SIEMENS

# Mikrocomputer Bausteine

SIEMENS

# Mikrocomputer-Bausteine

# SAB 179X/SAB 279X
# Floppy-Disk-Steuerbausteine

## Applikation 3.83

# Einleitung

Floppy Disks sind bei Mikrocomputern und Minicomputern das Massenspeichermedium, das am weitesten verbreitet ist. Dies ist im wesentlichen auf folgende Eigenschaften zurückzuführen: einfache Anwendung, große Zuverlässigkeit, relativ hohe Aufzeichnungsdichte und das mittlerweile große Angebot an Steuerbausteinen für das Interface zwischen Rechner und Floppy-Disk-Laufwerken. Dieses Interface ist eine ziemlich komplexe Schaltung, die bis vor kurzem noch aus ca. 60 TTL-Bausteinen bestand. Demgegenüber bietet die Familie der SAB 179X Floppy-Disk-Steuerbausteine die Möglichkeit, dieses Interface mit nur 6 Bausteinen aufzubauen; mit der Bausteinfamilie SAB 279X benötigt man sogar nur 3 Bausteine.

In dieser Applikation werden die wichtigsten Punkte behandelt, die beim Aufbau einer Floppy-Disk-Steuerung mit einem Baustein SAB 179X/SAB 279X beachtet werden müssen.

Um einen schnellen Einstieg zu erleichtern, wird die Hard- und Software einer ausgeführten und getesteten Lösung mit dem SAB 1797 erläutert. Die Unterschiede im Aufbau einer Steuerung mit einem Baustein SAB 279X sind in einem eigenen Kapitel zusammengefaßt. Wegen der vollen Softwarekompatibilität zwischen SAB 179X und SAB 279X ist die beschriebene Software für jeden Baustein verwendbar.

Zusätzlich werden allgemeine Informationen über Floppy-Disk-Laufwerke, Aufzeichnungsformate, Datentrennung u. ä. geliefert. Damit ist es dem Anwender möglich, sein Floppy-Disk-System optimal den eigenen Anforderungen anzupassen.

# Inhaltsverzeichnis



## Anhang

# 1. Übersicht über die SAB 179X-Bausteinfamilie

Die Familie der SAB 179X-Bausteine setzt sich aus mehreren Floppy-Disk-Steuerbausteinen zusammen, die als Peripheriebausteine für Mikroprozessoren entwickelt wurden. Es handelt sich dabei um LSI-Bausteine in N-Kanal-MOS-Technologie.

Die verschiedenen Bausteine unterscheiden sich in der Ausführung des Datenbusses und der Unterstützung von doppelseitig beschreibbaren Disketten. Die folgende Tabelle zeigt die unterschiedlichen Eigenschaften der einzelnen Bausteine:

| Eigenschaft | SAB 1791 | SAB 1793 | SAB 1795 | SAB 1797 |
|---|---|---|---|---|
| Bus nicht invertierend | | X | | X |
| Bus invertierend | X | | X | |
| Ausgang zur Seitenselektion | | | X | X |

Auf Grund der prinzipiell nur geringfügigen Unterschiede zwischen den Bausteinen werden sie in der Applikation nur als SAB 179X angesprochen.

Bild 1.1 zeigt ein vollständiges Floppy-Disk-Interface, wie es in typischen Mikrocomputersystemen Verwendung findet. Es kann mit einem Baustein SAB 179X mit Hilfe einiger Zusatzbausteine für Datenrückgewinnung, Taktversorgung und Schreibvorkompensation („write precompensation", nur für Disketten mit doppelter Aufzeichnungsdichte) aufgebaut werden. Die SAB 179X haben eine Standardschnittstelle zum Mikrocomputer, die einen problemlosen Anschluß an die meisten Prozessoren gewährleistet. Besonders einfach ist der Anschluß an Bausteine der Familien SAB 8051, SAB 8085 und SAB 8086. Des weiteren wird der Anschluß an Unterbrechungs- und DMA-Steuerbausteine unterstützt. In den meisten Anwendungsfällen kann das Floppy-Disk-Interface mit sechs oder weniger Bausteinen aufgebaut werden.

Der größte Teil der notwendigen Funktionen kann beim SAB 179X durch ein Kommando realisiert werden. Dies sind z. B.: Lesen eines Sektors, Schreiben eines Sektors, Positionieren des Kopfes auf eine bestimmte Spur, Positionieren auf Spur Ø, Formatieren einer Spur u. ä. Die Bausteine unterstützen sowohl FM (Frequenz-Modulation) – bei Disketten mit einfacher Schreibdichte – als auch MFM (Modifizierte Frequenz-Modulation) – für Disketten mit doppelter Schreibdichte. Zusätzlich werden die Signale generiert, die für die Schreibvorkompensation notwendig sind.

Nahezu alle Signale, die ein Floppy-Disk-Laufwerk benötigt, werden vom SAB 179X direkt erzeugt.

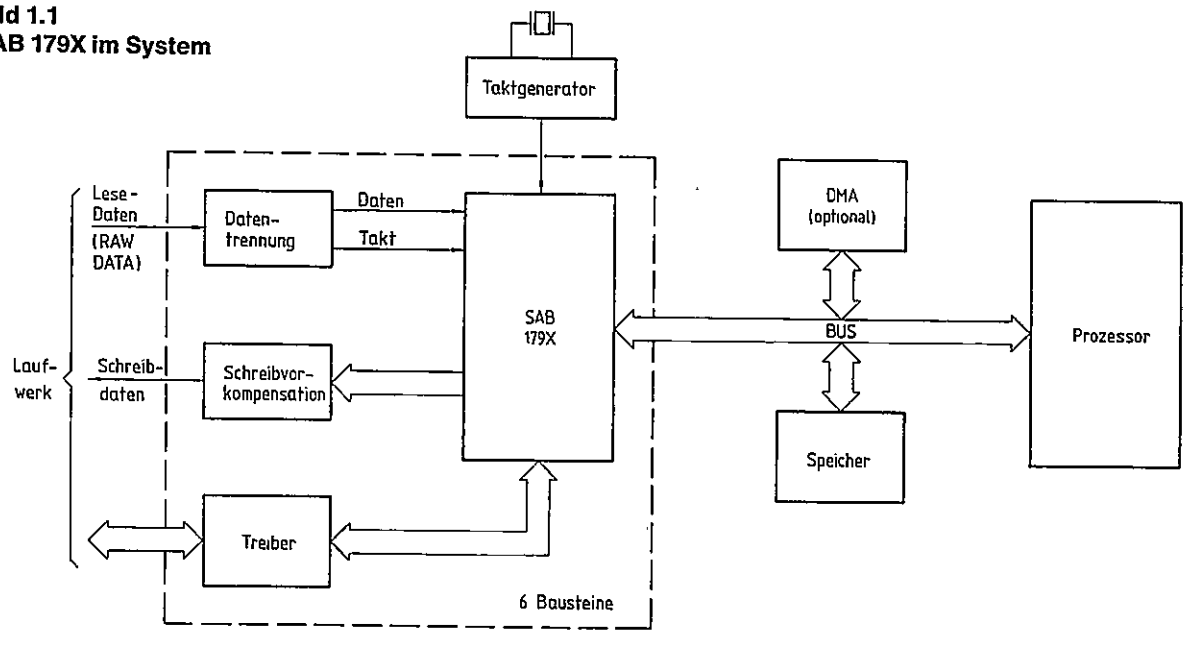
Beispiele dafür sind:
Schreibsignal, Auswahlsignal für die Diskettenseiten, Schrittmotorimpulse und Drehrichtung. Signale, die vom Laufwerk kommen (z. B. Ready, Indexloch, Schreibschutz, Spur-Ø-Erkennung), können ebenfalls direkt angeschlossen werden.

Zusätzlich zu den normalen Befehlen werden sehr komfortable Kommandos, wie Schreiben und Lesen mehrerer Sektoren oder Lesen einer Spur mit allen Bytes (auch den Gap-Bytes), zur Verfügung gestellt. Gerade der letzte Befehl ist sehr nützlich beim Retten von Dateien auf defekten Disketten (z. B. bei CRC-Fehler beim Lesen). Auch die automatische Überprüfung der Sektoradressen ist bei allen Befehlen möglich.

Die Bausteine SAB 1795 und SAB 1797 unterstützen doppelseitig beschreibbare Disketten und können als die am vielseitigsten einsetzbaren Bausteine bezeichnet werden.

Sie sind voll softwarekompatibel und bei der Anschlußbelegung aufwärtskompatibel zu SAB 1791 bzw. SAB 1793.

**Bild 1.1**
**SAB 179X im System**



# 2. Floppy Disks – Material und Aufzeichnungsformate

Eine Floppy Disk besteht aus biegsamer, runder Kunststoffolie, die sich in einer Plastikhulle befindet Die Hülle dient dem Schutz der Diskette, bewirkt eine ständige Selbstreinigung und erleichtert die Handhabung. Das eigentliche Speichermedium ist eine dunne Schicht magnetisch aktiver Metalloxide

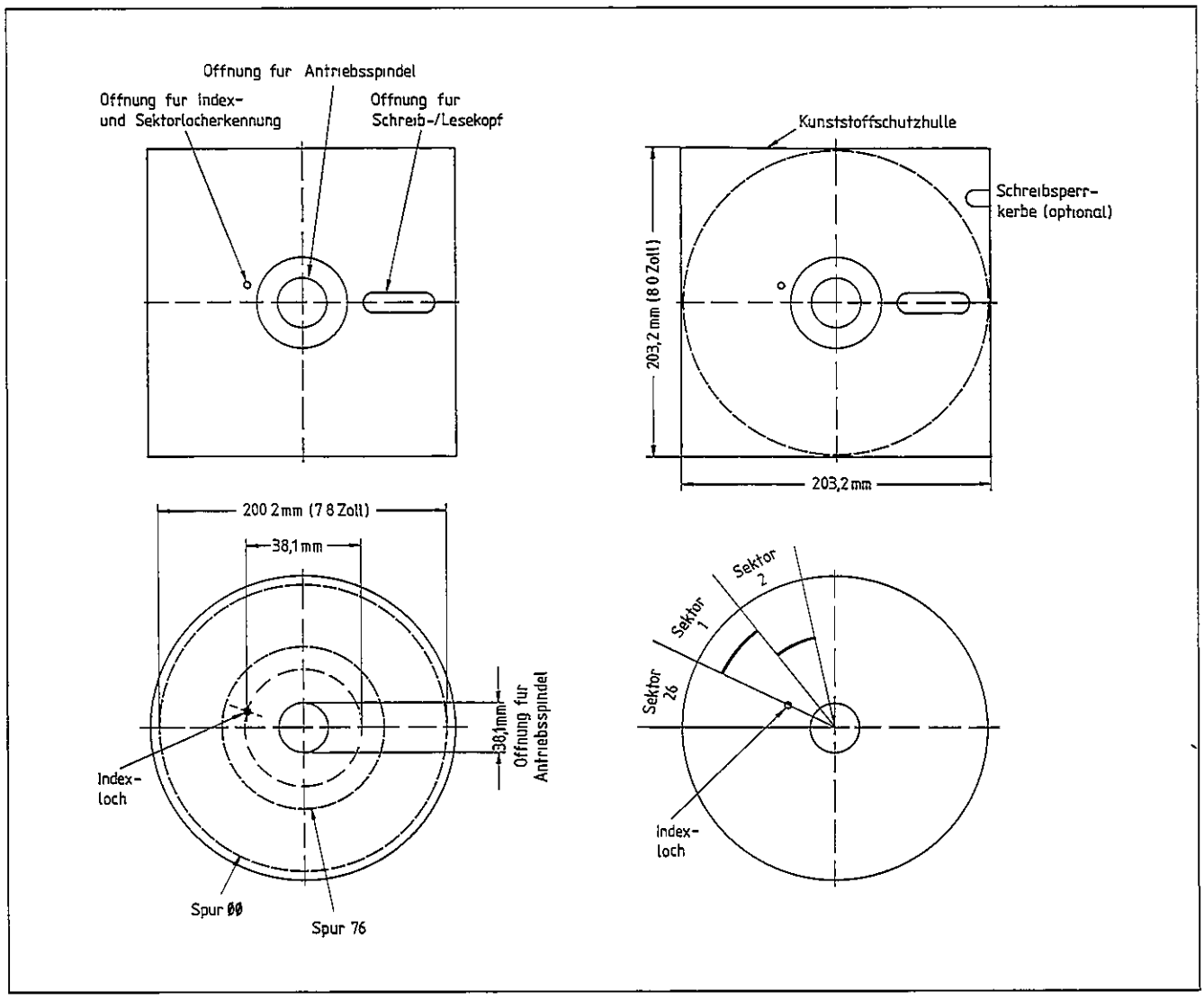
Floppy Disks gibt es in zwei z Zt gangigen Großen mit 8 Zoll Durchmesser (Standard-Floppy) und mit 5¼ Zoll (Minifloppy) In Bild 2 1 sind die wichtigsten Daten einer Standard-8-Zoll-Floppy zusammengestellt. Auf einer Standarddiskette konnen 77 Spuren aufgezeichnet werden, wobei die Spur Ø außen liegt Eine Minifloppy enthalt dagegen nur 36 Spuren. Die Spuren werden radial in eine Zahl von Abschnitten (Sektoren genannt) aufgeteilt. Ublicherweise verwendet man bei Standarddisketten 26 Sektoren pro Spur Der Sektor 1 beginnt dabei nach einer zusatzlichen Markierung auf der Diskette, dem Indexloch Auf diese Weise kann jedem Sektor auf der Floppy eine eindeutige Adresse zugeordnet werden, die aus der Spurnummer und der Sektornummer besteht Das Indexloch erzeugt einen Impuls (Indeximpuls), der dem Steuerbaustein den Beginn des 1. Sektors mitteilt.

