	1	u	h	v	r1	r0	Kopf bewegen
I	Step In	0	1	0	u	h	v	r1	r0	Kopf Richtung 80
I	Step Out	0	1	1	u	h	v	r1	r0	Kopf Richtung 00
II	Read Sector	1	0	0	m	s	e	c	0	Sektor lesen
II	Write Sector	1	0	1	m	s	e	c	a	Sektor schreiben
III	Read Address	1	1	0	0	0	e	0	0	ID lesen
III	Read Track	1	1	1	0	0	e	0	0	Track lesen
III	Write Track	1	1	1	1	0	e	0	0	Track schreiben
IV	Force Interrupt	1	1	0	1	I3	I2	I1	I0	Abbrechen

Tabelle 2: Flags

Flag	Bit	Name	wenn gesetzt (=1):
Klasse I:			
u	4	Update	Track-Reg. aktualisieren
h	3	Head Load	Kopf an Disk-Oberfläche laden
v	2	Verify	Track-Nr. vergleichen
r1,0	1,0	Step Rate	5.25", 3.5": 00=6, 01=12, 10=20, 11=30 ms 8": jeweils die Hälfte (3/6/10/15)
Klassen II&III:			
m	4	Multiple Record	mehrere Sektoren lesen/schreiben
s	3	Side Select	Seite 1 (nur mit c zusammen)
e	2	Delay	15 ms (8") bzw. 30 ms warten
c	1	Side Compare	Seite s überprüfen (sonst ist s egal)
a	0	Data Address Mark	Deleted Data Mark (F8h), sonst Data Mark (FBh)
Klasse IV:			
i3	3	Immediate	sofortiger Interrupt
i2	2	Index	Interrupt bei Index-Loch
i1	1	Not-Ready	Interrupt bei Übergang Ready -> Not Ready
i0	0	Ready	Interrupt bei Übergang Not Ready -> Ready
i0-i3=0000: kein Interrupt			

Tabelle 3a: Status-Byte bei Kommandos der Klasse I

Bit	Name	Bedeutung, wenn gesetzt (=1)
7	Not Ready	das Laufwerk ist nicht bereit
6	Protected	die Diskette ist schreibgeschützt
5	Head Loaded	der Kopf ist geladen
4	Seek Error	der Kopf steht nicht über dem richtigen Track
3	CRC Error	die CRC-Prüfsumme ist falsch
2	Track 00	der Kopf befindet sich über Track 0
1	Index	das Index-Loch wurde überquert
0	Busy	ein Kommando wird gerade abgearbeitet

Tabelle 3b: Status-Byte bei Kommandos der Klassen II und III

Bit	Name	Bedeutung, wenn gesetzt (=1)
7	Not Ready	das Laufwerk ist nicht bereit
6	Protected	Write: die Diskette ist schreibgeschützt sonst: 0
5	Record Type Write Fault	Read Sector: Adreßmarke ist "Deleted" Write : es ist ein Fehler aufgetreten sonst : 0
4	Record Not Found	Read/Write Sector: keinen Sektor mit dieser Track-, Sektor- und (evtl.) Seiten-Nr. gefunden Read Address : keinen Sektor gefunden sonst : 0
3	CRC Error	Read/Write Sector, Read Address: wenn Bit 4=1: Fehler im ID-Feld wenn Bit 4=0: Fehler im Daten-Feld sonst: 0
2	Lost Data	der Prozessor hat auf ein DRQ nicht rechtzeitig reagiert
1	Data Request	der Prozessor kann ein Datum schreiben/lesen
0	Busy	ein Kommando wird gerade abgearbeitet

Zugriff auf Disketten-Controller beim TRS-80 und Kompatiblen

Nachdem der Artikel über die Disketten-Controller 1771 bis 2797 fertiggestellt war, mußte ich mir von Helmut und Hartmut vorhalten lassen, daß viele TRS-80-Besitzer solange nichts damit anfangen können, wie der Zugriff nicht klarer wird. Also habe ich mir den Grosser geschnappt und werde das Kapitel 1 hier noch einmal nacherzählen. Es geht um die Adressen und Ports, über die der Disketten-Controller und die Laufwerke gesteuert werden.

Der Zugriff erfolgt bei den meisten Geräten über eine Memory-Map, d.h. an bestimmten Speicherstellen im Speicher befindet sich kein RAM, sondern bei einer Schreib- oder Lese-Operation wird die Disketten-Einheit angesprochen. Diese Adressen reichen (für den Disketten-Zugriff) von 37E0h bis 37FFh. Für den Zugriff werden wie beim normalen Speicher der Assembler-Befehl "LD" oder die Basic-Befehle "PEEK" und "POKE" benutzt. Allerdings sind die Hochsprachen (Basic, Turbo-Pascal) für diesen direkten Zugriff einfach zu langsam. Hier muß innerhalb von Mikrosekunden reagiert werden, was nur mit Assembler geht.

Unter Newdos ist die Memory-Map immer vorhanden und zugreifbar, nur das Genie IIs und IIIs schalten sie manchmal aus und dann befindet sich dort normales RAM. Schwierig wird's unter CP/M, wo dort immer RAM sein muß. Dazu unten mehr.

Ohne Memory-Map arbeiten dagegen das Model III und das Model IV(p). Sie verwenden Ports für den Zugriff. Der Vorteil ist, daß sich dort, wo sonst die Memory-Map liegt, jetzt freies RAM befinden kann und Probleme unter CP/M umgangen werden. Auch dazu unten mehr.

Die vom Controller benutzten Register sind (unter Newdos) einfach zu erreichen: Status-/Kommando-Register = (37ECh), Track-Register = (37EDh), Sektor-Register = (37EEh), Daten-Register = (37EFh).

Zu beachten ist allerdings, daß beim Lesen nicht unbedingt der Wert herauskommt, der vorher geschrieben wurde, sondern einer, den der Controller bestimmt. Z.B. werden in das Kommando-Register Kommandos geschrieben, aber beim Lesen bekommt man nicht das zuletzt gesetzte Kommando, sondern den momentanen Status des Controllers. Außerdem enthält das Track-Register nur die Nummer des Tracks, über dem der Kopf des angewählten Laufwerks steht. Wenn das Laufwerk gewechselt wird, muß das Programm dem Controller mitteilen, über welchem Track sich der Kopf des neuen Laufwerks befindet, oder ihn mit dem Kommando "Restore" über Track 00 positionieren (s. Beispiel unten).

Es gilt jetzt noch einige Steuerungsaufgaben zu lösen, die nicht vom Controller vorgenommen werden. Wichtig ist vor allem die Auswahl des Laufwerks, mit dem der Controller arbeiten soll. Dafür ist die Adresse 37E0h zuständig. Allerdings kann auf diese Adresse zum Laufwerk-Anwählen nur geschrieben werden. Wenn dabei Bit 0 gesetzt ist, wird Laufwerk 0 angewählt (merkt man daran, daß der Motor anläuft und die Laufwerkslampe leuchtet). Bit 1 gilt für Laufwerk 1, Bit 2 für Laufwerk 2. Bei Bit 3 scheiden sich die Geister: Im TRS-80 M.I und den Genies I und II wird dieses Bit zur Auswahl der Disketten-Seite benutzt (wenn es sich um doppelseitige Laufwerke handelt und die Hardware dies unterstützt). Im Genie III und IIIs übernimmt Bit 4 diese Aufgabe, während mit Bit 3 Laufwerk 3 angewählt wird. Im Genie IIs ist beides möglich.

Wenn sich im System ein Controller befindet, der mit beiden Schreibdichten arbeiten kann (meist 1791), wird diese über 37ECh ausgewählt. Damit der Controller diese Auswahl nicht als Kommando auffaßt, muß sie sich von allen möglichen Kommandos unterscheiden. Es werden deshalb alle Bits außer Bit 0 gesetzt; die Auswahl SD oder DD erfolgt über Bit 0.

Die letzte Unterscheidung betrifft die Auswahl zwischen 8-Zoll- und 5.25-Zoll-Laufwerken. Sie wird über 37EEh vorgenommen, also parallel zum Sektor-Register. Dazu muß ein Wert >=80h in diese Speicherstelle geschrieben werden (was bedeutet, daß keine Sektoren mit Nummern größer als 127 mehr benutzt werden können). Mit LD A,80h und LD (37EEh),A werden 5.25" angewählt, mit C0h dagegen 8". Achtung! Die IBM-HD-Laufwerke mit 1.2 MB in der Größe 5.25 Zoll zählen als 8-Zoll-Laufwerke! Ebenso die HD-Laufwerke in der Größe 3.5 Zoll mit 1.44 MB.

Noch ein kleines Schmankerl, das aber m.W. im TRS-80 selten benutzt wird: Beim Auslesen der Adresse 37E0h (Drive-Auswahl) haben die Bits 6 und 7 eine besondere Bedeutung. Wenn Bit 7 gesetzt ist, kam der letzte Interrupt (Ansprung über 0038h) von der eingebauten "Uhr", einem Taktgeber, der alle 25 ms einen Interrupt erzeugt (wird teilweise bei der Tastatur-Entprellung und der System-Uhr benutzt). Wenn Bit 6 gesetzt ist, kam der letzte Interrupt vom Floppy-Controller, d.h. ein Interrupt-gesteuerter Betrieb ist möglich, wird aber im Newdos nicht benutzt. Dort werden beim Disketten-Zugriff die Interrupts ausgeschaltet, womit die System-"Uhr" dann aus dem Takt kommt und ein paar (Milli-) Sekunden verliert.

Die Reihenfolge beim (ersten) Zugriff auf ein Laufwerk müßte so aussehen:
- Controller rücksetzen (altes Kommando abbrechen, Force Interrupt)
  LD A,0D0h
  LD (37ECh),A
- Auswahl der Laufwerksgröße über 37EEh (80h=5.25", C0h=8")
- Auswahl der Schreibdichte über 37ECh (FEh=SD, FFh=DD)
- Anwahl des Laufwerks und der Seite über 37E0h
  (Bit 0-2/3: Laufwerk; Bit 3/4: Seite)
- Warten, bis der Motor läuft (Warteschleife)
- Restore-Kommando an Controller (FDC) über 37ECh (00h schreiben)
- Warten, bis FDC nicht mehr Busy (Lesen von 37ECh, bis Bit 0=0)
- Track-Nr. ins Datenregister (37EFh) schreiben
- Seek-Kommando an Controller (37ECh, 10h schreiben)
- Warten, bis FDC nicht mehr Busy
- Sektor-Nr. ins Sektor-Register (37EEh) schreiben
- Kommando an Controller (z.B. Read Sector=80h, Write Sector=A0h)
  - auf DRQs warten (Lesen 37ECh, Bit 1 gesetzt = DRQ)
  - jeweils ein Byte aus dem Daten-Register (37EFh) holen oder hineinschreiben
  - Ende bei Motor Off (Bit 7 von 37ECh gesetzt) oder
  - Ende bei "nicht mehr Busy" (Bit 0 von 37ECh gelöscht)
  - evtl. zwischendurch nochmal Select (37E0h schreiben)
    (damit der Motor weiterläuft)
- Auswertung der Fehler (37ECh lesen, je nach Kommando andere Bedeutung)

Zu den Models III und IV(p)

Hier gibt es keine Memory-Map, sondern der Zugriff erfolgt über Ports (Befehle IN/OUT in Assembler oder Basic). Die Controller-Register liegen bei F0h bis F3h (Kommando-Register usw.). Die Auswahl des Laufwerks und der Zugriffs-Parameter erfolgt über Port F4h. Die Bits 0 bis 3 wählen eins von 4 Laufwerken aus, Bit 4 dient zur Seiten-Anwahl, Bit 7 zur Schreibdichte-Auswahl (SD/DD). Neu sind Bit 5, mit dem eine Write-Precompensation ein- und ausgeschaltet werden kann, und Bit 6, mit dem die CPU scheinbar zum Warten auf den Controller veranlaßt wird. Genaues weiß ich aber leider dazu nicht. Allerdings wird beim Model IV zumindest unter CP/M (Montezuma) ein Interrupt-gesteuerter Diskettenzugriff durchgeführt.