**Bild 2.1**
**Eigenschaften einer 8-Zoll-Diskette**

| Parameter | Technische Daten |
|---|---|
| Diskettenart | ANSI-Standard ECMA 54, DIN 66237, DIN 66238 |
| Diskettendurchmesser | 200,2 mm |
| Diskettendicke | 0,08 mm |
| Drehzahl | 360 U/min |
| Umlaufzeit | 166,67 ms |
| Durchschnittl Zugriffszeit pro Spur | 83,33 ms |
| Anzahl der Spuren | 77 |
| Spurdichte pro mm | 1,89 (48 Spuren/Zoll) |
| Schreibdichte | 128,66 bits/mm (3268 bpi) (normale Schreibdichte) 257,32 bits/mm (6536 bpi) (doppelte Schreibdichte) |

**Anmerkung:**

Um eine optimale Zentrierung zu erreichen, mussen die Disketten bei laufender Antriebsspindel eingelegt werden.



Bei sogenannten „hart-sektorierten (hard sectored)" Disketten gibt es zusätzlich dazu beim Beginn jedes Sektors ein weiteres Indexloch. Diese Art der Disketten hat heute jedoch weitgehend an Bedeutung verloren. In den meisten Rechnersystemen werden die „soft-sektorierten (soft sectored)" Disketten verwendet.

Bei dieser Floppy Disk wird die Aufteilung der Spuren in Sektoren (Formatierung) durch die Software vorgenommen.

Zwei Aufzeichnungsformate haben sich dabei als Standards herausgebildet:

- IBM 3740 für Disketten mit einfacher Aufzeichnungsdichte und
- IBM System 34 für Disketten mit doppelter Aufzeichnungsdichte.

Beide Formate sind sowohl im Datenblatt als auch im Anhang C beschrieben.

Eine doppelseitige (double sided) Floppy Disk kann auf beiden Seiten des Trägers beschrieben und gelesen werden. Dazu wird für jede Seite ein eigener Kopf benötigt. Beide Köpfe werden gleichzeitig bewegt, es ist aber nur jeweils einer aktiv. Daduch erreicht man zwar die doppelte Aufzeichnungskapazität, aber die Laufwerke und Disketten sind erheblich teurer. In Bild 2.2 sind die Aufzeichnungskapazitäten und Übertragungsraten der handelsüblichen Disketten zusammengefaßt.

**Bild 2.2**
**Aufzeichnungskapazitäten**

| Größe | Aufzeichnungsdichte | Seiten | Kapazität unformatiert | | Übertragungszeit für ein Byte | Kapazität formatiert | |
|---|---|---|---|---|---|---|---|
| | | | pro Spur | pro Diskette | | pro Spur | pro Diskette |
| 5¼" | Einfach | 1 | 3125 | 109,375* | 64 µs | 2304** | 80,640 |
| 5¼" | Doppelt | 1 | 6250 | 218,750 | 32 µs | 4608*** | 161,280 |
| 5¼" | Einfach | 2 | 3125 | 218,750 | 64 µs | 2304 | 161,280 |
| 5¼" | Doppelt | 2 | 6250 | 437,500 | 32 µs | 4608 | 322,560 |
| 8" | Einfach | 1 | 5208 | 401,016 | 32 µs | 3328 | 256,256 |
| 8" | Doppelt | 1 | 10416 | 802,032 | 16 µs | 6656 | 512,512 |
| 8" | Einfach | 2 | 5208 | 802,032 | 32 µs | 3328 | 512,512 |
| 8" | Doppelt | 2 | 10416 | 1604,064 | 16 µs | 6656 | 1025,024 |

* bei 35 Spuren/Seite
** bei 18 Sektoren/Spur (128 Bytes/s)
*** bei 18 Sektoren/Spur (256 Bytes/s)

## 2.1 Lese- und Schreiboperationen

Der grundsätzliche Aufbau eines Schreib-/Lesekopfes ist in Bild 2.3 gezeigt. Es handelt sich um einen Ring aus magnetischem Material mit einem schmalen Spalt. Die Anregung erfolgt durch eine Spule, die um den Ring gewickelt ist.

Das Schreiben eines Bits erfolgt durch Umkehrung des magnetischen Flusses durch eine schnelle Umpolung des durch die Spule fließenden Stromes. Beim Lesen wird analog dazu ein Bit dann gelesen, wenn eine Umkehr des magnetischen Flusses auf der Diskette durch den Schreib-/Lesekopf registriert wird (siehe ebenfalls Bild 2.3).

## 2.2 Aufzeichnungsarten

Bei Disketten mit einfacher Aufzeichnungsdichte ist die häufigste Aufzeichnungsart die Frequenzmodulation (FM). Jedes aufgezeichnete Bit beansprucht dabei ein Zeitintervall (Bitzelle) von 4 µs (siehe Bild 2.4). Jede Bitzelle beginnt mit einem Taktimpuls. Eine aufgezeichnete logische „1" wird durch das Vorhandensein eines Bits (Umkehr des magnetischen Flusses) in der Mitte der Bitzelle repräsentiert. Entsprechend zeigt das Fehlen dieses Bits eine logische „Ø" an.

Bei Disketten mit doppelter Aufzeichnungsdichte wird meistens MFM (modifizierte Frequenzmodulation) verwendet (siehe Bild 2.5). Hierbei benötigt eine Bitzelle nur noch 2 µs (deshalb doppelte Aufzeichnungsdichte). Die Aufzeichnungskodierung ist allerdings etwas komplizierter, da das Taktbit nicht immer am Beginn jeder Bitzelle erscheint. Es wird nur noch bei aufeinanderfolgenden logischen Nullen eingefügt.

Es ist zu beachten, daß sowohl bei FM als auch bei MFM die kürzeste Zeit zwischen zwei aufgezeichneten Bits (= Flußumkehrungen) 2 µs beträgt. Die Verdoppelung der Aufzeichnungsdichte wird nur durch die Art der Kodierung erreicht.

Die SAB 179X-Steuerbausteine unterstützen sowohl FM- als auch MFM-Kodierungen.

## 2.3 Schreib-/Lesekopf

Der Schreib-/Lesekopf bewegt sich in radialer Richtung über die Diskettenoberfläche. Er wird über eine Spindel von einem Schrittmotor angetrieben. Ein Bewegungsschritt des Motors dreht die Spindel um 15°. Dies bewegt den Kopf um eine Spur weiter. Die Bewegungsrichtung wird über das DIRC-Signal vom Steuerbaustein angegeben. Eine vorgegebene Spur kann also durch eine Serie von Schrittimpulsen angefahren werden. Das kann z. B. durch den SEEK-Befehl des Steuerbausteines sehr einfach durchgeführt werden.
Der gewünschte Sektor auf dieser Spur wird dann durch Lesen der ID-Felder (Identifikationsfelder) der einzelnen Sektoren gefunden.

**Bild 2.3**
**Zusammanhang zwischen den Impulsen und magnetischem Fluß**

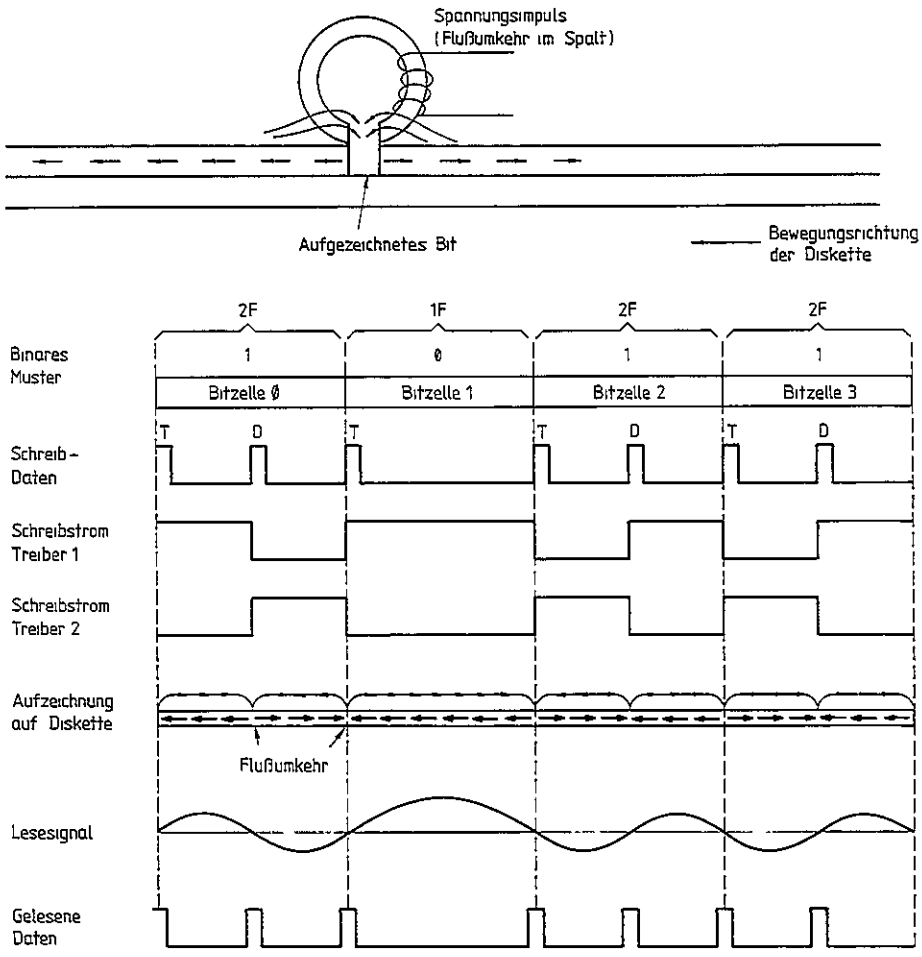
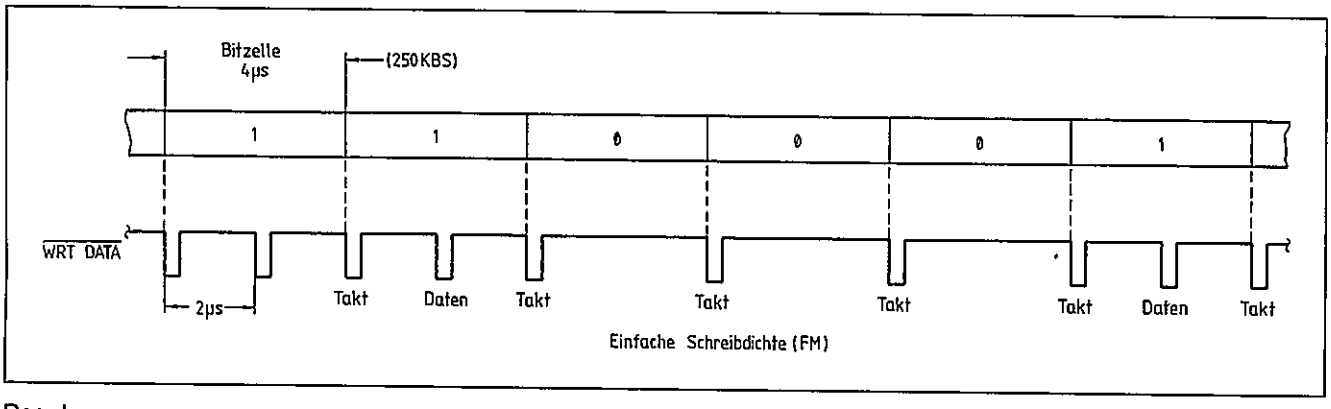
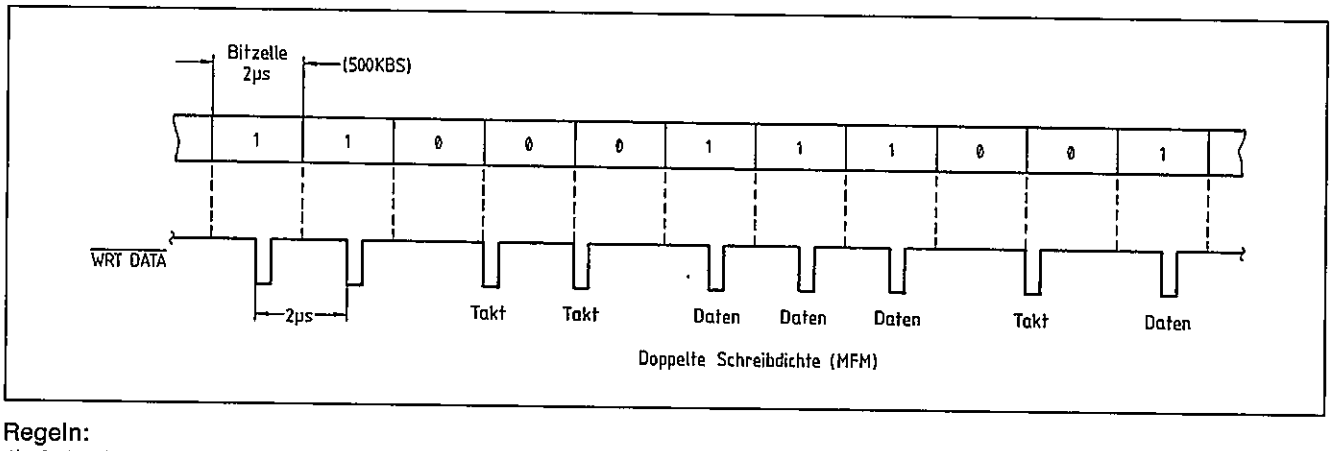


**Bild 2.4**
**Aufzeichnung mit einfacher Schreibdichte**




Regeln:
1) Schreibe Datenbit in die Mitte der Bitzelle, wenn „1" vorliegt
2) Schreibe Taktbit am Beginn jeder Bitzelle

**Bild 2.5**
**Aufzeichnung mit doppelter Schreibdichte**



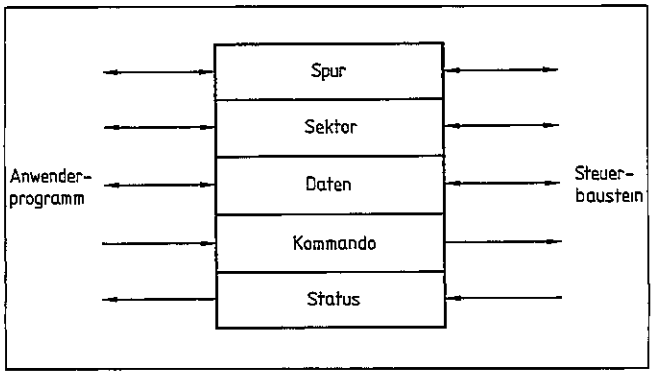

Regeln:
1) Schreibe Datenbit in die Mitte der Bitzelle, wenn „1" vorliegt
2) Schreibe Taktbit am Beginn einer Bitzelle, wenn:
   A) kein Datenbit in der letzten Bitzelle geschrieben wurde
   **und**
   B) kein Datenbit in der laufenden Bitzelle geschrieben wird

# 3. Beschreibung der Steuerbausteine SAB 179X

Die Informationen in diesem Kapitel sind als Erganzung zu den Datenblattern SAB 179X und anderen Applikationen gedacht (siehe Literaturhinweise)

Der Registersatz des SAB 179X unterstutzt ein sehr einfaches Interface zwischen dem Anwenderprogramm und dem Steuerbaustein (siehe Bild 3 1)

**Bild 3.1**
**SAB 179X Register**




Auf jedes Register kann zu jeder Zeit zugegriffen werden, da jedes eine eigene Adresse hat. Da das Kommandoregister ein „Nur-Schreib-Register" ist, und auf das Statusregister nur lesend zugegriffen werden kann, ist ihnen eine gemeinsame Adresse zugeordnet Auf die anderen Register kann man sowohl lesend als auch schreibend zugreifen. Die internen Adressen der einzelnen Register zeigt Bild 3 2.

**Bild 3.2**
**SAB 179X Registerauswahl**

| PIN 3 $\overline{CS}$ | PIN 6 A1 | PIN 5 A0 | PIN 4 $\overline{RE} = 0$ | PIN 2 $\overline{WE} = 0$ |
|---|---|---|---|---|
| 0 | 0 | 0 | STATUSREG | KOMMANDOREG |
| 0 | 0 | 1 | SPURREG | SPURREG |
| 0 | 1 | 0 | SEKTORREG | SEKTORREG |
| 0 | 1 | 1 | DATENREG | DATENREG. |
| 1 | X | X | TRISTATE | TRISTATE |

Zum Auslosen einer Funktion muß der Anwender die notwendigen Parameter in das Spur-, Sektor- und/oder Datenregister einschreiben Danach kann er den gewunschten Befehl uber das Kommandoregister ubergeben Nach Abschluß der Befehlsausfuhrung wird das BUSY-Bit im Statusregister zuruckgesetzt, und die INTRQ-Leitung auf „High"-Potential gelegt Dadurch hat der Anwender die Moglichkeit, zwischen Polling- und Interruptbetrieb zu wahlen Ahnlich hat man bei der Übergabe der Schreib-/Lesedaten die Auswahl zwischen zwei Betriebsarten

- Polling uber das Statusbit DRQ und
- Betrieb des SAB 179X zusammen mit einem DMA-Steuerbaustein uber den Anschluß DRQ

In beiden Fallen werden die Daten vom bzw zum SAB 179X uber das Datenregister ubergeben Im Zustandsregister werden genaue Informationen uber den Ablauf des letzten Kommandos angeboten. Einzelheiten dazu kann man dem Datenblatt entnehmen

Im Spurregister (track register) steht jeweils die Nummer der Spur, auf der der Schreib-/Lesekopf gerade steht Nach einem RESTORE-Kommando wird es auf 0 zuruckgesetzt Bei jedem STEP-Kommando (sofern der Parameter u = 1 ist) und SEEK-Kommando wird es auf den neuesten Stand gebracht

Das Sektorregister muß vom Benutzer geladen werden, um dem Steuerbaustein mitzuteilen, welcher Sektor gelesen oder beschrieben werden soll Bei einem Mehrsektor-Schreib-/Lese-Kommando (multiple read/write command) wird das Sektorregister vom Steuerbaustein auf dem neuesten Stand gehalten. Es gibt dann den jeweils nachsten zu beschreibenden Sektor an.

Das Datenregister ist normalerweise fur die Ubergabe von Datenbytes gedacht Bei einem SEEK-Kommando jedoch wird die Spurnummer, auf die positioniert werden soll, im Datenregister ubergeben. Der Steuerbaustein vergleicht Spur- und Datenregister. Der Kopf wird solange spurweise in die entsprechende Richtung bewegt, bis beide Inhalte ubereinstimmen

Bild 3 3 zeigt eine Ubersicht der einzelnen Kommandos und die Bedeutung der einzelnen Bits Die Kommandos werden in vier Gruppen eingeteilt Kommandos vom Typ I sind Befehle um den Kopf zu positionieren. In Gruppe II sind die Befehle zum Lesen und Schreiben von Sektoren zusammengefaßt Die Kommandos in Gruppe III dienen zum Schreiben/Lesen ganzer Spuren Das Schreiben einer Spur wird zum Formatieren einer Diskette benotigt Das Lesen einer Spur ist sehr nutzlich fur Diagnosezwecke und zum Retten von Daten einer defekten Diskette.

Das Kommando vom Typ IV ist der Abbruchbefehl (Force Interrupt Command). Es wird im allgemeinen benutzt, um ein Mehrsektor-Schreib-/Lese-Kommando abzubrechen, nachdem die gewunschte Anzahl von Sektoren bearbeitet ist

Nachdem ein Kommando an den Baustein ubergeben wurde, kann das Statusregister gelesen werden. Es enthalt Information uber den Ablauf des Kommandos Der Inhalt des Statusregisters ist allerdings erst einige Zeit nach der Kommandoubergabe gultig (siehe Bild 3 4) Falls das Statusregister zu fruh gelesen wird, fuhrt dies zu fehlerhafter Bearbeitung des Kommandos. Bei Benutzung des INTRQ-Anschlusses sollte das Statusregister erst nach der Auslosung eines Interrupts gelesen werden. Das Lesen des Statusregisters setzt namlich die Unterbrechungsanforderung zuruck

Vor Ubergabe eines Kommandos sollte sichergestellt werden, daß das Laufwerk betriebsbereit ist Manche Kommandos, wie zum Beispiel „RESTORE mit Uberprufung", konnen bei nicht betriebsbereitem Laufwerk zu einem Zustand fuhren, der nur durch Rucksetzen verlassen werden kann

**Bild 3.3**
**SAB 179X Kommandos**

### Zusammenfassung

| Kommandos für SAB 1791, SAB 1793 | | Bits | | | | | | | | Kommandos für SAB 1795, SAB 1797 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | Bits | | | | | | | |
| Type | Command | 7 | 6 | 5 | 4 | 3 | 2 | 1 | Ø | 7 | 6 | 5 | 4 | 3 | 2 | 1 | Ø |
| I | Restore | Ø | Ø | Ø | Ø | h | V | $r_1$ | $r_Ø$ | Ø | Ø | Ø | Ø | h | V | $r_1$ | $r_Ø$ |
| I | Seek | Ø | Ø | Ø | 1 | h | V | $r_1$ | $r_Ø$ | Ø | Ø | Ø | 1 | h | V | $r_1$ | $r_Ø$ |
| I | Step | Ø | Ø | 1 | T | h | V | $r_1$ | $r_Ø$ | Ø | Ø | 1 | T | h | V | $r_1$ | $r_Ø$ |
| I | Step-in | Ø | 1 | Ø | T | h | V | $r_1$ | $r_Ø$ | Ø | 1 | Ø | T | h | V | $r_1$ | $r_Ø$ |
| I | Step-out | Ø | 1 | 1 | T | h | V | $r_1$ | $r_Ø$ | Ø | 1 | 1 | T | h | V | $r_1$ | $r_Ø$ |
| II | Read Sector | 1 | Ø | Ø | m | S | E | C | Ø | 1 | Ø | Ø | m | L | E | U | Ø |
| II | Write Sector | 1 | Ø | 1 | m | S | E | C | $a_Ø$ | 1 | Ø | 1 | m | L | E | U | $a_Ø$ |
| III | Read Address | 1 | 1 | Ø | Ø | Ø | E | Ø | Ø | 1 | 1 | Ø | Ø | Ø | E | U | Ø |
| III | Read Track | 1 | 1 | 1 | Ø | Ø | E | Ø | Ø | 1 | 1 | 1 | Ø | Ø | E | U | Ø |
| III | Write Track | 1 | 1 | 1 | 1 | Ø | E | Ø | Ø | 1 | 1 | 1 | 1 | Ø | E | U | Ø |
| IV | Force Interrupt | 1 | 1 | Ø | 1 | $I_3$ | $I_2$ | $I_1$ | $I_Ø$ | 1 | 1 | Ø | 1 | $I_3$ | $I_2$ | $I_1$ | $I_Ø$ |

### Zusammenfassung der Kommandobits

| Kommandotyp | Bitnummer | | | Beschreibung |
|---|---|---|---|---|
| I | Ø, 1 | $r_1$ $r_Ø$ | = Stepping Motor Rate | |
| I | 2 | V | = Track Number Verify Flag | V = Ø, No verify<br>V = 1, Verify on destination track |
| I | 3 | h | = Head Load Flag | h = Ø, Load head at beginning<br>h = 1, Unloaded heat at beginning |
| I | 4 | T | = Track Update Flag | T = Ø, No update<br>T = 1, Update track register |
| II & III | Ø | $a_Ø$ | = Data Address Mark | $a_Ø$ = Ø, FB (DAM)<br>$a_Ø$ = 1, F8 (deleted DAM) |
| II | 1 | C | = Side Compare Flag | C = Ø, Disable side compare<br>C = 1, Enable side compare |
| II & III | 1 | U | = Update SSO | U = Ø, Update SSO to Ø<br>U = 1, Update SSO to 1 |
| II & III | 2 | E | = 15 ms Delay | E = Ø, No 15 ms delay<br>E = 1, 15 ms delay |
| II | 3 | S | = Side Compare Flag | S = Ø, Compare for side Ø<br>S = 1, Compare for side 1 |
| II | 3 | L | = Sector Length Flag | |
| | | L | = 1 (implicit) for SAB 1791/3 | |

| | LBS's Sector Length in ID Field ØØ | Ø1 | 10 | 11 |
|---|---|---|---|---|
| L = Ø | 256 | 512 | 1024 | 128 |
| L = 1 | 128 | 256 | 512 | 1024 |

| | | | | |
|---|---|---|---|---|
| II | 4 | m | = Multiple Record Flag | m = Ø, Single record<br>m = 1, Multiple records |
| IV | Ø, 3 | $I_x$ | = Interrupt Condition Flags | |
| | | $I_Ø$ | = 1 Not Ready To Ready Transition | |
| | | $I_1$ | = 1 Ready To Not Ready Transition | |
| | | $I_2$ | = 1 Index Pulse | |
| | | $I_3$ | = 1 Immediate Interrupt, Requires A Reset | |
| | | $I_3$-$I_1$ | = Ø Terminate With No Interrupt (INTRO) | |

**Bild 3.4**
**Notwendige Verzögerungszeiten vor nächstem Lesen des Statusregisters**

| Ausgeführte Funktion | Nächste Funktion | Notwendige Verzögerung FM | MFM |
|---|---|---|---|
| Schreiben in Kommandoregister | Lesen des Busybits (Statusbit Ø) | 12 µs | 6 µs |
| Schreiben in Kommandoregister | Lesen Statusbits 1–7 | 28 µs | 14 µs |
| Schreiben in jedes Register | Lesen eines anderen Registers | Ø | Ø |