Memory-Maps unter CP/M

Im TRS-80 ist der Speicher so organisiert, daß RAM erst ab 4000h existiert und somit die Programme nie mit der Memory-Map bei 37E0h-37FFh (für Disk, 3800h-38FFh für Tastatur, 3C00h bis 3FFFh für Bildschirm) in Konflikt kommen. Unter CP/M stehen aber die Programme ab 100h durchgängig im Speicher, auch im Bereich 3xxxh. Also wurde dort RAM eingesetzt und die Memory-Map entweder in einen anderen Speicher-Bereich verschoben (nach "oben", bei Fxxxh), oder nur eingeschaltet, wenn ein Programm auf diese Memory-Map zugreifen wollte.

Im ersten Fall gibt es keine Probleme: Die Memory-Map ist ja immer da, nur nicht am gewohnten Platz. Im zweiten Fall kann es passieren, daß ein Programm bei 3xxxh steht (oder dort wichtige Daten wie den Stack o.ä. hat) und nun die Memory-Map einschaltet. Dann "verschwindet" aber das dort vorher befindliche RAM, bis die Memory-Map wieder ausgeschaltet wird. Also darf das Programm sich nicht in diesem Bereich befinden, dort keinen Stack haben und nicht auf Daten zugreifen, die dort abgelegt wurden oder abgelegt werden sollen. Wenn zum Beispiel ein Sektor nach 3700h eingelesen werden soll, muß das Programm (meist das BIOS) ihn erst irgendwo anders ablegen, nach dem Lesen dann die Memory-Map ausschalten und ihn an seinen Bestimmungsort (hier 3700h) verschieben.

Solange das BIOS alle diese Zugriffe erledigt, gibt es keine Probleme, denn das BIOS steht oben im Speicher, meist über E000h, und hat alle Stacks und Daten dort gelagert. Sobald aber direkt von einem normalen Programm auf die Memory-Map zugegriffen werden soll, muß folgendes sichergestellt werden:

1. Das Programm muß komplett vor oder hinter der Memory-Map stehen.
2. Der Stack darf nicht im Memory-Map-Bereich liegen.
3. Es dürfen keine für den Zugriff wichtigen Daten (z.B. die Tracknummer) im RAM-Bereich "neben" der Memory-Map stehen.
4. Wenn die Memory-Map eingeschaltet ist, sollte das BIOS nicht dazwischenfunken, also: Keine Interrupts, keine BIOS-Aufrufe!
5. Nach dem Zugriff muß die Memory-Map wieder ausgeschaltet und evtl. Interrupts eingeschaltet werden.

Der Punkt "Keine BIOS-Aufrufe" ist nicht zu unterschätzen, denn meist schaltet das BIOS die Memory-Map wieder aus, wenn es seine Arbeit beendet hat. Während das Programm "denkt", sie wäre noch eingeschaltet. Dann kommt nur noch Daten-Müll zustande.

Dieser Bereich läßt sich auch ziemlich gut anhand des Beispielprogramms nachvollziehen, das irgendwo in diesem Info steht.

Gerald Schröder

Lit.: Hartmut Grosser: Das DOS-Buch, Röckrath, Aachen 1985
besonders Kapitel 1
Diverse Listings, u.a.:
Röckrath : ROM-Listing des TRS-80
TCS : BIOS für Speedmaster
Montezuma: BIOS für Model IVp
Schmidtke: BIOS für Genie I/II u.a.
myself : BIOS für Genie IIs

Das FDC-Demo-Programm

Wieder muß ich zugeben, daß Helmut und Hartmut als treibende Kräfte hinter diesem Artikel standen; jaja, ich werde von den beiden zu den übelsten Machenschaften getrieben. Eigentlich sollte das Ding nur kurz demonstrieren, wie der FDC (Floppy-Disk-Controller) programmiert werden kann. Im TRS-80-Disk-Guide gibt es dazu ein herrlich kurzes Programm und Manfred Held schickte mir ebenfalls ein sehr schönes kurzes Exemplar. Aber es überkam mich wieder mal. Also: Für die Experten sind nur die Seiten 1, 3 und die erste Hälfte von Seite 4 interessant. Der Rest sind Umschaltungen und Fehler-Ausgaben, die der Profi von Hand erledigt bzw. an irgendwelchen gesetzten oder ungesetzten Bits erkennt. Dem Assembler-Neuling empfehle ich, das komplette Ding abzutippen.

Allerdings sind dabei einige Fallen zu beachten, denn ich habe das Ding auf einem Genie IIs unter CP/M erstellt. Unter Newdos dürfte es trotzdem genauso laufen. Ich werde die Unterschiede unten zusammenfassen. Erstmal gehe ich auf die CP/M-Version ein.

Der Zugriff auf den Controller erfolgt hier über die alte TRS-80-Memory-Map, die beim Genie IIs eingeblendet werden kann. Wie das bei anderen TRS-80-Kompatiblen funktioniert, weiß ich nicht. Auf jeden Fall müssen die Routinen "MemMapOn" und "MemMapOff" geändert werden (im Zweifelsfall einfach ein "RET"). Damit verbunden ist auch das Ein- und Ausschalten der Interrupts. "DisableInt" und "EnableInt" können auch nur nur aus "DI" und "EI" (& "RET") bestehen. Im Zweifelsfall lieber ein "DI" zuiel, dafür aber kein "EI".

Das Programm läuft total unter Debugger-Kontrolle. Ihr könnt es auch so starten, aber dabei kommt nicht viel heraus (außer, daß es hoffentlich nicht abstürzt). Also folgender Aufruf: "DDTZ FDCTEST.COM" (so heißt das Programm bei mir). Dann mit "B103" einen Breakpoint bei 103h setzen und mit "G100" das Programm starten. Evtl. auftretende Fehler werden angezeigt, ansonsten sollte die Meldung "Es trat kein Fehler auf." erscheinen und die Ausgabe aller Register-Inhalte. Mit "D6000" (und mehrmals <ENTER>) kann nun der geladene Sektor betrachtet werden. Evtl. vor dem Programmstart mit "Z6000,7000,00" diesen Bereich löschen, damit sicher ist, daß auch wirklich etwas geladen wurde.

Wenn andere Parameter (Laufwerk, Track, Sektor, Buffer, Kommandos) gewünscht werden: Einfach die EQUates ändern, neu assemblieren und testen. Oder direkt per DDTZ und Kommando "A"ssemble eines der diversen "LD A,.." patchen. Alle Parameter werden nur am Anfang (erste Seite) benutzt und jeder taucht nur einmal auf. Ein Wort zum Asseblieren: Nachdem der Text mit Wordstar oder so bearbeitet wurde, Macro-80 mit "M80 =fdctest" aufrufen. Wenn er "NO FATAL ERRORS" meldet, kann gelinkt werden mit "L80 fdctest,fdctest/n/e". Wenn alles geklappt hat, steht der COM-File auf der Disk (schneller geht's mit 'ner Ramdisk).

Das Programm eignet sich nicht nur zum Lesen eines Sektors. Es ist vor allem zum Experimentieren gedacht. Also immer kräftig Breakpoints setzen und die Register belauern. Und alles mögliche variieren, z.B. andere Kommandos geben, Warte-Schleifen ändern, Status-Ausgaben genauer überprüfen usw. Außerdem kann das Ding erweitert werden, z.B. zu einem Formatier- oder Kopier-Programm, direkt auf Hardware-Ebene, so daß auch "kopiergeschützte" Disks problemlos angegangen werden können.

Zum Programm im einzelnen

Zuerst wird auf einen eigenen Stack umgeschaltet, denn es ist nicht sicher, ob der vorherige nicht in dem Bereich liegt, wo später (evtl.) die Memory-Map eingeblendet wird. Dann wird dem Controller befohlen, ein evtl. laufendes Kommando zu unterbrechen, und danach ein Laufwerk angewählt (DiskStart). Hier kommen die ersten Feinheiten: Wenn nicht lange genug gewartet wird, kann der Motor nicht starten und nichts klappt mehr. Nehmt

dazu einfach mal die Warteschleife raus. Übrigens läßt sich schlecht testen, ob auch wirklich eine Diskette im Laufwerk liegt. Eine Möglichkeit: Man frage das Index-Bit (Bit 1 im Status-Register, 37ECh) ab. Es müßte fünfmal in der Sekunde von 0 auf 1 und wieder zurück springen, wenn sich eine Diskette im Laufwerk dreht (300 U/min, falls ich ich mich nicht irre). Sonst bleibt es immer 1. Ich habe auf die Abfrage verzichtet. (Nachzulesen im Grosser, Seite 3-32, 47ECh-4808h: TSTDSK).

Nun führt der Controller ein Restore aus, d.h. er sucht Track 0. Das ist nötig, weil er nicht wissen kann, wo der Kopf des Laufwerks vorher gestanden hat und die im Track-Register stehende Track-Nummer höchstwahrscheinlich für ein anderes Laufwerk (bzw. dessen Kopf) gilt. Nach diesem Kommando stimmt das Track-Register mit der aktuellen Kopf-Position überein (beide sind 00). In dieser Version (Kommando 04h) liest der FDC auch noch von der Diskette einen Sektor-Header und sieht nach, ob dort wirklich die (Track-)-Nummer 0 eingetragen ist. Wenn das Kommando 00h benutzt wird, prüft er das nicht nach, sondern verläßt sich auf das vom Laufwerk gelieferte Signal "Track00", das aber auch kommt, wenn keine Diskette im Laufwerk liegt! Die kleine Warte-Schleife nach dem Ausgeben des Kommandos an den Controller ist nötig, weil der erstmal die Status-Bits aktualisieren muß. Wenn die Schleife fehlt, wird vom ersten "LD A,(fdccmd)" der vor dem Kommando gültige Status gelesen, was wahrscheinich Quatsch ergibt. Nach der Schleife wartet das Programm, bis das Busy-Bit gelöscht wird (Kommando abgearbeitet) oder das Laufwerk sich abmeldet. Die Abfrage des Busy-Bit allein reicht nicht, denn der Controller macht 5 Disk-Umdrehungen lang Leseversuche, aber wenn keine Diskette im Laufwerk liegt, kommen keine Index-Impulse! Dann wartet er ewig, daß die erste Umdrehung beendet ist... (S. auch c't-Artikel zum WD-2797). Eigentlich müßte hier (und bei den anderen Routinen auch) ein "Force Interrupt" am Ende dafür sorgen, daß das letzte Kommando wirklich abgebrochen wurde. Habe ich wieder drauf verzichtet.