# 4. Zeitverhalten und Kompatibilität mit verschiedenen Prozessorfamilien

Bild 4 1 zeigt das dynamische Verhalten der MC-Busschnittstelle des SAB 179X Beim Anschluß an die SAB-Standardprozessoren werden folgende Wartezyklen benotigt

| Prozessor | Wartezyklen |
|---|---|
| SAB 8051 | Ø |
| SAB 8085A | Ø |
| SAB 8085A-2 | 1 |
| SAB 8086/88 | 1 |
| SAB 8086/88-2 | 2 |

Ein anderer zeitkritischer Parameter ist die große vom SAB 179X geforderte Datenubertragungsrate Durch sie wird bestimmt, in welcher Zeit eine Ubertragungsanforderung vom Prozessor bearbeitet werden muß. In Bild 4.2 sind diese Zeiten zusammengestellt Daraus kann man ersehen, daß der Prozessor bzw eine DMA-Steuerung bei einer 8-Zoll-Floppy mit doppelter Schreibdichte einen Datentransfer im Extremfall innerhalb von 11,5 µs durchfuhren muß. Ein Beispiel für die Bearbeitung der Ubertragung eines Datenblockes zeigt das Flußdiagramm in

**Bild 4 2**
**Nominelle und „Worst Case"-Bedienzeiten**

| Disketten-größe | Schreib-dichte | Nominelle Uber-tragungs-rate | Worst Case-Bedienzeiten beim SAB 179X | |
|---|---|---|---|---|
| | | | LESEN | SCHREIBEN |
| 5¼" | Einfach | 64 µs | 55,0 µs | 47,0 µs |
| 5¼" | Doppelt | 32 µs | 27,5 µs | 23,5 µs |
| 8" | Einfach | 32 µs | 27,5 µs | 23,5 µs |
| 8" | Doppelt | 16 µs | 13,5 µs | 11,5 µs |

Bild 4 3 Genauere Untersuchungen zeigen, daß beim Anschluß einer 8 Zoll Double Density Floppy Disk auf jeden Fall eine DMA-Steuerung notwendig ist. Weder bei Polling- noch bei Interruptbetrieb kann man die Bearbeitungszeit garantieren

Fur SAB 8085A- und SAB 8088-Systeme ist also ein SAB 8237A und fur SAB 8086-(bzw auch SAB 8088-)Systeme ein SAB 8089 notwendig

**Bild 4.1**
**SAB 179X Schaltzeiten der Busschnittstelle**

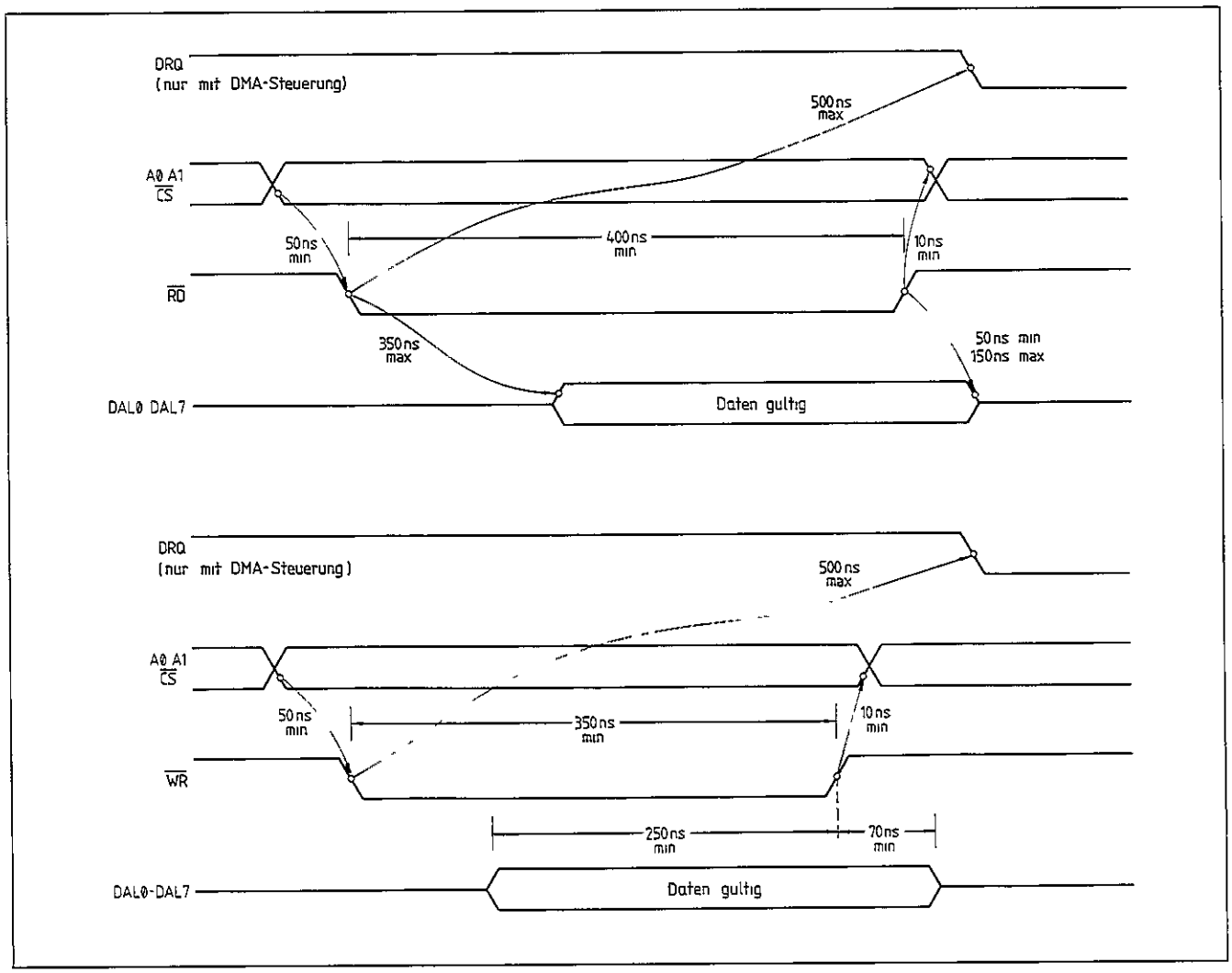
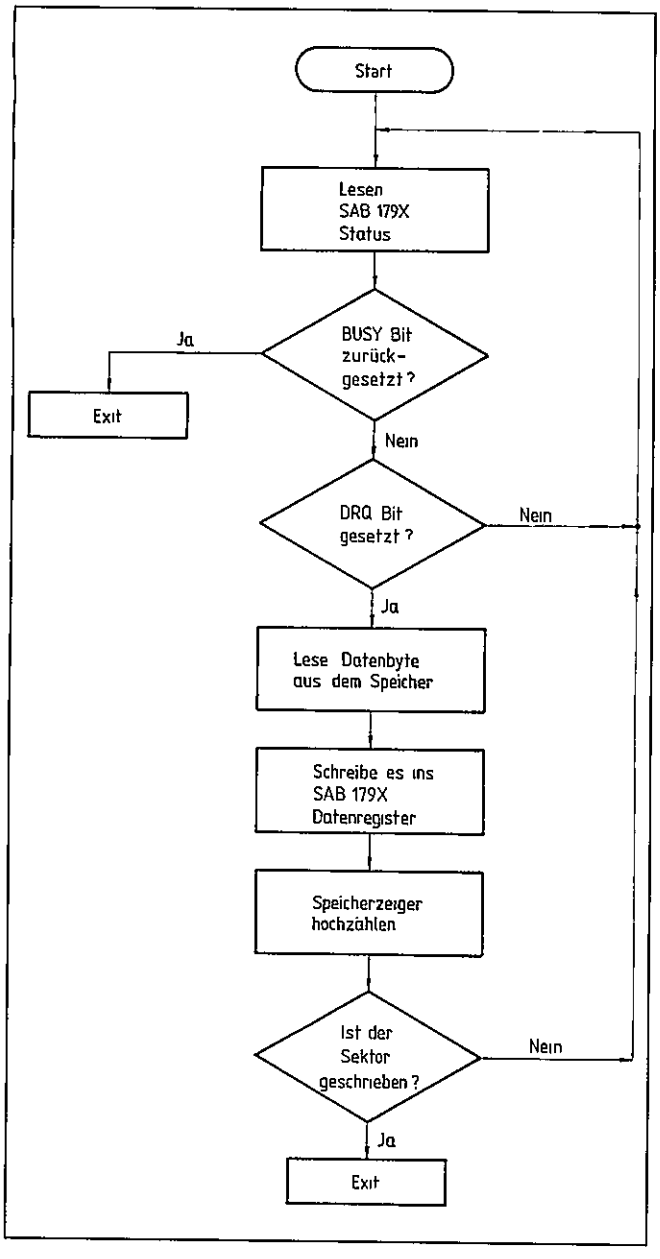


**Bild 4.3**
**Schreiben eines Sektors mit Polling beim SAB 179X**




Die Verwendung einer DMA-Steuerung hat zusätzlich weitere Vorteile. Während der Datenübertragung von/zu der Floppy Disk ist der Prozessor frei für andere Aufgaben. Dies sorgt für eine gute Systemverfügbarkeit. Bei 8-Zoll-Disketten mit einfacher Schreibdichte und bei Minifloppys bietet auch das Pollingverfahren mit Hilfe des DRQ-Statusbits ausreichend kurze Bearbeitungszeiten. Eine ausführliche Beschreibung der DMA-Steuerung finden Sie in Kapitel 8.3.

# 5. Beschreibung der Applikationshardware

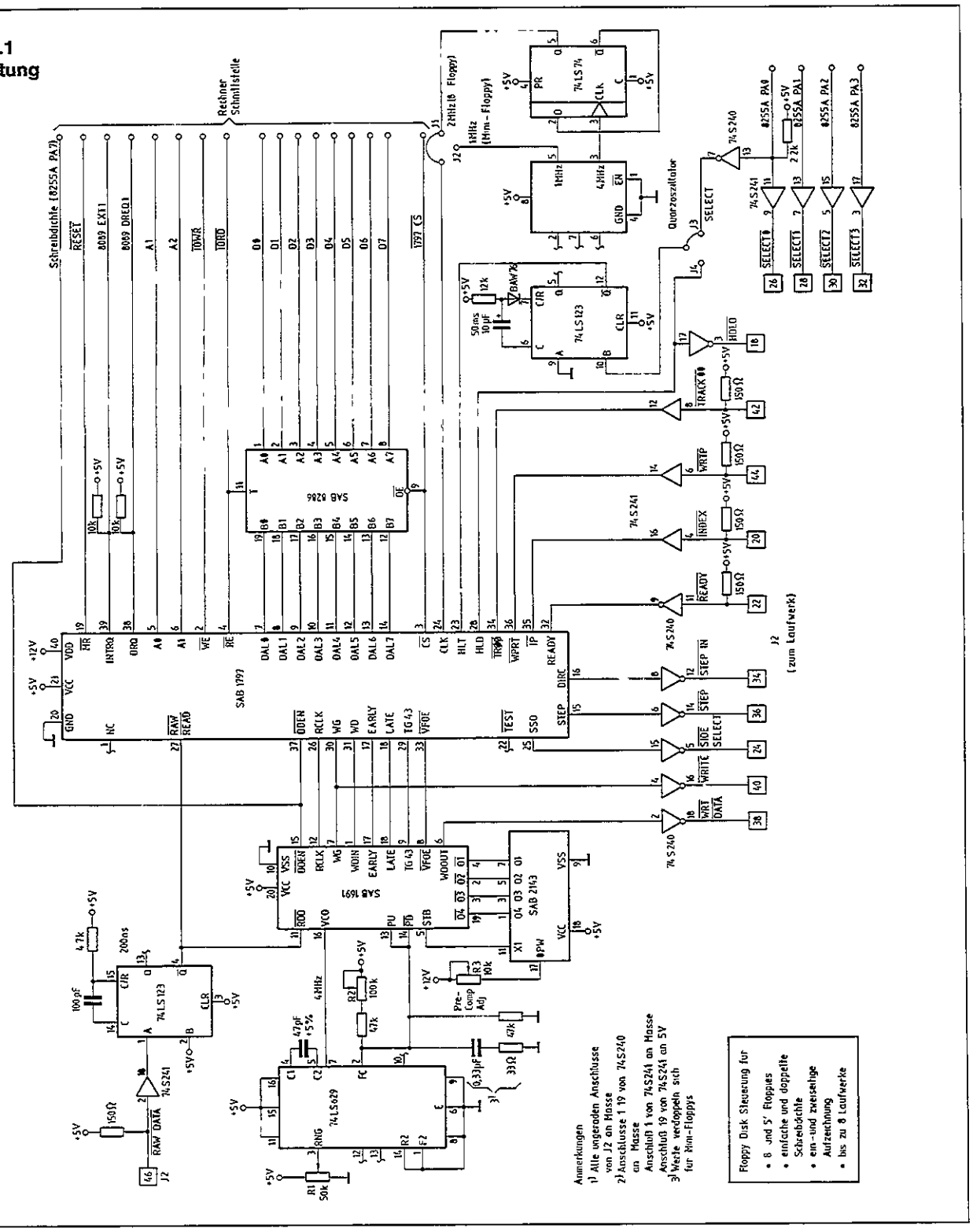
Bild 5 1 zeigt die Schaltung der Hardware, die fur die Applikation verwendet wurde Sie ist mit dem Floppy-Disk-Steuerbaustein SAB 1797 aufgebaut Dieselbe Schaltung gestattet aber auch die Verwendung der Bausteine SAB 1791/92/93. Bei Verwendung der Bausteine SAB 1791/95 (mit invertiertem Datenbus) muß der bidirektionale Bustreiber SAB 8286 durch den SAB 8287 ersetzt werden

Die Applikationshardware kann bis zu 8 Laufwerke (8 oder 5¼ Zoll) in jeder beliebigen Kombination von

- einfacher oder doppelter Schreibdichte und
- mit einseitiger oder zweiseitiger Aufzeichnung

bedienen.

**Bild 5.1**
**Schaltung**



Die Floppy-Disk-Steuerung wurde an ein SAB 8086-Mikrocomputersystem angeschlossen. Die Ansteuerung erfolgte über den Ein-/Ausgabeprozessor SAB 8089, die Interruptsteuerung SAB 8259A und den Parallel-Ein-/Ausgabebaustein SAB 8255A und einige TTL-Bausteine. (Der SAB 8089 wird nur bei 8-Zoll-Disketten mit doppelter Schreibdichte benötigt.)

Die Rechnerschnittstelle besteht aus den Datenleitungen, den Adreßleitungen A1 und A2 zur Auswahl der bausteininternen Register, dem Baustein-Auswahlsignal (C3) und einigen Steuersignalen.

Zusätzlich dazu werden noch die Signale SELECT Ø-3 zur Auswahl der Laufwerke und ein Signal zur Unterscheidung zwischen einfacher und doppelter Schreibdichte benötigt. Diese werden über den SAB 8255A (Kanal A) erzeugt.

Ein Quarzoszillator mit nachgeschaltetem Teiler-FF liefert den 2 MHz-Takt für 8-Zoll-Floppys bzw. den 1 MHz-Takt für 5¼-Zoll-Floppys. Die Auswahl zwischen den beiden Takten erfolgt über Steckbrücken.

Das 50 ms-Monoflop zur Erzeugung des Signals HLT kann entweder mit dem SELECT-Signal oder mit dem Signal HLD getriggert werden. Die Auswahl erfolgt ebenfalls mit Steckbrücken.

In vielen Systemen (wie z. B. im Mikrocomputer-Entwicklungssystem SME) erfolgt das Absenken des Kopfes direkt bei Selektierung des Laufwerkes. Für solche Systeme ist das Signal HLD (Head Load) ohne Bedeutung.

Die Dekoder für die Laufwerkauswahl in den einzelnen Laufwerken werden freigegeben, wenn SELECT Ø angelegt wird. An SELECT 1–3 muß die Adresse des angesprochenen Laufwerks angelegt werden (siehe Bild 5.2). Wenn SELECT Ø nicht aktiviert ist, wird kein Laufwerk angesprochen.

**Bild 5.2**
**Laufwerksauswahl**

| $\overline{\text{SELECT 1}}$ | $\overline{\text{SELECT 2}}$ | $\overline{\text{SELECT 3}}$ | Ausgewähltes Laufwerk |
|---|---|---|---|
| 1 | 1 | 1 | Ø |
| Ø | 1 | 1 | 1 |
| 1 | Ø | 1 | 2 |
| Ø | 1 | 1 | 3 |
| 1 | 1 | Ø | 4 |
| Ø | 1 | Ø | 5 |
| 1 | Ø | Ø | Ø |
| Ø | Ø | Ø | 7 |

Um die Leitungen zu den Laufwerken zu treiben, werden Bausteine vom Typ 74S240 (bzw. 74S241 für nichtinvertierte Signale) wegen ihrer hohen Treiberleistung verwendet (IOL = 64 mA und IOH = −15 mA). Dieselben Bausteine eignen sich durch ihre Schmitt-Trigger-Eingänge auch sehr gut zum Empfang von Signalen von den Laufwerken. Bei diesen Signalen wird allerdings ein Pull-Up-Widerstand von 150 Ohm benötigt.

Diese Beschaltung ist jedoch nicht für Laufwerke aller Hersteller geeignet. Sie ist für Siemens- und Shugartlaufwerke erprobt; bei anderen Herstellern können unterschiedliche Beschaltungen notwendig sein. Häufig findet man auch Systeme, die mit Bausteinen vom LS-Typ aufgebaut sind und Abschlußwiderstände von 220 Ohm nach +5 V und 330 Ohm nach Masse aufweisen. Im Anhang A sind Einzelheiten über die Interfacesignale von Standard-Floppy-Disk-Laufwerken zu finden.

Aus drei Bausteinen besteht die Schaltung für die Schreibvorkompensation (write precompensation) und Datentrennung. Die Grundlagen für die Datentrennung sind in Anhang B zu finden, die für Schreibvorkompensation in Anhang E. Der SAB 1691 ist der Kernbaustein. Er wird vom Baustein SAB 2143 mit einem 4-Phasen-Takt versorgt, der aber nur zur Schreibvorkompensation benötigt wird. Wenn man also nur Disketten mit einfacher Schreibdichte verwenden will, ist dieser Baustein nicht notwendig. Gesteuert durch die Signale EARLY und LATE vom SAB 2797 wird der Schreibimpuls mit unterschiedlichen Verzögerungen geschrieben. Der 74S629 (oder auch 74LS124) ist ein VCO-Baustein (voltage controlled oscillator). Bei richtiger Einstellung des Widerstandes R1 erzeugt er einen 4 MHz-1:1-Takt. Gesteuert wird er durch die Anschlüsse PD (Pump Down) und PU (Pump Up). Sie sind normalerweise im Tristate-Zustand, so daß die Spannung am Pin 2 vom 74S629 (bei angegebener Beschaltung) 1,4 V beträgt. Muß die Normalfrequenz von 4 MHz erhöht werden, legt der SAB 1691 das Signal PU an. Dadurch wird die Spannung am Pin 2 des 74S629 erhöht, wodurch auch die Frequenz ansteigt. Analog senkt das Anlegen des Signals PD die Frequenz. Mit Hilfe dieser Signale kann der SAB 1691 die Normalfrequenz von 4 MHz entsprechend der Frequenz der ankommenden RAW DATA-Impulse regeln. Gesteuert wird dies durch einen digitalen Phasenkomparator, der die Phase der Datenimpulse mit der des 4 MHz-Taktes vergleicht. Durch Teiler wird aus dem VCO-Takt der Lesetakt für den SAB 1797 (RCLK = read clock) erzeugt. Bei MFM beträgt der Teilungsfaktor 8, bei FM 16 (siehe auch Anhang B).

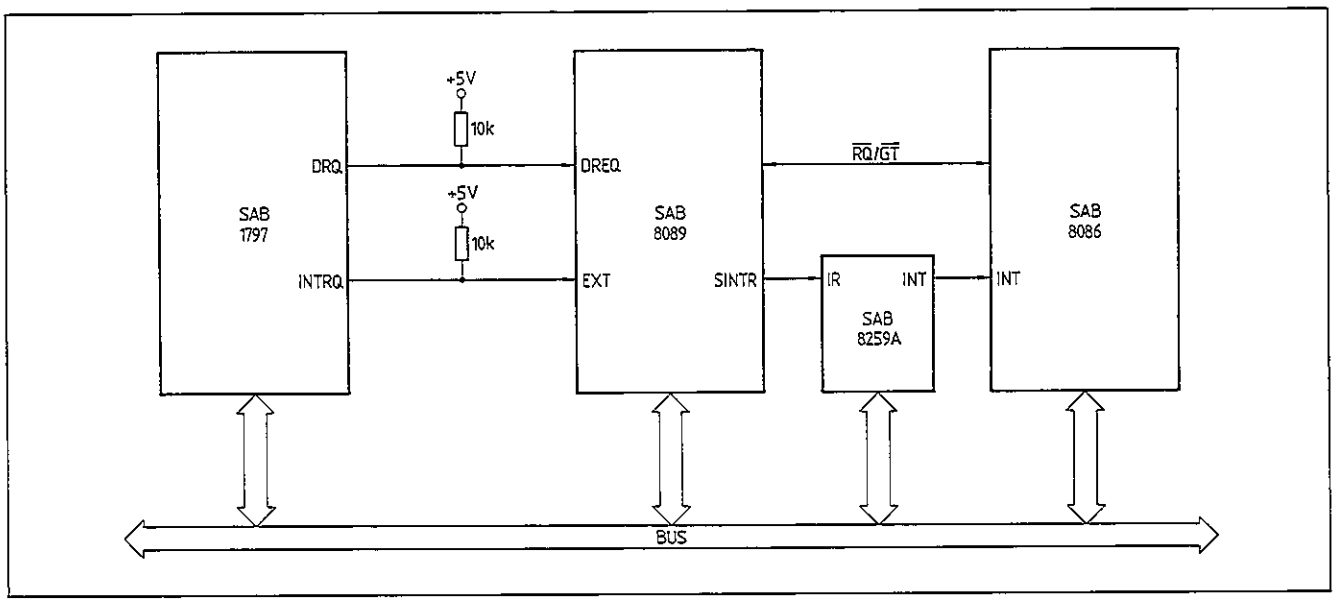
Die RAW DATA-Impulse werden auf eine konstante Breite von 200 ns gebracht (Minimum = 100 ns). Dies ist unbedingt notwendig, um ein einwandfreies Arbeiten des SAB 1691 zu gewährleisten.

Der bidirektionale Treiber SAB 8286 wird nur bei einem stark belasteten Bus benötigt. Er ist nicht notwendig, wenn die Treiberleistung des SAB 1797 ausreicht (IOL = 1,6 mA, IOH = −100 µA).

Zu beachten sind die Pull-Up-Widerstände für die INTRQ- und DRQ-Leitungen.

Wird der SAB 8089 für den DMA-Betrieb verwendet, zeigt Bild 5.3 den Anschluß der INTRQ- und DRQ-Leitungen. Wird der SAB 8089 nicht verwendet (wie beim hier gezeigten Software-Treiber für einfache Schreibdichte), können die Leitungen offenbleiben. Der Anschluß EXT (External Terminate Input) des SAB 8089 wird verwendet, um das Signal INTRQ zu bedienen. In den meisten Fällen zeigt es das Ende eines Kommandos an und kann verwendet werden, um die Datenübertragung des SAB 8089 zu beenden. Dies hat außerdem den Vorteil, daß nur ein Interrupt des SAB 8089 für die Floppy-Disk-Steuerung benötigt wird.

**Bild 5.3**
**SAB 1797 mit SAB 8089**



# 6. Abgleich und Test der Hardware

Bevor man versucht, ein Laufwerk anzusteuern, müssen erst einige Abgleicharbeiten durchgeführt werden. Anschließend ist ein einwandfreier Ablauf gewährleistet.

Zunächst müssen die Zeiten der beiden Monoflops eingestellt werden. Das eine erzeugt das HLT-Signal, das eine Breite von 50 ms haben muß. Das zweite begrenzt die RAW READ-Impulse auf 200 ns. Die einfachste Möglichkeit ist es, einen externen Takt zuzuführen und die Ausgangsimpulse mit Hilfe eines Oszilloskops einzustellen. Mit den in der Schaltung angegebenen Werten für die Widerstände und Kapazitäten sollte das richtige Timing erreicht werden.

Der dritte Abgleich wird an R2 durchgeführt. R2 muß so eingestellt werden, daß die Ruhespannung am Anschluß 2 von 74LS629 bei 1,4 V liegt. Die Ruhespannung liegt vor, wenn an den Anschlüssen RDD, PU und PD keine Signale anliegen. Als nächstes wird der Widerstand R1 so eingestellt, daß der 74LS629 ein 4 MHz-Signal an Pin 7 liefert. Mit dieser Justierung von R1 und R2 sollte es keine Probleme mit der Datentrennung geben.

Die letzte Abgleichmaßnahme an R3 ist nur notwendig, wenn Schreibvorkompensation benötigt wird (bei 8-Zoll-Floppys mit doppelter Schreibdichte). Sie wird am einfachsten mit Hilfe eines „Schreibe-Spur"-Kommandos (write track) und aufgetrennter IP-Leitung durchgeführt. R3 muß so eingestellt werden, daß die Pulsbreite an Pin 4 des SAB 1691 ungefähr 150 ns beträgt (≈ Vorkompensationszeit).

Nachdem diese Einstellungsmaßnahmen durchgeführt wurden, kann das Laufwerk an die Steuerung angeschlossen werden.

Es empfiehlt sich jedoch, zuerst noch das Businterface zum SAB 1797 zu testen. Dazu kann ein Monitorprogramm, das auf der Zentraleinheit abläuft, ausreichend sein. Ein Testhilfsmittel, wie z.B. ein Emulations- und Testadapter (ETA = ICE = In Circuit Emulator), ist allerdings eine gute Hilfe. Versuchen Sie, die Sektor- und Spurregister des SAB 1797 zu beschreiben und wieder auszulesen. Die Werte müssen übereinstimmen. Stimmen sie nicht überein, müssen die Signale $\overline{CS}$, $\overline{RE}$, $\overline{WE}$, AØ und A1 überprüft werden. Falls ein Datenbustreiber (SAB 8286) vorhanden ist, sind seine Steuersignale ebenfalls zu kontrollieren. Es muß möglich sein, die Register des SAB 1797 fehlerfrei zu beschreiben und zu lesen, bevor man weitere Tests durchführt.