Nun fährt das Laufwerk den richtigen Track an. Dazu erhält der FDC den gewünschten Track im Daten-Register (im Track-Register steht der aktuelle Track, noch 00, der nicht überschrieben werden darf!). Dann kommt das Seek-Kommando ("Suche Track"). Wieder überprüft der FDC am Ende, ob die Track-Nummer auch in einem Sektor-Header vermerkt ist. Wenn das nicht passieren soll, muß 10h statt 14h genommen werden (weitere Möglichkeiten: s. FDC-Artikel).

Wenn bei einem dieser Kommandos ein Fehler auftrat, wird dieser angezeigt und das Programm abgebrochen. Der Freak setzt hier Breakpoints und versucht, allein aus dem in A stehenden Fehler-Status herauszubekommen, was falsch gelaufen ist. Sollte jeder zum Test mal versuchen (z.B. nach den CALLs oder bei dem "AND 98h" oder evtl. schon nach dem ersten "LD A,(fdc-cmd)"). Allerdings kann dann nicht fortgesetzt werden, denn das Laufwerk schaltet sich meistens automatisch ab. Außerdem ist Vorsicht bei Breakpoints geboten, die beim "x-ten" Durchlauf erst wirksam werden (deaktivierte Breakpoints im DDTZ; ich habe keine Ahnung, wie man sie ganz entfernen kann). DDTZ fragt da mal "kurz" die Tastatur ab; das dauert zu lange und evtl. wird die Memory-Map wieder abgeschaltet! "Trace" würde ich lieber gleich vergessen (zu langsam). Schließlich geht es hier um "Echtzeit"-Operationen.

Kommen wir zum wichtigsten Teil: Dem Lesen des Sektors. Zuerst wird die Sektor-Nummer gesetzt und dann das Read-Kommando ausgegeben. Hier ist die Warteschleife danach besonders wichtig, denn sonst kommt der Status vom "Seek"-Kommando durch. Es dauert relativ lange, bis das erste DRQ erzeugt wird, das anzeigt, das ein Byte abgeholt werden muß. Aber nun geht es Schlag auf Schlag. Jedes DRQ muß sofort mit dem Lesen des Daten-Registers (37EFh) beantwortet werden, sonst kommt ein "Lost Data Error". Und hier heißt "sofort" wirklich SOFORT. Deshalb: Keine Interrupts, möglichst nur mit Registern und schnellen Kommandos arbeiten. Bei meinem 9.2-Mhz-Genie-IIs klappt die vorgestellte Routine, aber einige eingefügte NOPs bringen das Ganze zum Zusammenbruch. Ich hoffe, daß mit den angemerkten Zeilen auch die alten 1.77-Mhz-Maschinen mithalten können.

Hier beginnt nun das große Spielfeld: Statt einem einfachen "Read Sector" können mit einem gesetzten Multiple-Bit (Bit 4, +10h) alle auf einem Track befindlichen Sektoren (ab der gewählten Sektor-Nummer) eingelesen werden, ohne Änderung der Routine, ebenso ein ganzer Track (Kommando E0h) oder einfach ein Sektor-Header (Read Address, Kommando C0h). Zum Schreiben muß nicht viel verändert werden: Die beiden Zeilen bei "ReadData" werden einfach vertauscht, fertig! Allerdings sollte bei "ReadEnde" dann ein "AND FCh" folgen, denn hier gelten andere Fehler-Meldungen (die Fehler-Routine stimmt dann nicht mehr ganz). Dann kann natürlich auch ein ganzer Track geschrieben werden (Write Track, Kommando F0h) und schon ist das Format-Programm (fast) fertig!

Zurück zu dem vorliegenden Programm. Wenn kein Fehler aufgetreten auf, kommt die entsprechende Meldung und ab geht's in den Debugger, damit Ihr Euch den gelesenen Kram ansehen könnt. Falls irgendwo ein Fehler aufgetreten ist, wird dieser angezeigt und ebenfalls der Debugger angesprungen. Übrigens können die Fehler-Routinen auch angesprungen werden, wenn Ihr einfach mal wissen wollt, wie an einer Stelle der Status aussieht, z.B. bei "StepLoop" nach dem x-ten "LD A,(fdccmd)" ein Sprung zu "StepError" und schon steht's auf dem Bildschirm, Phosphor für Phosphor, grün auf schwarz bzw. sehr-hellgrün auf hellgrün (Gruß an Arnulf!).

Womit wir beim Thema wären: Wie steht's mit Newdos?

Das ORG sollte 5200h sein; dann dürfte es keine Probleme geben. Die Labels sollten ohne ":" eingegeben werden (jedenfalls unter ZEUS). Vielleicht sind einige Namen auch zu lang. Müßt Ihr mal probieren. Am Anfang sollte statt JP 0000 ein JP 440Dh stehen, dann könnt Ihr das Ding einfach als CMD-File starten und DEBUG meldet sich nach dem Lesen des Sektors automatisch. Oder Ihr setzt die letzte Zeile auf "END 440dh", womit DEBUG sofort nach dem Laden des Programms angesprungen wird.

Ansonsten ist der meiste Kram meiner Meinung nach unkritisch (habe ich aber nicht unter Newdos getestet!). Die Memory-Map muß jedenfalls nicht eingeschaltet werden (ersatzlos streichen). Die Routine "Print" besteht unter Newdos nur aus einer Zeile: "JP 4467h" (gibt den String (HL) bis zu einem 03 oder 0dh aus). Da Newdos aber nur 0ah haben möchte, um in die nächste Zeile zu kommen, müssen ab "KeinFehler" alle 0dh rausgeworfen werden (sonst kommen statt Fehlermeldungen nur Leerzeilen).

Andere Rechner

Für alle Rechner mit einem Controller der Familie 1771 bis 2797 müßte das Programm gültig sein, wenn auch evtl. andere Adressen benutzt werden. Wer (sinnvollerweise) seinen Controller über Ports anspricht, kann von mir eine Port-Version bekommen. Finde ich persönlich viel eleganter, aber der TRS-80 macht's nicht (von sich aus, läßt sich ändern, s. Model III und IV sowie meinen Port-Umbau).

Allerdings fehlt hier etwas ganz: Die Auswahl von 8-/5.25-Zoll-Laufwerken und einfacher bzw. doppelter Schreibdichte. Das dürfte überall anders gelöst sein und ist zum "Spielen" nicht so wichtig. Die Seiten-Auswahl kann einfach beim Drive-Select mit Bit 3 bzw. 4 (Genie III(s), einige IIs) erledigt werden, ist aber bei anderen Rechnern vielleicht anders gelöst, beim 2797 macht es z.B. der Controller selbst.