Die nächste Stufe ist der Test der Kommandos vom Typ I. Legen Sie die Auswahlsignale für das (ein) Laufwerk an, in dem sich eine Scratch-Diskette befindet. Überprüfen Sie, ob das READY-Signal abgegeben wird und ob Indeximpulse beim SAB 1797 ankommen. Mit RESTORE-, STEP- und STEPIN-Kommandos kann die Steuerung des Kopfes getestet werden. Funktioniert das Positionieren des Kopfes, sollten Sie dieselben Kommandos mit einer schon formatierten Diskette und gesetztem Verify-Flag ausprobieren. Wenn nach der Ausführung kein Fehler im Statusregister gemeldet wird, kann man annehmen, daß die Leseschaltung einschließlich der Datenseparation funktioniert. Falls ein CRC- oder SEEK-Fehler aufgetreten ist, führen Sie ein RESTORE-Kommando mit gesetztem Verify-Flag durch. Dieses Kommando muß fehlerfrei ausgeführt werden, bevor man weitere Tests durchführt.

Nachdem Sie nun Kommandos vom Typ I ausführen können, laden Sie das Sektorregister und lassen ein „Read Sector"-Kommando ausführen. Es müßte jetzt problemlos möglich sein.

Der letzte Test, bevor man auf den kompletten Softwaretreiber übergeht, ist das Schreiben eines Sektors und das Zurücklesen. Falls dies nicht fehlerfrei geht, überprüfen Sie bei Disketten mit doppelter Schreibdichte die Vorkompensation.

Die Erfahrung hat gezeigt, daß das Interface zum Floppy-Disk-Laufwerk funktionsfähig ist, wenn

- die fünf Abgleichmaßnahmen sauber durchgeführt wurden,
- das Businterface zum SAB 1797 und
- das Interface zum Laufwerk in Ordnung sind.

Die übrigen Teile der Steuerschaltung werfen normalerweise keine Probleme auf, und somit ist die Floppy-Disk-Steuerung betriebsbereit.

# 7. Beschreibung des Softwaretreibers

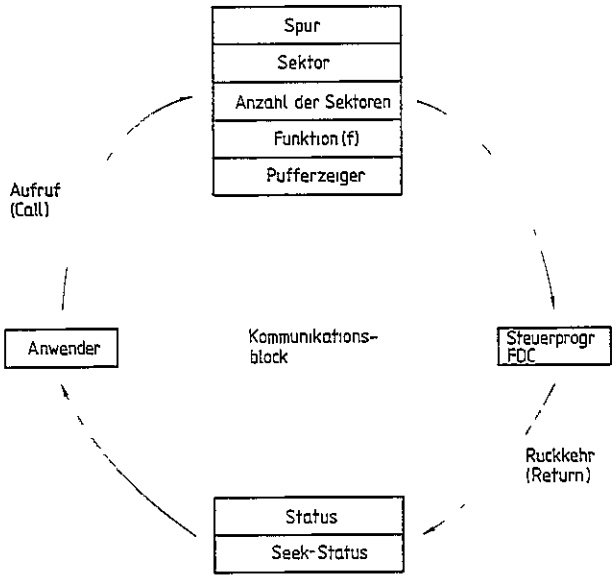
Der Softwaretreiber, der in diesem Kapitel beschrieben wird, basiert auf dem Prozessor SAB 8086 Um eine geringe Komplexität und damit ein leichteres Verständnis zu erreichen, wird hier kein Gebrauch von der DMA-Möglichkeit mit dem Prozessor SAB 8089 gemacht Der Softwaretreiber ist für ein System mit einem einzigen Laufwerk konzipiert Dadurch kann das Pollingverfahren angewendet werden, und es ist keine Interruptverarbeitung notwendig Durch das Fehlen von DMA können jedoch keine Laufwerke mit doppelter Schreibdichte unterstützt werden Der SAB 8086 kann bei einem Systemtakt von 5 MHz die notwendigen Statusabfragen (Polling) und Datentransfers nicht schnell genug durchführen

Aus diesem Grunde wurde ein zweiter Softwaretreiber beschrieben, der in Verbindung mit einem SAB 8089 doppelte Schreibdichte sowie mehrere Laufwerke unterstützt Diese Programme sind im Anhang F zu finden.

Mit dem vorliegenden Softwaretreiber soll hauptsächlich gezeigt werden, welche Programme unbedingt notwendig sind, um den SAB 1797 betreiben zu können.

Das Anwenderprogramm steht mit dem Softwaretreiber (FDC) über einen Kommunikationsblock in Verbindung Dieser 10 Byte lange Block muß vom Anwender als STRUCTURE vereinbart und PUBLIC deklariert werden Das Format des Blockes ist in Bild 7.1 gezeigt. Der Treiber FDC stellt dem Anwender 9 Funktionen zur Verfügung, die ebenfalls in Bild 7 1 aufgelistet sind Jede Funktion (f) benötigt einige Parameter, die vor dem Aufruf von FDC vom Anwender einzutragen sind Nachdem das Kommando ausgeführt wurde, übergibt FDC zwei Statusbytes an das Anwenderprogramm. Der Inhalt entspricht den zwei Statusregistern des SAB 1797 Das eine gehört zum SEEK-Kommando (bzw. einem anderen Typ I Kommando), das andere zu den Typ II bzw Typ III Kommandos Nach einem RESTORE-Befehl wird der neue Wert des SAB 1797 Registers in das Spurfeld des Kommunikationsblockes kopiert Bei allen übrigen Funktionen werden nur die Statusbytes verändert

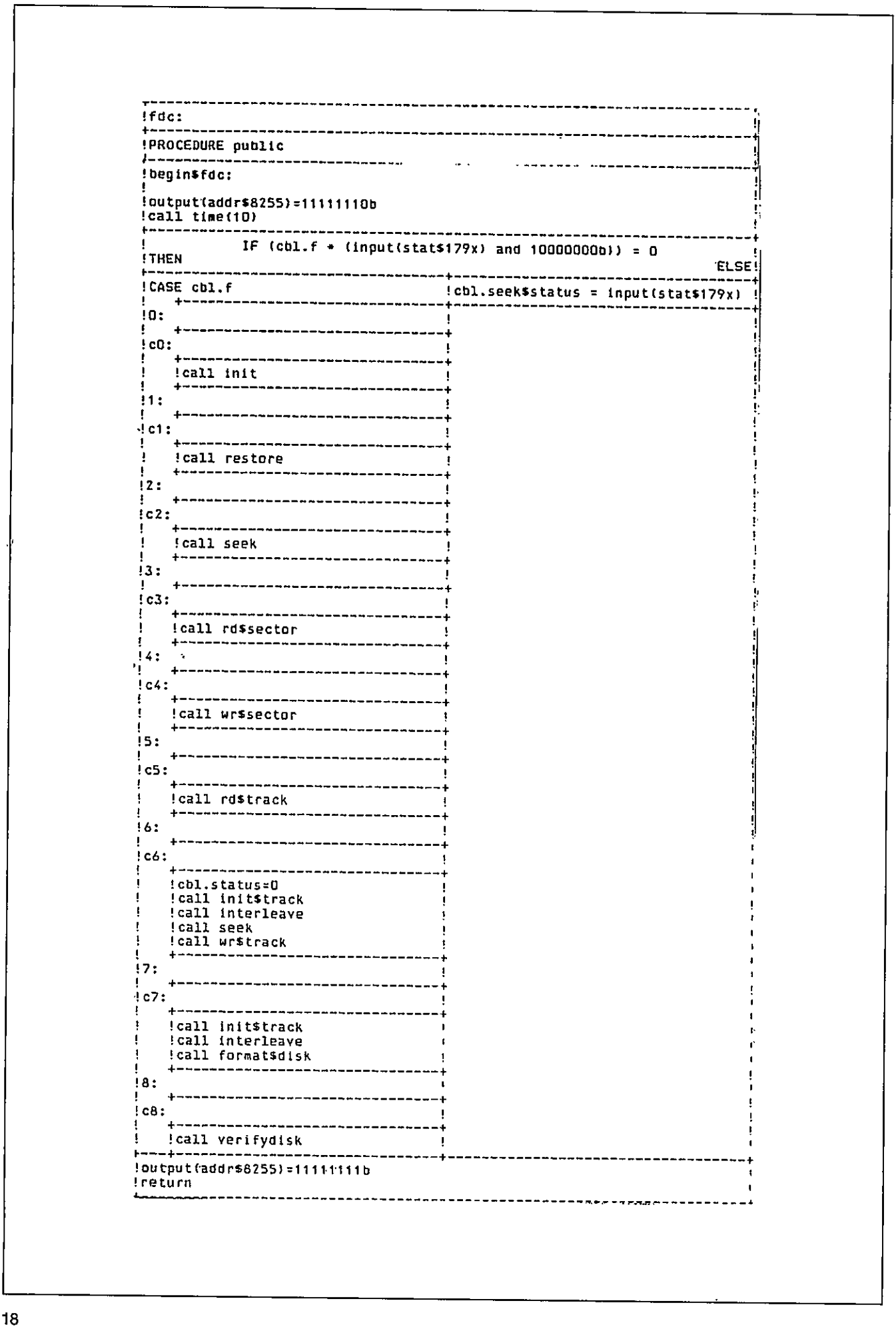
**Bild 7.1**
**Kommunikation zwischen Softwaretreiber FDC, Steuerung und Anwender**




Der Anwender muß vor dem Aufruf von FDC die notwendigen Parameter einsetzen. Der FDC liefert die Statusinformationen, die über die (nicht) erfolgreiche Ausführung des Kommandos Auskunft geben. Bei Funktionen, die mit einem Positionieren des Kopfes verbunden sind, sind beide Statusbytes gültig, sonst nur das obere

| f | Funktion | Beschreibung | Parameter |
|---|---|---|---|
| Ø | Initialisiere | Initialisierung des Programms | — |
| 1 | Restore | Kopf auf Spur Ø positionieren | — |
| 2 | Positioniere | Positionieren auf Spur cbl track | Spur |
| 3 | Lese Sektor(en) | Positionieren und Lesen von Sektor(en) | Spur, Sektor, Anzahl, Pufferzeiger |
| 4 | Schreibe Sektor(en) | Positionieren und Schreiben von Sektor(en) | Spur, Sektor, Anzahl, Pufferzeiger |
| 5 | Lese Spur | Positionieren und Lesen der ganzen Spur | Spur, Pufferzeiger |
| 6 | Schreibe Spur | Positionieren und Schreiben (Formatieren) der Spur | Spur |
| 7 | Formatiere Diskette | Diskette formatieren (IBM 3740) | — |
| 8 | Überprüfe Diskette | Diskette auf CRC-Fehler überprüfen | — |

**Bild 7.2**
**Oberste Ebene des Steuerprogramms**



Die Statusinformation des SAB 1797 wird unverandert an das Anwenderprogramm weitergeleitet Eine eventuell notwendige Fehlerbehandlung bleibt also dem Anwender uberlassen Der erste Aufruf von FDC muß mit der Funktion f = Ø erfolgen, um die FDC-Parameter zu initialisieren Als zweites Kommando sollte man den RESTORE-Befehl (f = 1) ausführen

Der große Teil des Softwaretreibers ist in PLM86 geschrieben, um das Verstandnis zu erleichtern Die Datenubertragungsroutinen mußten jedoch aus Geschwindigkeitsgrunden im Assembler (ASM86) geschrieben werden Aus den PLM-Programmen wurden mit Hilfe von STRUCT Strukturprogramme generiert Anhand dieser Ablaufdiagramme werden die einzelnen Programme erlautert Sie sind in den Bildern 7 2 bis 7 7 zu finden

In Bild 7.2 ist die oberste Programmebene dargestellt. Als erstes wird das Laufwerk Ø ausgewahlt, indem der Wert 1111111Ø auf dem Kanal A des SAB 8255A ausgegeben wird Nach einer Verzogerungszeit wird uberpruft, ob das Laufwerk bereit ist (Bit 7 des Statusregisters) Wenn nicht, wird der SAB 1797 Status in den Kommunikationsblock kopiert und zum Anwenderprogramm zuruckgesprungen. Auf diese Weise wird sichergestellt, daß das Laufwerk bei der Ausfuhrung eines Kommandos betriebsbereit ist Nicht notwendig ist dies bei der Initialisierung von FDC (f = Ø)

Ist das Laufwerk in Ordnung, wird das zur gewunschten Funktion gehorige Programm angesprungen. Dies geschieht durch einen DO CASE Block mit f (cbl f in der PLM-Nomenklatur) als Auswahlparameter Bild 7 2 zeigt die neun moglichen Falle fur f (sie sind in Bild 7 1 beschrieben)

**Bild 7.3**
**Restore (Positionieren auf Spur Ø) und Positionieren auf Spur**

```
restore:
PROCEDURE

output(comm$179x)=res$comm
call time(1)
WHILE (input(stat$179x) and 1) <> 0
cbl.seek$status=input(stat$179x)
cbl.track=input(track$179x)
return
```

```
seek
PROCEDURE

output(data$179x)=cbl.track
output(comm$179x)=seek$comm
call time(1)
WHILE (input(stat$179x) and 1) <> 0
cbl.seek$status=input(stat$179x)
return
```

## 7.1 RESTORE- und SEEK-Funktionen

Die Falle f = 1 und f = 2 stehen fur die Typ I Kommandos RESTORE und SEEK In Bild 7 3 sind die zugehorigen Ablaufplane zu finden Nach der Ubergabe der Kommandos muß eine gewisse Zeit gewartet werden, bevor das Statusregister gelesen werden darf Danach wird das Busybit solange abgefragt, bis der Nicht-busy-Zustand gemeldet wird Bei der RESTORE-Funktion wird dann die Status- und Spurinformation auf den neuesten Stand gebracht und ins Anwenderprogramm zuruckgekehrt.

Beim SEEK-Kommando wird vor der Ausfuhrung des Kommandos erst das Spurregister mit der gewunschten Spurnummer geladen Sowohl das RESTORE- als auch das SEEK-Kommando werden mit gesetztem Verify-Flag ausgefuhrt Dadurch werden die angefahrenen Spuren automatisch uberpruft

## 7.2 Lesen und Schreiben von Sektoren

Dies sind die am haufigsten benotigten Funktionen Zu ihnen gehoren die Nummern f = 3 bzw f = 4 Alle Typ II Kommandos sind als Mehrsektorenbefehle ausgefuhrt (Lesen und Schreiben mehrerer Sektoren ohne Unterbrechung). Im „Anzahl"-Feld des Kommunikationsblockes wird die Zahl der zu lesenden (schreibenden) Sektoren ubergeben. Begonnen wird mit dem Sektor, dessen Nummer im Sektorbyte ubergeben wird Man muß darauf achten, daß die Summe aus der Nummer des Anfangssektors plus der Anzahl nicht die Zahl der tatsachlich vorhandenen Sektoren uberschreitet Man erhalt sonst die Fehlermeldung „Record not found" (Datenblock nicht gefunden), da der SAB 1797 versucht, eine nicht vorhandene Sektornummer zu finden.

Bei beiden Funktionen wird zunachst eine SEEK-Operation auf die im Kommunikationsblock angegebene Spur durchgefuhrt. Danach wird der „Anzahl"-Parameter uberpruft (Ø < Anzahl ≤ Zahl der Sektoren). Danach wird das in ASM86 geschriebene Programm aufgerufen, das den eigentlichen Datentransfer durchfuhrt Die Routine benotigt zwei Parameter, die Anzahl der zu lesenden/schreibenden Bytes und die Adresse des Zeichenpuffers Diese Adresse wird im Kommunikationsblock ubergeben

Die Ubertragungsroutinen sind in ASM86 geschrieben, um die SAB 8086 Register optimal ausnutzen zu können. Dadurch wird die großtmogliche Ubertragungsgeschwindigkeit erreicht. Alle Übertragungsprogramme sind nach dem Ablaufplan von Bild 4 3 implementiert

Als Beispiel wollen wir die Schreibroutine betrachten (Seite 4 der Assemblerlistings) In den Programmzeilen 105 bis 109 werden die CPU-Register geladen. Dann wird ein „Schreibe-Sektor"-Kommando (mit gesetztem Mehrsektorflag) an den SAB 1797 ubergeben. Vor der ersten Abfrage des Statusregisters kommt wieder die Verzögerungszeit, die vom Steuerbaustein vorgeschrieben ist Das CX-Register wird mit der Anzahl von Bytes geladen, die gelesen/geschrieben werden sollen Sie errechnet sich aus der Anzahl von Sektoren mal die Anzahl von Bytes/Sektor Die Programmzeilen 114 bis 124 entsprechen den beiden Schleifen aus Bild 4 3 Zuerst wird uber das Busybit abgefragt, ob die Kommandobearbeitung schon beendet ist. Dies ware z.B bei einer schreibgeschutzten Diskette schon zu diesem Zeitpunkt der Fall. Danach wird das DRQ-(Data Request-)Bit abgefragt Wenn es gesetzt ist, benotigt der SAB 1797 das nachste Datenbyte Im Programm werden Befehle wie LODSB und LOOP verwendet, die mehrere Funktionen in einem Befehl vereinigen

LODSB holt ein Byte aus dem Speicher und inkrementiert gleichzeitig den Pointer. LOOP dekrementiert CX (Byte-Zahler) und springt nur, wenn CX ≠ Ø ist. Ist CX = Ø, wird der folgende Befehl abgearbeitet

**Bild 7.4**
**Lesen und Schreiben eines Sektors**

```
!rd$sector:
!PROCEDURE
!
!call seek
!output(sector$179x)=cbl.sector
!                              IF cbl.many = 0
!THEN                                                          ELSE!
!cbl.many = 1                                                  !    !
!                         IF cbl.many > no$sect                !    !
!THEN                                                          ELSE!
!cbl.many = no$sect                                            !    !
!call readsector(cbl.many*sector$length,cbl.buff$ptr)                !
!WHILE (input(stat$179x) and 1) <> 0                                 !
!cbl.status=input(stat$179x)                                         !
!return                                                              !

!wr$sector:
!PROCEDURE
!
!call seek
!output(sector$179x)=cbl.sector
!                              IF cbl.many = 0
!THEN                                                          ELSE!
!cbl.many = 1                                                  !    !
!                         IF cbl.many > no$sect                !    !
!THEN                                                          ELSE!
!cbl.many = no$sect                                            !    !
!call writesector(cbl.many*sector$length,cbl.buff$ptr)               !
!WHILE (input(stat$179x) and 1) <> 0                                 !
!cbl.status=input(stat$179x)                                         !
!return                                                              !
```

Wenn alle Bytes an den Steuerbaustein übergeben wurden, wird die Schleife verlassen und das Programm kommt zur Programmzeile 126. An dieser Stelle ist es sehr wichtig, eine Zeitverzögerung einzubauen. Sie ist aus folgendem Grund notwendig: Übergibt man das letzte Byte an den SAB 1797, ist er noch mit dem Schreiben des vorletzten Bytes auf die Diskette beschäftigt. Erteilt man nun einen Abbruchbefehl (Force Interrupt Command), wird die momentan ablaufende Aktion **sofort** abgebrochen. Dadurch würden das vorletzte und letzte Byte nicht richtig auf der Diskette abgespeichert werden. Das Lesen eines Sektors, bei dem der Abbruchbefehl zu früh übergeben wurde, ergibt einen CRC-Fehler.

Danach kommt der Programmablauf zurück in den PLM-Teil des Treibers. Hier wird wiederum das Busybit im Statusregister abgefragt, um sicherzustellen, daß das Kommando abgeschlossen wurde. Nun kann das Statusbyte im Kommunikationsblock auf den neuen Stand gebracht werden und die Kontrolle an das Anwenderprogramm zurückgegeben werden.

## 7.3 Lesen und Schreiben einer Spur

Bild 7.5 zeigt den Algorithmus der Kommandos zum Lesen und Schreiben einer ganzen Spur.

Die Funktion des Lesens einer ganzen Spur bietet die Möglichkeit, alle auf der Spur aufgezeichneten Bytes auszulesen. Diese beinhalten neben den Datenbytes auch die Lücken (Gaps), CRC-Bytes und ID-Feldinformation. Dies ist sehr hilfreich für Diagnosezwecke und besonders bei der „Reparatur" von Disketten, bei denen einige Bits umgekippt sind. Die „schlechte" Spur kann in den Speicher kopiert werden, die Spur neu formatiert werden (Schreibe-Spur-Kommando) und dann die korrigierten Sektoren wieder auf die Diskette geschrieben werden (Schreibe-Sektor-Kommandos). Die Ausführung des Spurlese-Kommandos ist recht einfach. Man versucht eine sehr große Zahl von Bytes von der Spur zu lesen (z. B. 1800H). Die Zahl selbst ist ohne Bedeutung, sie muß nur größer sein als die Zahl der Bytes auf der Spur. Wenn alle Bytes (z. B. 1460H) gelesen wurden, setzt der SAB 1797 das Busybit zurück und zeigt damit die Beendigung des Kommandos an.

**Bild 7.5**
**Lesen und Schreiben einer Spur**

```
rd$track
PROCEDURE

call seek
call readtrack(1800h,cbl.buff$ptr)
WHILE (input(stat$179x) and 1) <> 0
cbl.status=input(stat$179x)
return
```

```
wr$track:
PROCEDURE

track$form.track=cbl.track
call writetrack
WHILE (input(stat$179x) and 1) <> 0
cbl.status=input(stat$179x) or cbl.status
call verifytrack
WHILE (input(stat$179x) and 1) <> 0
cbl status=input(stat$179x) or cbl status
return
```

Die Funktion des Spurschreibens wird verwendet, um die Spur zu formatieren Bild 7 2 zeigt, daß sie aus mehreren Schritten besteht (f = 6) Das Programm Init$Track (Bild 7 7) beschreibt einen Puffer mit dem Format der Spur. Interleave ist ein Unterprogramm, das die Zuordnung zwischen logischer und physikalischer Sektorfolge festlegt. Wie Bild 7 7 zeigt, sind hier logische und physikalische Folge identisch Ein aufwendigeres Programm (siehe Anhang D) gestattet es, unterschiedliche Folgen zu definieren Dies dient dazu, die Zugriffszeit und damit auch die Ausfuhrungszeiten des Betriebssystems zu optimieren

Als nachste Operation folgt ein SEEK-Kommando, um den Kopf auf die gewunschte Spur zu positionieren Die Funktion „Schreibe Spur" benotigt zwei Assemblerprogramme Das eine beschreibt die Spur, wahrend das zweite dazu dient, das korrekte Beschreiben zu uberprufen.

Die beiden Programme zum Schreiben einer Spur und zum Schreiben eines Sektors sind sehr ähnlich. Die Schleifen sind dafur ausgelegt, 26 verschiedene Sektoren zu schreiben Wenn alle 26 Sektoren beschrieben sind, werden „ØFFH"-Bytes geschrieben, bis das Busybit zuruckgesetzt wird

Das Uberprüfen geschieht durch Lesen aller Sektoren (Multiple Sector Read), ohne daß die gelesenen Bytes abgespeichert werden Dadurch konnen CRC-Fehler erkannt werden, die im Statusregister angezeigt werden

Auf diese Weise wird jede geschriebene Spur sofort auf CRC-Fehler überpruft.

## 7.4 Formatieren und Überprüfen der Diskette

Die Struktogramme dieser Kommandos sind in Bild 7 6 dargestellt Formatieren einer Diskette heißt, daß alle 77 Spuren neu beschrieben werden Es besteht im wesentlichen aus der Wiederholung der Kommandos „Schreibe Spur" und STEPIN (siehe auch Anhang C)

Die Funktion „Uberprufen der Diskette" (Verify Disk) ist sehr nutzlich, um sicherzustellen, daß die Diskette frei von CRC-Fehlern ist Dabei werden sowohl die ID-Felder als auch die Datenfelder uberpruft Analog zum Formatieren wird die Funktion durch eine vielfache Aufeinanderfolge von „Uberprufe Spur"-Befehlen (Verify Track) und STEPIN-Kommandos realisiert

Der Softwaretreiber ist modular geschrieben, so daß zusatzliche Funktionen auf jeder Ebene leicht hinzugefugt werden konnen. Das zugehorige Unterprogramm wird einfach angehangt und als neuer Fall in die DO CASE-Anweisung eingefugt Vorstellbar sind z B. Funktionen wie „Lesen des Adreßfeldes" (read address mark) oder „Schreibe Sektor mit geloschtem Datenadreßfeld" (write sector with deleted data address mark)

**Bild 7.7**
**Hilfsprogramme**

```
init:
PROCEDURE

output(addr$8255+6)=init$ports
output(addr$8255) = 0ffh
cbl.track=input(track$179x)
cbl.sector=input(sector$179x)
cbl.status,cbl.seek$status=0
return
```

```
init$track:
PROCEDURE

call setb(0ffh,@track$form,96)
call setb(0,@track$form.pinam(0),6)
call setb(0,@track$form.pidam(0),6)
call setb(0,@track$form.pdtam(0),6)
call setb(0,@track$form.track,4)
call setb(0e5h,@track$form.dat(0),128)
track$form.inam = 0fch
track$form.idam = 0feh
track$form.dtam = 0fbh
track$form.idcrc,track$form.dtcrc = 0f7h
return
```

```
interleave:
PROCEDURE

FOR s = 0 to no$sect - 1
    sector$sequence(s) = s+1
return
```