Gerald Schröder

```
;         ReadSector: Liest einen Sektor von der Diskette
;         28.2.89 von Gerald Schröder, verbessert: 3.3.89
;         für FDC-Controller 1771/179x/279x
;         auf Genie IIs (TRS-80-kompatibel) unter CP/M
;         erweiterungsfähig in alle Richtungen
;         Vielen Dank an Manfred Held und Hartmut Obermann.

          .Z80                          ;für Macro-80 unter CP/M
          CSEG                          ;für Macro-80 (absolutes Segment)
;(Newdos: ORG 5200h)

;         Memory-Mapped I/O-Adressen (TRS-80 u. kompatible):
fdcsel    EQU       37e0h               ;Laufwerk-Auswahl
fdccmd    EQU       37ech               ;Kommando- und Status-Register
fdctrk    EQU       37edh               ;Track-Register
fdcsec    EQU       37eeh               ;Sektor-Register
fdcdat    EQU       37efh               ;Daten-Register

;         Parameter
Drive     EQU       02                  ;Laufwerk B:
Track     EQU       03                  ;Track 3
Sector    EQU       04                  ;Sektor 4
RestCMD   EQU       04                  ;Restore-Kommando mit Verify
SeekCMD   EQU       14h                 ;Seek-Kommando mit Verify
ReadCMD   EQU       80h                 ;Read-Sector-Kommando ohne alles
forcel    EQU       0D0h                ;Force-Interrupt-Kommando
Buffer    EQU       6000h               ;hier steht nachher der Sektor

Debugger:                       ;hier sollte ein Breakpoint hin!
          JP        0                   ;falls vergessen: Warm Boot
                                        ;(Newdos: JP 4400h ; Debug)
; ******************  Anfang Hauptprogramm  *********************
Start:
          LD        SP,Steck  ;eigenen Stack benutzen
          CALL      MemMapOn  ;evtl. Memory-Map einschalten

          LD        A,Forcel  ;vorher laufendes Kommando
          LD        (fdccmd),A          ;abbrechen

          LD        A,Drive             ;Laufwerk und Seite wählen
          CALL      DiskStart ;Motor anlaufen lassen und Select-Maske
                                        ;merken
          LD        A,RestCMD ;Restore-Kommando
          CALL      Step                ;ausführen
          JP        NZ,StepError

          LD        A,Track             ;Track-Nummer
          LD        (fdcdat),A          ;in des Daten-Register (nicht Track-R.!)
          LD        A,SeekCMD ;Seek-Kommando
          CALL      Step                ;ausführen
          JP        NZ,StepError

          LD        A,Sector
          LD        (fdcsec),A          ;Sektor auswählen
          LD        A,ReadCMD ;Read Sector
          CALL      ReadSector          ;Sektor lesen
          JP        NZ,ReadSError       ;falls Lesefehler

          CALL      MemMapOff ;Memory Map ausschalten
          LD        HL,KeinFehler       ;Meldung kein Fehler
          CALL      Print               ;anzeigen
          CALL      CrLf                ;CR,LF ausgeben
          JP        Debugger  ;ab in den Debugger oder so
```

```
; ************** Ende Hauptprogramm - Anfang Unterprogramme ******

;       MemMapOn: Memory-Map bei 3700h einblenden
;       nur nötig, wenn unter CP/M benutzt und keine Port-I/O
;       hier: für Genie IIs mit Port-Umbau
;       ----------------------------------------------------
sysport EQU     0feh
MemMapOn:
        CALL    DisableInt          ;Interrupts lieber aus
        IN      A,(sysport)         ;alten Zustand des Ports
        LD      (PortAlt),A         ;merken
        AND     0eeh                ;Port I/O aus, Memory Map ein
        OUT     (sysport),A
        RET

;       MemMapOff: Memory-Map ausblenden
;       nur nötig, wenn unter CP/M benutzt und keine Port-I/O
;       hier: für Genie IIs mit Port-Umbau
;       ----------------------------------------------------
MemMapOff:
        LD      A,00                ;alter Zustand
PortAlt EQU     $-1
        OUT     (sysport),A
        CALL    EnableInt ;hier dürfen sie wieder an
        RET

;       DisableInt: schaltet Interrupts aus
;       wenn mehrmals aufgerufen: merken, damit erst beim letzten
;       EnableInt diese wieder angeschaltet werden
DisableInt:
        LD      A,(AnzahlAufrufe)
        INC     A
        LD      (AnzahlAufrufe),A
        DI
        RET

;       EnableInt: schaltet Interrupts aus, wenn Aufruf 0 erreicht
;       Achtung! Bei TRS-80 (-Kompatiblen) unter CP/M mit DDTZ darf
;       hier nie ein EI erfolgen, denn der TRS-80 liefert alle 25 ms
;       einen INT0, der auf 0038h geht, was gleichzeitig Breakpoint
;       für DDTZ ist (RST 38h). Somit wird DDTZ aufgerufen, als ob
;       ein Breakpoint erreicht wäre.
EnableInt:
        LD      A,(AnzahlAufrufe)
        DEC     A
        LD      (AnzahlAufrufe),A
        RET     NZ                  ;noch nicht 0: zurück
;       EI                          ;Achtung! Nicht bei TRS-80-CP/M-DDTZ
        RET

AnzahlAufrufe:  DB      0

; ******************* FDC-Unterprogramme **********************

;       DiskStart: Motor des Laufwerks anlaufen lassen und
;                  Seite auswählen, Select-Maske retten
;       IN: A=Laufwerk/Seite
;           Bit 0: Lw. 0, Bit 1: Lw. 1, Bit 2: Lw. 2
;           Bit 3: Genie IIs/IIIs: Lw. 3; TRS-80 u.a.: Seite 1
;           Bit 4: Genie IIs/IIIs: Seite 1
;       OUT: / (hier könnte überprüft werden, ob das Laufwerk überhaupt
;              existiert und ob eine Diskette eingelegt ist
;       benutzte Register: A
;       ----------------------------------------------------
```

```
DiskStart:
          LD        (SelMask),A         ;Select-Maske merken
          CALL      DriveSel  ;Motor anwerfen
;evtl. prüfen, ob Lw. vorhanden und Diskette eingelegt
          RET

;         DriveSel: Läßt Motor des Laufwerks weiterlaufen
;                   vorher muß DiskStart zum Setzen der Select-Maske
;                     aufgerufen werden
;         IN: /
;         OUT: /
;         benutzte Register: A
;         ----------------------------------------------------
DriveSel:
          LD        A,0                 ;Select-Maske
SelMask   EQU       $-1
          LD        (fdcsel),A          ;Motor laufen lasse
          RET

;         Step: Kommando der Klasse I ausführen
;         IN: A=Kommando
;         OUT: Flag NZ = Fehler, Fehler-Maske in A
;         benutzte Register: A
;         ----------------------------------------
Step:
          LD        (fdccmd),A          ;Kommando ausgeben
          CALL      Wait                ;Warten
          CALL      DriveSel  ;Motor weiterlaufen lassen
Steploop:
          LD        A,(fdccmd)          ;Status
          BIT       0,A                 ;noch Busy?
          JR        Z,StepEnde          ;nein, fertig
          BIT       7,A                 ;Motor Off?
          JR        Z,Steploop          ;nö, warten
          LD        A,ForceI  ;ja, Kommando abbrechen
          LD        (fdccmd),A
          LD        A,(fdccmd)          ;und Fehler lesen
StepEnde:
          AND       98h                 ;Motor Off, Seek Error oder CRC-Error?
          RET                           ;wenn ja: NZ


;         ReadSector: liest einen Sektor von der Diskette in Buffer
;         IN: A=Kommando
;         OUT: Flag NZ=Fehler; Fehler-Maske in A
;
;         Vorsicht! Diese Routine ist unheimlich empfindlich gegenüber
;         Verzögerungen. Ein Befehl mehr oder weniger kann über den
;         Erfolg oder Mißerfolg entscheiden!
;         ------------------------------------------------------------
ReadSector:
          PUSH      AF
          CALL      DisableInt          ;Interrupt aus
          POP       AF
          LD        HL,fdccmd ;Adresse Kommando-/Status-Reg.
          LD        DE,fdcdat ;Adresse Daten-Register
          LD        BC,Buffer ;Anfang des Buffers
          LD        (HL),A              ;Kommando an den Controller
          CALL      Wait                ;warten, bis der das kapiert hat
          CALL      DriveSel  ;Motor laufen lassen
```

```
ReadLoop:
          BIT       1,(HL)              ;DRQ? (nur bei langsamen Rechnern)
          JP        NZ,ReadData         ;ja
          BIT       0,(HL)              ;noch Busy?
          JR        Z,ReadEnde          ;nein, Ende
          BIT       1,(HL)              ;DRQ? (langsame Rechner)
          JR        NZ,ReadData         ;ja
          BIT       7,(HL)              ;Motor off?
          JR        NZ,ReadBreak        ;ja, Fehler
          BIT       1,(HL)              ;DRQ?
          JR        Z,ReadLoop          ;nein, warten
ReadData:
          LD        A,(DE)              ;ja, Datum lesen
          LD        (BC),A              ;und ablegen
          INC       BC                  ;Buffer-Adresse weiter
          JP        ReadLoop  ;nächstes Datum
ReadBreak:
          LD        A,ForceI  ;Kommando abbrechen
          LD        (HL),A
          LD        A,(HL)              ;Fehlerstatus lesen
ReadEnde:
          LD        A,(HL)
          AND       9ch                 ;Fehler aufgetreten? (Not Ready, Record
                                        ;Not Found, CRC-Error, Lost Data)
                                        ;wenn ja: NZ
          PUSH      AF
          CALL      EnableInt ;evtl. Interrupts einschalten
          POP       AF
          RET


;         Wait: wartet einige Mikrosekunden oder so
;         IN: /
;         OUT: /
;         benutzte Register: A
;         ----------------------------------------
Time      EQU       20
Wait:     LD        A,Time
Wait1:    DEC       A
          JR        NZ,Wait1
          RET

  ********** Ende FDC-Routinen - Anfang Fehler-Ausgabe **********

;         Fehler-Routinen
;         Alle Routinen erwarten eine Fehler-Maske in A.
;         Zuerst wird immer auf den Normalbetrieb zurückgeschaltet.
;         Dann wird ein Zeiger auf eine Tabelle mit Zeigern auf Fehler-
;         meldungen nach HL geladen. Je nach FDC-Kommando sehen die
;         Tabelle und die Fehlermeldungen anders aus.
;         Die Routine 'LookForError' wertet die Fehler-Maske aus und
;         läßt über die Routine 'ShowFault' Fehler-Meldungen ausgeben.
;         Je nachdem, nach welchem FDC-Kommando der Fehler aufgetreten
;         ist, sind die verschiedenen Routinen anzuspringen.
;         Momentan wird danach der Debugger angesprungen. Alternativ ist
;         auch ein RETurn möglich

;         StepError: Bei Fehlern nach Kommandos der Klasse I oder II
;                    (Step oder Force Interrupt)
;         mögliche Fehler sind: Not Ready, Seek-Error, CRC-Error
;         außerdem können die Meldungen Protected, Head Loaded, Track 0,
;         und Index auftreten (dies sind keine Fehler!)
;         IN: A=Fehlermaske
;         OUT: /
;         ------------------------------------------------------------
```

```
StepError:
        PUSH    AF              ;Fehlermaske retten
        CALL    MemMapOff       ;nur unter CP/M
        POP     AF              ;Fehlermaske zurück
        LD      HL,StepTable    ;Tabelle mit Meldungen
        JR      LookForError    ;Fehler anzeigen


;       ReadSError: Bei Fehlern nach dem Read-Sector-Kommando
;       möglicne Fehler sind: Not Ready, Record Not Found, CRC-Error,
;        Lost Data
;       außerdem kann die Meldung Deleted Data Mark auftreten (dies ist
;       kein Fehler!)
;       IN: A=Fehlermaske
;       OUT: /
;       ------------------------------------------------------------
ReadSError:
        PUSH    AF              ;Fehlermaske retten
        CALL    MemMapOff       ;nur unter CP/M
        POP     AF              ;Fehlermaske zurück
        LD      HL,ReadSTable   ;Tabelle mit Meldungen
        JR      LookForError    ;Fehler anzeigen


;       LookForError: Auswertung einer Fehlermaske und Ausgabe von Fehler-
;                     meldungen
;       IN: A=Fehlermaske
;           HL=Zeiger auf Tabelle mit Zeigern auf Fehlermeldungen
;       OUT: /
;       ------------------------------------------------------------------
LookForError:
        LD      B,8             ;8 Bits
ErrorLoop:
        RRCA                    ;Bit unten herausrotieren
        CALL    C,ShowFault     ;Bit gesetzt: Fehler anzeigen
        INC     HL              ;nächste Meldung
        INC     HL
        DJNZ    ErrorLoop       ;für alle 8 Bits

        CALL    CrLf            ;neue Zeile
        JP      Debugger        ;Abgang


;       ShowFault: Zeigt eine Fehlermeldung
;       IN: A=Fehlermaske, B=Fehler-Zähler
;           HL=Zeiger auf Zeiger auf Fehlermeldung
;       OUT: /
;       ----------------------------------------
ShowFault:
        PUSH    AF              ;retten: Status-Bits
        PUSH    BC              ;        Zähler
        PUSH    HL              ;        Tabellen-Zeiger
        LD      A,(HL)          ;LSB vom Zeiger auf Fehlermeldung
        INC     HL
        LD      H,(HL)          ;MSB
        LD      L,A             ;nach HL
        CALL    Print           ;Fehlermeldung (HL) anzeigen
        POP     HL              ;zurück: Tabellen-Zeiger
        POP     BC              ;        Zähler
        POP     AF                      Fehler-Maske
        RET
```

```
;       Print: Zeigt Fehler-Meldung an
;       hier: für CP/M über BDOS-Funktion Nr. 9 (PrintString)
;       IN: HL=Zeiger auf Meldung
;       OUT: /
;       ------------------------------------------------
Print:  EX      DE,HL           ;Zeiger nach DE
        LD      C,9             ;BDOS-Fkt. PrintString
        JP      0005            ;BDOS aufrufen

;       CrLf: Springt in die nächste Zeile
;       unter CP/M wird der String CR/LF ausgegeben
;       unter Newdos kann ein CR ausgegeben werden
;       ------------------------------------------
CrLf.   LD      HL,CRLFString
        JR      Print

CrLfString:
        DB      0dh,0ah,'$'


;       Tabellen mit Zeigern auf Fehlermeldungen für verschiedene
;       Kommandos
;       Zu beachten: immer 8 Einträge
;       ------------------------------------------------------

StepTable:
        DW      Busy            ;Bit 0 (dürfte nie auftreten)
        DW      Index           ;Bit 1
        DW      Track0          ;Bit 2
        DW      CRCError        ;Bit 3
        DW      SeekError       ;Bit 4
        DW      HeadLoaded      ;Bit 5
        DW      Protected       ;Bit 6
        DW      NotReady        ;Bit 7

ReadSTable:
        DW      Busy            ;dürfte nie auftreten
        DW      DRQ             ;ebenso
        DW      LostData
        DW      CRCError
        DW      RecNotFnd
        DW      Deleted         ;kein Fehler
        DW      Nothing         ;kann nicht auftreten
        DW      NotReady

;       hier können weitere Tabellen angefügt werden
;       für: Write Sector, Read Track, Write Track, Read Address


;       Fehler-Meldungen (bzw. überhaupt alle Meldungen)
;       ------------------------------------------------
Ende    EQU     '$'     ;Markierung für CP/M, bei Newdos: 03
                        ;außerdem bei Newdos: alle 0dh entfernen!
KeinFehler:
        DB      0dh,0ah,'Es trat kein Fehler auf.',Ende
Busy:   DB      0dh,0ah,'Der FDC ist beschäftigt (kein Fehler).',Ende
Index:  DB      0dh,0ah,'Das Index-Loch wurde überquert (kein Fehler).',Ende
Track0: DB      0dh,0ah,'Der Kopf steht über Track 0 (kein Fehler).',Ende
CRCError:
        DB      0dh,0ah,'Eine Prüfsumme ist falsch!',Ende
SeekError:
        DB      0dh,0ah,'Der Track hat eine falsche Nummer.',Ende
```

»Verdammt noch mal, sag nicht immer PAPA zu mir!«

```
HeadLoaded:
        DB      Odh,Oah,'Der Kopf ist geladen (kein Fehler).',Ende
Protected:
        DB      Odh,Oah,'Die Diskette ist schreibgeschützt (hier '
        DB      'kein Fehler).',Ende
NotReady:
        DB      Odh,Oah,'Das Laufwerk ist nicht bereit!',Ende
DRQ:    DB      Odh,Oah,'Der FDC hat ein Data Request erzeugt (kein Fehler)
        DB      Ende
LostData:
        DB      Odh,Oah,'Beim Übertragen sind Daten verlorengegangen!',Ende
RecNotFnd:
        DB      Odh,Oah,'Der Sektor wurde nicht gefunden!',Ende
Deleted:
        DB      Odh,Oah,'Die Adreßmarke ist * Deleted * (kein Fehler).',Ende
Nothing:
        DB      Odh,Oah,'Dieser Fehler kann/darf nicht auftreten.'
        DB      Odh,Oah,'Mad Programmer Error.',Ende

        DS      30
Stack:
        END     Start           ;Newdos: END 440Dh ;Enter Debug
```

Ein Controller muß den Speicher steuern

Wer ein Floppy-Disk-Laufwerk — das Prinzip ist übrigens bei Festplatten-Laufwerken dasselbe — an seinen Computer anschließen will, benötigt dazu einen Controller, der auf den jeweiligen Rechnertyp zugeschnitten sein muß. Der Controller kann Bestandteil von Rechner oder Laufwerk sein, aber auch die Form einer Steckkarte haben, die ins Computergehäuse oder eine Erweiterungsbox paßt.

Die Information befindet sich bitseriell als magnetisches Abbild des verwendeten Codes in konzentrischen Spuren auf den Datenträgeroberflächen. Das Umsetzen elektrische Größe in magnetische Größe und umgekehrt erfolgt durch einen Schreib-/Lesekopf. Zum Auffinden des Speicherorts wird dieser vor einem Schreib- oder Lesevorgang zur gewünschten Spur positioniert. Durch die Rotation des Datenträgers bewegt sich während einer Umdrehung eine gesamte Spur am Kopf vorbei.

Da jedoch sowohl beim Lesen als auch beim Schreiben üblicherweise nur ein Teil einer Spur, dieser aber gezielt, transferiert werden soll, ist ein Verfahren notwendig, um Informationen innerhalb einer Spur genau adressieren zu können.

Adressen und Daten sind durch Polynome gesichert

Aus diesem Grund wird jede »jungfräuliche« Diskette oder Platte vor der erstmaligen Verwendung zur Datenspeicherung »formatiert«. Hierbei teilt man durch das vollständige Beschreiben aller Spuren den Speicherplatz auf dem Datenträger bei einem Synchronisationspunkt beginnend ein und macht ihn adressierbar.

Eine Spur wird in »Sektoren« unterteilt. Diese bilden die kleinste adressierbare Einheit und bestehen üblicherweise aus einem Datenfeld, für die zu speichernden Nutzdaten auch noch aus einem Adreßfeld. In diesem sind die Spuradresse, die Kopfadresse (bei mehreren Datenoberflächen pro Laufwerk), die Sektornummern sowie möglicherweise weitere Kennungen (zum Beispiel über die Länge des Datenfeldes) eingetragen. Die Adreßfelder der Sektoren werden nach dem Formatieren nicht mehr verändert. Sie dienen bei allen späteren Transfers zum Auffinden des Speicherplatzes.

Nach der Positionierung auf die gewünschte Spur liest und interpretiert der Controller alle den Schreib-/Lesekopf passierenden Adreßfelder. Sind Spurnummer und Kopfnummer identisch mit der gewünschten Vorgabe, so ist richtig positioniert. Stimmen dann auch noch die Sektornummern des Adreßfeldes mit dem gewünschten Wert überein, so ist richtig adressiert und der entsprechende Sektor befindet sich unter dem Schreib-/Lesekopf. Es wird kurz darauf das zugehörige Datenfeld entweder überschrieben oder gelesen.

Adreßfeld und Datenfeld eines Sektors grenzen nicht bündig aneinander, sondern es befinden sich Lücken zwischen ihnen. Regelmäßige Muster in diesen Lücken werden zur Synchronisation eines »Phase Locked Oszillator« (PLO) verwendet. Dieser erzeugt aus den gelesenen und in digitale Signale umgewandelten Lese-Rohdaten den »Lesetakt«, mit dessen Hilfe diese Rohdaten decodiert werden. Um aus dem ankommenden Bitstrom den richtigen Bytestrom zu erzeugen, werden sowohl Adreßfelder als auch Datenfelder von vereinbarten Mustern, den sogenannten Adreßmarken, eingeleitet, die unter anderem zur Bytesynchronisation dienen.

Sowohl Adreß- als auch Datenfeld sind durch ein Polynom gesichert, dessen Summe jeweils unmittelbar im Anschluß an die Nutzinformation in diesen Feldern aufgezeichnet wird. Beim Lesen wird die Summe nach derselben Vorschrift wie beim Schreiben gebildet. Sind die aufgezeichnete und die beim Lesen gebildete Summe identisch, so ist die Wahrscheinlichkeit sehr hoch, daß die Daten korrekt gelesen wurden. Für die Auswertung des Polynoms sowie für das Einleiten der weiteren Aktionen benötigt der Controller Zeit, die ihm durch die Lücken zur Verfügung gestellt werden.

Bei Schreibtransfer muß der Kopf von Lesen auf Schreiben, beziehungsweise umgekehrt, umgeschaltet werden. Die hierdurch entstehenden Stoßstellen mit eventuell aufgezeichneten Störimpulsen werden ebenfalls in die Lücken gelegt. Dies geht so vor sich, daß der Controller nach Auswertung des Adreßfeldes an einem definierten Ort der folgenden Lücke den Schreibstrom einschaltet, Synchronisierbytes und Adreßmarke des Datenfeldes selbst erzeugt und aufzeichnet, anschließend die Nettodaten transferiert und auf den Datenträger schreibt (wobei jedes Byte in die Bildung des Sicherungspolynoms einbezogen wird), die Summe des Polynoms hinzufügt und schließlich in der folgenden Lücke den Schreibstrom abschaltet.

Sieben Aufgaben müssen mindestens erfüllt werden

Aus den bisher skizzierten Vorgängen bei Informationstransfers von und zu rotierenden Speichern lassen sich folgende Aufgaben ableiten, die ein Controller unbedingt erfüllen muß:

— Erzeugen aller Formatdaten (Adreßfelder, Lücken, Datenfelder, Adreßmarken, Summe des Sicherungspolynoms) und deren bündige Aufzeichnung beim Formatieren des Datenträgers;

— Parallel-Seriell-Umsetzen der Nutzdaten beim Schreiben;

— Seriell-Parallel-Umsetzung der Nutzdaten beim Lesen mit vorheriger Bytesynchronisierung;

— Erzeugen der Summe von Sicherungspolynomen und Erkennen von Lesefehlern;

— Aufsuchen und Identifizieren des gewünschten Speicherorts durch Positionieren des Schreib-/Lesekopfes auf die richtige Spur und Identifikation des Sektors durch Auswerten des Adreßfeldes;

— zeitrichtiges Umschalten des Schreib-/Lesekopfes bei Schreibtransfer mit dem Hinzufügen einiger Formatdaten (Synchronisierbytes, Adreßmarke, Summe des Sicherungspolynoms) zu den Nutzdaten;
— Ausfiltern der Nettodaten aus den Formatdaten und ihre Weiterleitung beim Lesen.

Neben diesen Basisaufgaben verfügt ein Controller im allgemeinen über weitere Funktionen. Deren Art und Umfang hängen in einem starken Maße von der Schnittstelle und Art des Laufwerkes und des Gastrechners sowie vom gebotenen Komfort und Leistungsumfang des Controllers ab.

Die gängige Schnittstelle ist recht einfach

Die gängige Schnittstelle für Floppy-Disk-Laufwerke ist recht einfach gestaltet und legt die meisten Steuerungsaufgaben in die Hand des Controllers. Dieser wählt den Kopf an, falls mehrere vorhanden sind, steuert die Positionierung, indem er ein Richtungssignal und zeitlich richtig dosierte Impulse für einen Schrittmotor ausgibt und so weiter. Auch die Codierung und Decodierung übernimmt er. Bei »doppelter Aufzeichnungsdichte« (MFM) führt er gegebenenfalls eine bitmusterabhängige Vorkompensation der Schreibimpulse durch.

Bei Festplattenlaufwerken mit floppyähnlicher Schnittstelle sind die Aufgaben grundsätzlich ähnlich wie bei den Floppy-Disk-Laufwerken, nur werden die einzuhaltenden Zeitbedingungen wegen der merklich höheren Datenrate spürbar kritischer. Das gilt für die Auswertung der Adreßfelder und die zeitrichtige Umschaltung des Schreib-/Lesekopfes ebenso wie für den Datenkanal (PLO-Dimensionierung, Vorkompensation, Adreßmarkenerkennung mit Bytesynchronisierung, Transfergeschwindigkeit und so weiter).

Eine merkliche Entlastung des Controllers ist dann gegeben, wenn Laufwerke mit integriertem PLO verwendet werden. Laufwerke mit einer mikroprozessorähnlichen Schnittstelle (Disk-Bus-, ANSI-Schnittstelle) übernehmen meist sowohl die Codierung und Vorkompensation als auch die Lesetakterzeugung (PLO) und Decodierung selbst.

Der Datenkanal ist jedoch nach wie vor bitseriell, das heißt Bytesynchronisierung, Seriell-Parallel- und Parallel-Seriell-Umsetzung und so weiter ist nach wie vor Aufgabe des Controllers. Bei diesem Schnittstellentyp übernimmt das Laufwerk über einen Datenbus Kommandos und liefert über ihn Statusinformationen. Nach Erhalt der Zieldaten steuert es die Positionierung und Kopfauswahl selbst.

Bei Festplatten dürfen im Interesse einer niedrigen Ausschußquote bei der Produktion die Datenträger bereits bei der Auslieferung über einige feste Defektstellen verfügen. Sinnvollerweise schließt man diese Stellen beim Betrieb vom Zugriff aus. Dies kann per Systemsoftware (Eintrag in Buchführung) ohne oder mit Unterstützung des Controllers (zum Beispiel hardwaremäßiges Kennzeichnen des Defektes) erfolgen. Besser ist es, wenn der für die Systemsoftware sichtbare Externspeicherbereich durchgehend fehlerfrei erscheint. Dies läßt sich durch Zuweisung eines Ersatzbereiches für einen Defektbereich erreichen. Komfortable Controller stellen hierfür Hilfsmittel zur Verfügung (Formatieren von Ersatzsektoren, Ersatzspuren) und greifen im Idealfall ohne Belastung der Systemsoftware selbständig auf den Ersatzbereich statt auf den Defektbereich zu.

Für Fehler, die während des Betriebes dynamisch entstehen, ist es von großer Bedeutung, ob der Controller das Datenfeld mit einem fehlererkennenden (CRC-) oder fehlerkorrigierenden (ECC-)Code absichert. Während bei CRC nur der Nachweis möglich ist, daß ein Fehler aufgetaucht ist, besteht bei ECC die Möglichkeit, den Fehler zu identifizieren und zu korrigieren. Je nach verwendetem Code lassen sich 4 bis 11 zusammenhängende fehlerhafte Bits wieder richtig rekonstruieren.

Bei den heute gängigen Floppy-Formaten ist CRC üblich, während bei Festplatten der Mehraufwand für ECC in vielen Fällen aufgebracht wird.

Zur Entkoppelung der Zeitbedingungen zwischen Laufwerks- und Rechnerschnittstelle befindet sich in der Regel zwischen diesen ein Puffer, in dem die zu lesenden beziehungsweise zu schreibenden Daten zwischengespeichert werden. Bei dem verbreiteten einfachen Sektorpuffer ist die Pufferorganisation relativ einfach, und es werden an die rechnerseitige Schnittstelle keine besonderen Geschwindigkeitsanforderungen gestellt.

Allerdings läßt sich hiermit die vom Laufwerk her mögliche Datenrate auch bei einer schnellen Rechnerschnittstelle nicht voll ausnutzen (bestenfalls halbe Transferrate), da Transfers zwischen Laufwerk und Puffer nicht simultan zu Transfers zwischen Puffer und Rechner stattfinden können. Hingegen ist bei einem Wechselpuffer die Nutzung der vollen Datenrate möglich, wobei jedoch dessen Organisation nicht einfach ist und entsprechend hohe Anforderungen an die einzuhaltenden Zeitbedingungen der steuernden Einheit (Controller oder Adapter) gestellt werden.

Ein Großteil der Funktionen auf einem Chip

Ein FIFO als Pufferspeicher führt zu einem sehr guten Zeitverhalten, jedoch verlangt es von der Rechnerschnittstelle, daß die mittlere Datenrate gleich der Laufwerksdatenrate ist und daß zeitliche Störungen an dieser Schnittstelle ein gewisses Maß nicht überschreiten. Dieser Puffertyp macht, abgesehen von speziellen Lösungen, die Fehlerkorrektur über ECC nicht oder nur sehr umständlich möglich.

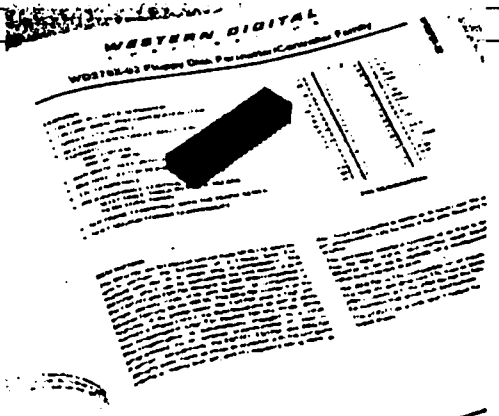
Das Zeitverhalten bei Transfers ist dann nicht unwesentlich, wenn Floppy-Disks als Back-up-Medium verwendet werden. Ein einfacher Sektorpuffer halbiert mindestens die mögliche Datenrate, wodurch die Zeit für den Back-up-Vorgang merklich ansteigt.

Ein Großteil der Funktionen eines Floppy-Disk-Controllers ist heute als LSI-Chip erhältlich. Allerdings müssen PLO, Vorkompensation, Puffer und ähnliches extern realisiert werden. Die angebotenen Controllerbaugruppen (meist mit dem LSI-Chip) unterscheiden sich bezüglich des Umfanges an Vorverarbeitung, der Leistungsfähigkeit und des Komforts.

Controller für Festplattenlaufwerke sind heute noch grundsätzlich als eine oder mehrere Baugruppe(n) ausgeführt. Bei einigen Typen werden zusätzlich auch Floppy-Disk-Laufwerke oder Magnetband-Kassettenlaufwerke gesteuert. Dies ergibt kompakte und wirtschaftliche Lösungen.

Armin Schwarz

Rolf Keller



Der Floppy-Disk-Controller-Chip WD2797 von Western Digital übertrumpft andere Bausteine ähnlicher Art durch seine integrierten Schaltungsteile für Datenseparation und Schreib-Vorkompensation. Damit werden einige Bauelemente bei der äußeren Beschaltung eingespart, und man kommt mit der 'unberechenbaren' Analogtechnik kaum noch in Berührung. Der 2797 wird zum Beispiel im c't 86 und im c't 68000 eingesetzt.

Floppy-Disk-Controller WD2797

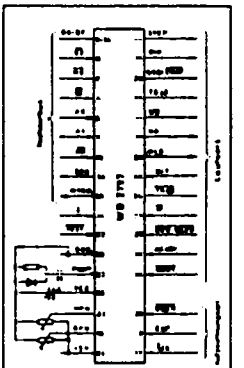


Bild 1. Pinbelegung des WD2797, geordnet nach Funktionsgruppen

Der 2797 ist ein Mitglied der Bausteinfamilie 279x, die ihrerseits eine (softwarekompatible) Weiterentwicklung der verbreiteten Familie 179x ist. Behandelt wird hier nur der Typ 2797, die anderen Typen unterscheiden sich davon geringfügig. Zur Klärung dieser Unterschiede sollte man das Datenblatt zu Rate ziehen, wenn es um die letzten Feinheiten geht.

Alle Typen der Familie 279x können 8"- und 5,25"-Laufwerke (und dazu kompatible Mikrofloppy-Laufwerke) in Single und Double Density ansteuern. Dabei können alle verbreiteten IBM-ähnlichen Datenformate benutzt werden.

Beschaltung

Bild 1 zeigt die Pinbelegung des 2797, geordnet nach Funktionsgruppen. Die auf dem Chip integrierten Analogschaltungen für Datenseparation und Schreib-Vorkompensation benötigen einige Bauteile für den Abgleich (Bild 1 links unten). Bei der Aktivierung des Eingangs $\overline{\text{TEST}}$ kann der Abgleich ohne Umstände in der betriebsbereiten Schaltung erfolgen.

Die Laufwerksignale können in der Regel direkt mit den Pins der ICs verbunden werden, die die gleiche Signalbezeichnung tragen. Besondere Beachtung verdienen die Signale HLT (Head Load Timing = Kopfaufsetzzeit) und READY. HLT teilt dem 2797 mit, daß die Beruhigungszeit nach dem Aufsetzen des Kopfes abgelaufen ist. Die meisten Laufwerke stellen dieses Signal nicht zur Verfügung; man muß es dann mit einem Zeitglied erzeugen. Das Signal READY soll die Lese-/Schreib-Bereitschaft des Laufwerks anzeigen, das heißt Diskette eingelegt, Tür geschlossen und Solldrehzahl erreicht. Meistens liefern 5,25"-Laufwerke dieses Signal nicht. Vielfach sieht man Schaltungen, die die Signale HLT und READY nicht unter allen Bedingungen (zum Beispiel beim Umschalten zwischen zwei Laufwerken) korrekt erzeugen. Dieser Fehler bleibt oft unerkannt, da der 2797 einen Lese-/Schreib-Vorgang erst nach fünf Plattenumdrehungen abbricht. Diese Zeit ist gewöhnlich länger als die Hochlaufzeit des Motors und die Beruhigungszeit beim Kopfaufsetzen, so daß am Ende meist alles gutgeht. Tatsächlich funktioniert es daher sogar, wenn man HLT und READY fest auf High-Pegel legt, aber empfehlenswert ist dieses Verfahren nicht:

Der Eingang $\overline{\text{DDEN}}$ des ICs dient zum Umschalten auf Double-Density-Betrieb, der Eingang ENP zum Einschalten der Schreib-Vorkompensation. Mit dem Eingang 5/8 kann man zwischen 5,25"- und 8"-Laufwerken wählen, muß aber gleichzeitig den Takt Φ umschalten (1 MHz bei 5,25", 2 MHz bei 8"). $\overline{\text{MR}}$ ist der Reset-Eingang des Bausteins.

Die fünf adressierbaren Register des Bausteins werden wie folgt angesprochen:

CS	A1	A0	RE	WE
0	0	0	Status	Befehl
0	0	1	Spur	Spur
0	1	0	Sektor	Sektor
0	1	1	Daten	Daten

Der Ausgang DRQ (Data Request = Datenanforderung) kann bei reinem Polling-Betrieb unbenutzt bleiben, ansonsten wird er mit einem DMA-Baustein oder über eine Zusatzlogik mit dem $\overline{\text{WAIT}}$-Eingang der CPU verbunden. Voll DMA-fähig ist der 2797 allerdings nicht: Ihm fehlt dazu ein DMA-Acknowledge-Eingang. Alternativ kann man den Ausgang DRQ auch als Interrupt-Ausgang (Daten-Interrupts) betrachten.

Der Ausgang INTRQ (Interrupt Request = Interrupt-Anforderung) kann beim Polling ebenso unbenutzt bleiben. Sein Hauptzweck ist es, beim DMA-Betrieb das Befehlsende zu melden. Im Fehlerfall würde der DMA-Baustein sonst 'ewig' auf Daten warten.

Leider 'kennt' der 2797 nur ein einziges Laufwerk, das heißt, er hat nur für dieses eine 'Drive' Anschlußpins und interne Register. Um mehrere Laufwerke anschließen zu können, ist ein zusätzlicher Ausgabeport nötig, mit dem diese selektiert werden können. Mit den übrigen Portbits kann man dann gleich die Signale $\overline{\text{DDEN}}$, ENP, 5/8 und am besten auch $\overline{\text{MR}}$ per Software steuern.

Programmierung

Die möglichen Befehle (Tabelle 1) sind in vier Typen eingeteilt. Der Befehl 'Interrupt' nimmt eine Sonderstellung ein. Befehle der Typen 1 bis 3 erteilt man einfach durch Schreiben des entsprechenden Codes in das Befehlsregister. Dies darf aber nur dann geschehen, wenn der 2797 gerade nichts zu tun hat (BUSY-Bit im Statusregister = 0). Die Spur- und Sektorregister müssen vor der Ausgabe des Befehls geladen werden, sofern die Werte für die Ausführung des Befehls benötigt werden. Falls man mit mehreren Laufwerken arbeitet, muß man beim Wechsel auf ein anderes Laufwerk Spur- und Sektorregister entsprechend umladen, da der 2797 diese Werte nur für ein Laufwerk speichern kann.

Die Befehlsbearbeitung beginnt mit dem Setzen des Statusregisters. Der Baustein setzt das BUSY-Bit, löscht die Fehlerbits und aktualisiert die übrigen Bits gemäß Tabelle 2 beziehungsweise 3. Dabei ist wichtig, daß die Statusbits 0 und 7

Bit								Typ	Befehl	Aktion
7	6	5	4	3	2	1	0			
0	0	0	0	K	P	R1	R0	1	Nullstellung	Positionieren auf Spur 0
0	0	0	1	K	P	R1	R0	1	Suchen	Positionieren auf Spur n; Datenregister = n, Spurregister = aktuelle Spur
0	0	1	A	K	P	R1	R0	1	Spur weiter	Positionieren auf Nachbarspur; Richtung wie beim letzten Positionierbefehl
0	1	0	A	K	P	R1	R0	1	Spur vor	Positionieren auf nächsthöhere Spur
0	1	1	A	K	P	R1	R0	1	Spur zurück	Positionieren auf nächstniedrigere Spur
1	0	0	M	L	W	S	0	2	Sektor lesen	Sektor n der aktuellen Spur lesen; Sektorregister = n Kopf muß gemäß Spurregister positioniert sein
1	0	1	M	L	W	S	A0	2	Sektor schreiben	Sektor n der aktuellen Spur schreiben; Kopf muß gemäß Spurregister positioniert sein
1	1	0	0	0	W	S	0	3	Adreßfeld lesen	Nächstes vorbeikommendes Sektoradreßfeld der aktuellen Spur lesen
1	1	1	0	0	W	S	0	3	Spur lesen	Gesamte aktuelle Spur transparent lesen
1	1	1	1	0	W	S	0	3	Spur formatieren	Gesamte aktuelle Spur formatieren
1	1	0	1	I3	I2	I1	I0	4	Interrupt	siehe Text

Bezeichnungen zu den Befehlen

K (Kopf)
1: Zu Beginn des Befehls den Kopf aufsetzen
0: Zu Beginn des Befehls den Kopf abheben

P (Prüfung)
1: Nach Erreichen der Zielspur wird automatisch durch Lesen der Adreßfelder auf der Platte geprüft, ob es sich wirklich um die gewünschte Spur handelt, deren Nummer im Spurregister steht
0: Keine Prüfung

A (Aktualisieren)
1: Das Spurregister wird bei Positioniervorgängen automatisch aktualisiert (weitergezählt).
0: Spurregister bleibt unverändert

M (Mehrere Sektoren)
1: Beginnend mit dem vorgegebenen Sektor werden automatisch Sektoren mit den nachfolgenden Nummern so lange bearbeitet, bis das Spurende erreicht ist.
0: Nur der eine vorgegebene Sektor wird bearbeitet

W (Wartezeit)
1: Nach Aktivierung des Ausgangs HLD (Kopf aufsetzen) läuft eine Wartezeit (15 ms bei 8", 30 ms bei 5,25") ab, bevor ersatzweise der Eingang HLT (Kopf ist aufgesetzt?) abgefragt wird.
0: Keine Wartezeit

S (Seitenwahl)
1: Seite 1 anwählen
0: Seite 0 anwählen

A0 (Adreßmarke)
1: Adreßmarke $F8 schreiben (deleted data address mark)
0: Adreßmarke $FB schreiben (normal data address mark), dies ist bei dem meisten Betriebssystemen der Normalfall

L (Länge eines Sektors)

	Kodierung im Adreßfeld:			
	00	01	10	11
L = 0:	256	512	1024	128
L = 1:	128	256	512	1024

L = 1 bei IBM Format

R (Schrittrate bei Spurwechsel)

R1	R0	
0	0	6 ms
0	1	12 ms
1	0	20 ms
1	1	30 ms

Die angegebenen Zeiten gelten für 5,25". Bei 8" halbieren sich die Zeiten

I (Interruptbedingung)

I0: Wechsel nicht READY → READY
I1: Wechsel READY → nicht READY
I2: Index-Impuls
I3: sofortiger Interrupt

Tabelle 1. Übersicht über die Befehle des FDC 2797

zu jedem Zeitpunkt dieselbe Bedeutung haben; bei den Bits 1 bis 6 ist das nicht der Fall.

Nach der Ausgabe eines Befehls muß man mit dem Auslesen des Statuswortes etwas warten (max. 56 μs): Der Chip 'denkt' nicht so schnell. Ansonsten kann aber das Statusregister zu jedem Zeitpunkt gelesen werden.

Ist die Ausführung eines Befehls beendet, so wird das BUSY-Bit wieder gelöscht. Trat während der Ausführung des Befehls ein Fehler auf, ist das jeweilige Fehlerbit im Statusregister gesetzt.

Datentransfer

Weil der 2797 nur ein oder zwei Datenbytes intern speichern kann, muß er jedes Byte von beziehungsweise zur Diskette synchron zu deren Drehzahl abgenommen beziehungsweise geliefert bekommen. Dies geschieht durch Lesen/Schreiben des Datenregisters. Der 2797 fordert dazu pro Byte einmal auf, indem er das DRQ-Bit (Data Request = Datenanforderung) im Statusregister (Tabelle 3) setzt. Der Ausgangspin DRQ zeigt diese Anforderung ebenfalls an. Die DRQ-Anforderung wird beim Zugriff auf das Datenregister automatisch wieder gelöscht. Die maximal zulässige Reaktionszeit beträgt beim Schreiben 11,5 μs (8" DD), 23,5 μs (8" SD/5,25" DD) oder 47 μs (5,25" SD). Um hier mithalten zu können, ist ein schnelles Programm (Polling unter Interrupt-Sperre) oder Zusatzhardware (zum Beispiel DMA mit Hilfe des DRQ-Pins) erforderlich. Wird die Zeit 'verpaßt', so ist Datenverlust die Folge. Der 2797 setzt dann das Fehlerbit 2 im Statusregister (Tabelle 3).

Interrupts

Nach der Ausführung jedes Befehls (mit Ausnahme eines 'Interrupt'-Befehls mit I3 = 0) aktiviert der 2797 seinen INTRQ-Ausgang. Dabei ist es gleichgültig, ob der Befehl erfolgreich ausgeführt wurde oder nicht. Eine Interrupt-Sperre ist nicht möglich.

Zusätzlich kann man noch mit Hilfe eines 'Interrupt'-Befehls weitere Interrupt-Bedingungen vorgeben (Tabelle 1). Der Befehl '$D1' bewirkt zum Beispiel, daß dann ein Interrupt erzeugt wird, wenn der Pegel am READY-Eingang von high auf low wechselt. 'I3 = 1' löst sofort einen Interrupt aus. Damit kann man den Durchlauf der Interrupt-Routine erzwingen, wenn das programmtechnisch erforderlich ist.

Alle Interrupts werden quittiert durch Lesen des Statusregisters oder durch Schreiben in das Befehlsregister. Ein durch 'I3 = 1' ausgelöster Interrupt kann jedoch nur durch den 'Interrupt'-Befehl '$D0' quittiert werden.

Eine zweite Wirkung des 'Interrupt'-Befehls ist es, daß er die Ausführung jedes anderen Befehls unmittelbar abbricht. Dies ist möglich, weil der 'Interrupt'-Befehl der einzige Befehl ist, der ausgegeben werden darf, während ein anderer in Bearbeitung ist (BUSY = 1).

Um es ganz kompliziert zu machen, beeinflußt der 'Interrupt'-Befehl auch noch das Statusregister. War gerade ein anderer Befehl in Bearbeitung, so wird nur das BUSY-Bit auf null gesetzt, die übrigen Bits bleiben erhalten. War aber der 2797 gerade nicht 'beschäftigt', so wechselt das Statusregister in den Zustand, den es nach der Ausführung eines Befehls vom Typ 1 hat (Tabelle 2).

Verwendung der Befehle

Vor jedem Datenzugriff muß der Kopf des Laufwerks auf die gewünschte Spur gestellt werden. Dazu sind die Befehle des Typs 1 (Positionierbefehle) bestimmt. Für einen 'normalen' Disk-Treiber benötigt man davon nur die Befehle 'Nullstellung' und 'Suchen'.

Vor der Ausgabe des Befehls 'Suchen' muß das Spurregister die Nummer der Spur enthalten, auf der der Kopf gerade steht. Nach dem Einschalten und nach Fehlern ist das nicht unbedingt der Fall. Man gibt dann zunächst den Befehl 'Nullstellung'. Dabei werden so lange Schritte in Richtung Spur

Bit	
7	Laufwerk nicht READY Das Bit zeigt den invertierten Zustand des READY-Eingangs.
6	Diskette schreibgeschützt Das Bit zeigt den invertierten Zustand des WPRT-Eingangs.
5	Kopf aufgesetzt Das Bit zeigt an, daß die Signale HLD und HLT auf High-Pegel liegen.
4	Positionierfehler Das Bit zeigt an, daß nach der Ausführung eines Befehls mit gesetztem P-Bit der Kopf nicht auf der richtigen Spur steht.
3	Prüfsummenfehler Das Bit zeigt an, daß bei der Prüfung nach einem Befehl mit gesetztem P-Bit ein Sektor-Adreßfeld mit verfälschter Information gefunden wurde.
2	Spur 00 Das Bit zeigt den invertierten Zustand des TR00-Eingangs.
1	Index Impuls Das Bit zeigt den invertierten Zustand des IP-Eingangs.
0	Baustein BUSY Das Bit zeigt an, daß der Baustein gerade mit der Ausführung eines Befehls beschäftigt ist.

Tabelle 2. Statusregister nach Befehlen vom Typ 1

Bit	
7	Laufwerk nicht READY Das Bit zeigt den invertierten Zustand des READY-Eingangs.
6	Diskette schreibgeschützt Nur bei den Befehlen 'Sektor schreiben' und 'Spur formatieren', sonst 0.
5	Art der Adreßmarke Nur beim Befehl 'Sektor lesen', sonst 0. Das Bit zeigt an, welche Daten Adreßmarke gelesen wurde. 0: Adreßmarke $FB (normal data address mark); 1: Adreßmarke $F8 (deleted data address mark).
4	Sektor nicht gefunden Nur bei den Befehlen 'Sektor lesen', 'Sektor schreiben' und 'Adreßfeld lesen', sonst 0. Das Bit zeigt an, daß der Baustein kein Sektor-Adreßfeld mit passender Sektor- Spur- und Seitennummer einschließlich korrekter Prüfsumme gefunden hat.
3	Prüfsummen-Fehler Nur bei den Befehlen 'Sektor lesen', 'Sektor schreiben' und 'Adreßfeld lesen', sonst 0. Das Bit zeigt an, daß der Baustein anhand der Prüfsumme einen Fehler auf der Platte erkannt hat. Ist Bit 4 ebenfalls gesetzt, so lag der Fehler in einem Sektor-Adreßfeld, andernfalls in einem Datenfeld.
2	Datenverlust Das Bit zeigt an, daß der Baustein nach einer Anforderung über das Statusbit 1 nicht rechtzeitig (schritthaltend mit der Umdrehung der Diskette) durch Lesen bzw. Schreiben des Datenregisters bedient wurde.
1	Datenanforderung Das Bit fordert zum Lesen des nächsten Datenbytes (nach Lesebefehlen) aus dem Datenregister bzw. zum Schreiben des nächsten Bytes (nach Schreibbefehlen) in das Datenregister auf.
0	Baustein BUSY Das Bit zeigt an, daß der Baustein gerade mit der Ausführung eines Befehls beschäftigt ist.

Tabelle 3. Statusregister nach Befehlen von Typ 2 und 3

00 ausgeführt, bis das Signal TR00 (vom Endschalter des Laufwerks) das Erreichen von Spur 00 meldet. Damit ist die aktuelle Position des Kopfes bekannt; der 2797 selbst setzt sein Spurregister auf 00.

Vor der Ausgabe des Befehls 'Suchen' lädt man die Nummer der zu suchenden Spur in das Datenregister. Der Baustein vergleicht sie mit der Nummer der momentanen Spur im Spurregister und führt so viele Schritte in der richtigen Richtung aus, bis die gewünschte Spur erreicht ist. Damit steht im Spurregister dann auch die Nummer der neuen, nunmehr aktuellen Spur.

Jedes Laufwerk verträgt nur eine bestimmte Schrittrate und kommt 'aus dem Tritt', wenn man sie überschreitet. Deshalb müssen die Bits R1 und R0 bei jedem Positionierbefehl korrekt gesetzt werden.

Nach einem Positionierbefehl folgt sinnvollerweise ein Schreib- oder Lesebefehl, für den der Kopf aufgesetzt sein muß. Um die Beruhigungszeit beim Aufsetzen zu sparen, kann man den Kopf mit Hilfe des 'V-Bits' schon zu Beginn des Suchens aufsetzen. Das Setzen des 'P-Bits' ist meist nicht sinnvoll, da die Ausführung der Prüfung Zeit kostet. Bei den Befehlen des Typs 2 wird außerdem die Spurnummer automatisch kontrolliert.

Datenzugriff

Das Schreiben und Lesen eines Sektors wird durch die Befehle des Typs 2 ermöglicht. Zuvor muß der Kopf auf die gewünschte Spur gestellt worden sein. Vor der Befehlsausgabe lädt man das Sektorregister mit der gewünschten Sektornummer. Das Spurregister steht meist wegen des vorangegangenen Positionierbefehls schon richtig. Zu Beginn wählt der 2797 über den Ausgang SSO (Side Select Output = Seitenwahl) die durch das 'S-Bit' vorgegebene Plattenseite an und setzt den Kopf auf. Dann liest er die Spur (auch bei 'Sektor schreiben') so lange, bis ein Sektoradreßfeld erscheint, das die richtige Spur-, Sektor- und Seitennummer einschließlich korrekter Prüfsumme (CRC) enthält. Falls der Controller innerhalb von fünf Plattenumdrehungen kein passendes Sektoradreßfeld findet, bricht er ab und setzt das Fehlerbit 4 (zusätzlich auch Bit 3) im Statusregister. Dieser Fehler kann durch schlechte Positionierung verursacht worden sein. Ein 'defensives' Programm wird deshalb 'Nullstellung' und 'Suchen' befehlen und dann einen neuen Versuch starten.

Im Normalfall wird ein passendes Sektoradreßfeld gefunden, und der 2797 wartet auf den Beginn des folgenden zugehörigen Datenfeldes, um direkt mit dem Schreiben beziehungsweise Lesen zu beginnen (DRQ-Anforderung). Die Länge des Sektors (Anzahl der Datenbytes) ist beim Formatieren der Diskette festgelegt worden; der Baustein findet diese Angabe (Tabelle 1, L-Bit) im Sektoradreßfeld.

Adreßfeld lesen

Der Befehl 'Adreßfeld lesen' liest das nächste in der momentanen Spur am Kopf vorbeilaufende Sektoradreßfeld und liefert seinen Inhalt (6 Byte: Spur-, Seiten-, Sektornummer, Sektorlänge, Prüfsumme davon) als Daten ab. Im Normalfall kann man damit nicht allzuviel anfangen. Eine Anwendung wäre zum Beispiel ein schnelles 1:1-Kopierprogramm, das jeweils eine komplette Spur in einer einzigen Plattenumdrehung liest beziehungsweise schreibt. Der Befehl macht es dann möglich, unmittelbar an der augenblicklichen Stelle mitten in der Spur zu beginnen. Bei den folgenden Befehlen ('Sektor lesen/schreiben') fängt man dann nicht mit Sektor 1 an, sondern mit demjenigen, der gerade vorbeiläuft.

Spur lesen

Der Befehl 'Spur lesen' ermöglicht es, eine komplette Spur in einem Stück transparent (d. h. einschließlich aller Adreßmarken, Datenbytes, Prüfsummen, Synchronisationsbytes und Füllbytes) zu lesen. Die Auswertung des Gelesenen ist aber recht kompliziert, insbesondere dadurch, daß einige Bytes 'angeschnitten' sein können. Der Befehl wird im normalen Betrieb nicht benutzt. Er kann Anwendung finden bei Hardware-Tests, beim Überprüfen von Formatierprogrammen oder zur Untersuchung fremder Aufzeichnungsarten.

Spur formatieren

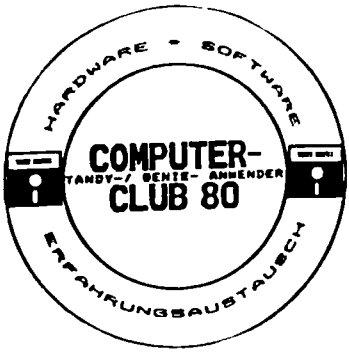
Der Befehl 'Spur formatieren' formatiert die gesamte Spur, auf der der Kopf gerade steht, indem er sie vollkommen neu mit leeren Sektoren beschreibt. Ein vorheriges Kontroll-Lesen, wie etwa bei 'Sektor schreiben', findet nicht statt. Da man normalerweise die gesamte Platte formatiert und damit löscht, ist das nicht gefährlich. Es ist sogar erwünscht, denn nur so kann man eine fabrikneue oder fehlerhaft beschriebene Diskette 'auf Vordermann bringen'. Man beginnt dann mit Spur 00 und geht schrittweise mit Hilfe des 'Spur vor'-Befehls weiter, so daß Positionierfehler nicht zu befürchten sind.

Im Gegensatz zu anderen Bausteinen (zum Beispiel µPD 765) ist das Formatieren beim 2797 ziemlich programmaufwendig. Man muß dem Chip als Daten jeweils den kompletten Spurinhalt übergeben. Um diese Daten zusammensteilen zu können, bedarf es einer genauen Kenntnis der Datenstruktur auf einer Diskette. Es handelt sich dabei um mehrere tausend Bytes (siehe auch unter 'Spur lesen'), nicht etwa nur um Anzahl und Länge der Sektoren. Anders gesagt: Der Baustein 'weiß' gar nicht, wie man eine Platte formatiert. Es wird ihm per Programm mitgeteilt. Der Vorteil dieser Methode ist es, daß der Programmierer freie Hand hat und auch 'ausgefallenere' Formate erzeugen kann.

Probleme ...

Beim Arbeiten mit dem 2797 fielen zwei Probleme auf. Wie schon erwähnt, versucht der Baustein im Fehlerfall fünf Plattenumdrehungen lang, den gewünschten Sektor aufzufinden. Leider mißt er aber diese fünf Umdrehungen nicht zeitlich ab, sondern zählt einfach fünf Indexpulse (Eingang $\overline{IP}$). Falls nun keine Diskette eingelegt ist, das Signal READY aber trotzdem 'high' ist (Tür kann geschlossen sein, Motor kann laufen), so wartet der 2797 'ewig'. Ein ihm anzulastender Fehler ist das strenggenommen nicht, es stört nur bei einigen 'Sparschaltungen'.

Ein echter Fehler zeigt sich dagegen bei den Befehlen 'Sektor lesen', 'Sektor schreiben' und 'Spur lesen'. Dabei darf man nach der Befehlsausgabe zunächst nur die Bits 0 (BUSY) und 1 (Datenanforderung) abfragen. Die übrigen Bits zeigen zu Beginn Fehler an, die nicht aufgetreten sind. Erst nach der Datenanforderung für das allererste Datenbyte (Statusbit 1) oder nach Abbruch des Befehls (Statusbit 0) sollte man die übrigen Statusbits abfragen. Dieser Fehler wirkt sich allerdings nur aus, wenn man den Datentransfer durch Polling abwickelt. □



Ausgabe Juni 89