**Bild 7.6**
**Formatieren und Überprüfen (Verify) der Diskette**

```
!format$disk:
!PROCEDURE
!
!call restore
!cbl.status = 0
!FOR cbl.track=0 to 76
!    !call wr$track
!    !output(comm$179x)=stepin$comm
!    !call time(1)
!    !WHILE (cbl.seek$status:=input(stat$179x) and 1) <> 0
!call restore
!return


!verify$disk:
!PROCEDURE
!
!call restore
!cbl.status=0
!FOR cbl.track=0 to 76
!    !call verifytrack
!    !WHILE (input(stat$179x) and 1) <> 0
!    !cbl.status = input(stat$179x) or cbl.status
!    !output(comm$179x)=stepin$comm
!    !call time(1)
!    !WHILE (cbl.seek$status:=input(stat$179x) and 1) <> 0
!call restore
!return
```

```
SERIES-III PL/M-86 V2.0 COMPILATION OF MODULE FDCMOD
OBJECT MODULE PLACED IN .F1:FDC51.OBJ
COMPILER INVOKED BY:  PLM86.86 :F1:FDC51.P86


                $optimize(3) compact debug
                $title( **** Floppy Disk Controller - SAB 179X - Software Driver **** )

                /*  Sharad Gandhi, Mikrocomputer-Systemberatung, Siemens, Munich */

                /*  no interrupt, single density, no dma, single drive  */

                /*=============================================================*/
   1            fdc$mod.
                do;

                /* communications block between user and fdc */
                /*-------------------------------------------

                /----------------------/
                / track      : byte    /  --  track number
                /----------------------/
                / sector     : byte    /  --  sector number
                /----------------------/
                / many       : byte    /  --  number of sectors
                /----------------------/
                / status     : byte    /  --  status after read/write operation
                /----------------------/
                / seek$status: byte    /  --  status after restore/seek operation
                /----------------------/
                / f          : byte    /  --  function desired
                /----------------------/
                / buff$ptr   : pointer /  --  pointer to buffer
                /----------------------/


                The user fills in the relevant parameters for a function (f)
                and calls FDC. The driver returns to user with status in
                the block reflecting the success of the operation  For some
                functions both status fields are relevant.


                f   function        description                          parameters
                =   ========        ===========                          ==========
                0   initialize      initialize the fdc driver            none
                1   restore         set r/w head to track 00             none
                2   seek            search for track in cb1.track field  track
                3   read sector(s)  seek track and read one/many sectors track,sector,many,buff$ptr
                4   write sector(s) seek track and write one/many sectors track,sector,many,buff$ptr
                5   read track      seek track and read out the whole track track,buff$ptr
                6   write track     seek track and write (format) the track track
                7   format disk     format the whole disk (IBM 3740 format) none
                8   verify disk     verify the CRC integrity of the disk    none
                */
                $eject
```

```
                /*=============================================*/

  2   1         declare cbl structure(
                    track   byte,
                    sector  byte,
                    many    byte,
                    status  byte,
                    seek$status byte,
                    f   byte,
                    buff$ptr pointer) external;

                /* SAB 179X port addresses */

  3   1         declare stat$179x literally '0fff0h';
  4   1         declare comm$179x literally '0fff0h';
  5   1         declare track$179x literally '0fff2h';
  6   1         declare sector$179x literally '0fff4h';
  7   1         declare data$179x literally '0fff6h';

  8   1         declare init$ports literally '8bh';
  9   1         declare addr$8255 literally '0ffd0h';

                /* sab 179x commands */

 10   1         declare res$comm literally  '00000101b';
 11   1         declare seek$comm literally '00010101b';
 12   1         declare stepin$comm literally   '01010001b';

 13   1         declare sector$length literally '128';
 14   1         declare no$sect literally '26';

 15   1         declare sector$sequence(26) byte public; /* for writing track */
 16   1         declare track$buff(250) byte; /* for writing track format */

 17   1         declare track$form structure(
                    g0(40) byte,            /* ff */
                    pinam(6) byte,          /* 00 */
                    inam byte,              /* fc */
                    g1(26) byte,            /* ff */
                    pidam(6) byte,          /* 00 */
                    idam byte,              /* fe */
                    track byte,             /* 00 */
                    side byte,              /* 00 */
                    sector byte,            /* 01 */
                    seclen byte,            /* 00 */
                    idcrc byte,             /* f7 */
                    g2(11) byte,            /* ff */
                    pdtam(6) byte,          /* 00 */
                    dtam byte,              /* fb */
                    dat(128) byte,          /* e5 */
                    dtcrc byte)             /* f7 */
                    at (@track$buff);

 18   1        declare gap0 byte public at(@track$form.g0(0));
 19   1        declare gap1 byte public at(@track$form.g1(0));
 20   1        declare sec byte public at(@track$form.sector);
                $eject
```

```
                /*=============================================*/

                /* external routines for fast data transfer */
                /*---------------------------------------------*/

  21    1       readsector: procedure(p1,p2) external;
  22    2       declare p1 word,
                    p2 pointer,
  23    2       end readsector;

  24    1       writesector: procedure(p1,p2) external;
  25    2       declare p1 word,
                    p2 pointer;
  26    2       end writesector,

  27    1       readtrack: procedure(p1,p2) external;
  28    2       declare p1 word,
                    p2 pointer;
  29    2       end readtrack;

  30    1       writetrack: procedure external;
  31    2       end writetrack;

  32    1       verifytrack: procedure external;
  33    2       end verifytrack;
                $eject
```

```
                /*================================================*/

 34   1         fdc: procedure public;
                /*====================*/

                /*------------------------------------------------*/
 35   2         interleave: procedure;  /* to fill sector$sequence */

 36   3             declare s byte;

 37   3             do s = 0 to no$sect - 1;
 38   4             sector$sequence(s) = s+1;
 39   4             end;

 40   3             return;
 41   3         end;
                /*------------------------------------------------*/
 42   2         init$track: procedure;

                /* initialize ibm 3740 track$format */

 43   3             call setb(0ffh,@track$form,96);
 44   3             call setb(0,@track$form.pinam(0),6);
 45   3             call setb(0,@track$form.pidam(0),6);
 46   3             call setb(0,@track$form.pdtam(0),6);
 47   3             call setb(0,@track$form.track,4);
 48   3             call setb(0e5h,@track$form.dat(0),128);
 49   3             track$form.inam = 0fch;
 50   3             track$form.idam = 0feh;
 51   3             track$form.dtam = 0fbh;
 52   3             track$form.idcrc,track$form.dtcrc = 0f7h;
 53   3             return;
 54   3         end;

                /*------------------------------------------------*/

 55   2         restore: procedure;
 56   3             output(comm$179x)=res$comm;
 57   3             call time(1);           /* delay */
 58   4             do while (input(stat$179x) and 1) <> 0;end;
 60   3             cbl.seek$status=input(stat$179x);
 61   3             cbl.track=input(track$179x);
 62   3             return;
 63   3         end;
                /*------------------------------------------------*/
 64   2         seek:   procedure;
 65   3             output(data$179x)=cbl.track;
 66   3             output(comm$179x)=seek$comm;
 67   3             call time(1);           /* delay */
 68   4             do while (input(stat$179x) and 1) <> 0;end;
 70   3             cbl.seek$status=input(stat$179x);
 71   3             return;
 72   3         end;
                /*------------------------------------------------*/
                $eject
```

```
 73   2      rd$sector: procedure,

 74   3          call seek;

 75   3          output(sector$179x)=cbl.sector;
 76   3          if cbl.many = 0 then cbl.many = 1;
 78   3          if cbl.many > no$sect then cbl.many = no$sect;
 80   3          call readsector(cbl.many*sector$length,cbl.buff$ptr);

 81   4          do while (input(stat$179x) and 1) <> 0,end;
 83   3          cbl.status=input(stat$179x);
 84   3          return;
 85   3      end;
             /*-------------------------------------------------------------*/
 86   2      wr$sector: procedure;

 87   3          call seek;

 88   3          output(sector$179x)=cbl.sector;
 89   3          if cbl.many = 0 then cbl.many = 1;
 91   3          if cbl.many > no$sect then cbl.many = no$sect;
 93   3          call writesector(cbl.many*sector$length,cbl.buff$ptr);
 94   4          do while (input(stat$179x) and 1) <> 0;end;
 96   3          cbl.status=input(stat$179x);

 97   3          return;
 98   3      end;
             /*-------------------------------------------------------------*/
 99   2      rd$track: procedure,

100   3          call seek;

101   3          call readtrack(1800h,cbl.buff$ptr);
102   4          do while (input(stat$179x) and 1) <> 0;end;
104   3          cbl.status=input(stat$179x);

105   3          return;
106   3      end;
             /*-------------------------------------------------------------*/
107   2      wr$track: procedure;

108   3          track$form.track=cbl.track;
109   3          call writetrack;
110   4          do while (input(stat$179x) and 1) <> 0;end;
112   3          cbl.status=input(stat$179x) or cbl.status;

113   3          call verifytrack;       /* auto-verify */
114   4          do while (input(stat$179x) and 1) <> 0;end;
116   3          cbl.status=input(stat$179x) or cbl.status;

117   3          return;
118   3      end;
             /*-------------------------------------------------------------*/
             $eject
```

```
119   2      format$disk: procedure;

120   3          call restore;

121   3          cbl.status = 0;
122   3          do cbl.track=0 to 76;
123   4          call wr$track;
124   4          output(comm$179x)=stepin$comm;
125   4          call time(1);           /* delay */
126   5          do while (cbl.seek$status:=input(stat$179x) and 1) <> 0;end;
128   4          end;

129   3          call    restore;

130   3          return;
131   3      end;
             /*-------------------------------------------------------------*/
132   2      verify$disk: procedure;

133   3          call    restore;

134   3          cbl.status=0;
135   3          do cbl.track=0 to 76;
136   4          call verifytrack;
137   5          do while (input(stat$179x) and 1) <> 0;end;
139   4          cbl.status = input(stat$179x) or cbl.status;

140   4          output(comm$179x)=stepin$comm;
141   4          call time(1);           /* delay */
142   5          do while (cbl.seek$status:=input(stat$179x) and 1) <> 0;end;
144   4          end;

145   3          call    restore;

146   3          return;

147   3      end;
             /*-------------------------------------------------------------*/
148   2      init:   procedure;

             /* initialize 8255a */

149   3          output(addr$8255+6)=init$ports;
150   3          output(addr$8255) = 0ffh;   /* all drives deselected */

             /* initialize parameters */

151   3          cbl.track=input(track$179x);
152   3          cbl.sector=input(sector$179x);
153   3          cbl.status,cbl.seek$status=0;

154   3          return;

155   3      end;
             $eject
```

```
                /*=======================================================*/

                /* entry point to fdc */

 156    2       begin$fdc:

                    output(addr$8255)=11111110b;    /* select drive 0, single density */
 157    2          call time(10);
 158    2          if (cb1.f * (input(stat$179x) and 10000000b)) = 0 then
                   /* if drive ready or f = 0 */
 159    2            do case cb1.f;
 160    3       c0:    call init;
 161    3       c1:    call restore;
 162    3       c2.    call seek;
 163    3       c3:    call rd$sector;
 164    3       c4:    call wr$sector;
 165    3       c5:    call rd$track;
 166    3       c6:    do;
 167    4              cb1.status=0;
 168    4              call init$track;
 169    4              call interleave;
 170    4              call seek;
 171    4              call wr$track;
 172    4              end;
 173    3       c7:    do;
 174    4              call init$track;
 175    4              call interleave;
 176    4              call format$disk;
 177    4              end;
 178    3       c8:    call verifydisk;
 179    3            end;
 180    2           else cb1.seek$status = input(stat$179x);
 181    2           output(addr$8255)=11111111b;   /* de-select drives */

 182    2          return;

 183    2       end fdc;
 184    1       end;
```

MODULE INFORMATION:

```
     CODE AREA SIZE     = 0317H    791D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 0115H    277D
     MAXIMUM STACK SIZE = 000EH     14D
     335 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS
```

END OF PL/M-86 COMPILATION

```
8086/8087/8088 MACRO ASSEMBLER    **** Floppy Disk Controller - SAB 179X - Transfer Routines *


SERIES-III 8086/8087/8088 MACRO ASSEMBLER V1.1 ASSEMBLY OF MODULE TRANS
OBJECT MODULE PLACED IN :F1:FDC52.OBJ
ASSEMBLER INVOKED BY:  ASM86.86 :F1:FDC52.A86


LOC  OBJ                  LINE     SOURCE

                             1 +1  $debug
                             2 +1  $title('**** Floppy Disk Controller - SAB 179X - Transfer Routines ****')
                             3
                             4     ; Sharad Gandhi, Mikrocomputer-Systemberatung, Siemens, Munich
                             5
                             6     ; Trans is called by FDC to do fast data transfer.
                             7     ; It is written for single density, 8 inch floppies
                             8
                             9     ;=======================================================
                            10             name    trans
                            11
                            12             public  readsector,writesector,readtrack,writetrack,verifytrack
                            13
                            14     ; SAB 179X port addresses
                            15     ;------------------------
                            16
  FFF0                      17     s179x   equ     0fff0h          ; status
  FFF0                      18     c179x   equ     0fff0h          ; command
  FFF2                      19     t179x   equ     0fff2h          ; track
  FFF4                      20     se179x  equ     0fff4h          ; sector
  FFF6                      21     d179x   equ     0fff6h          ; data
                            22
                            23     ; SAB 179X commands
                            24     ;-------------------
                            25
  0098                      26     r_s_com equ     98h             ; read sectors
  00B8                      27     w_s_com equ     0b8h            ; write sectors
  00E0                      28     r_t_com equ     0e0h            ; read track
  00F0                      29     w_t_com equ     0f0h            ; write track
  00D0                      30     f_i_com equ     0d0h            ; forced interrupt
  001A                      31     no_of_sect equ  26              ; number of sectors per track
  0080                      32     sect_len equ    128             ; length of sector in bytes
                            33
----                        34     sta     struc                   ;stack layout relative to base pointer
0000                        35     old_bp  dw      ?
0002                        36     old_ip  dw      ?
0004                        37     mem     dd      ?               ;buffer pointer
0008                        38     len     dw      ?               ;byte count
----                        39     sta     ends
                            40
                            41 +1  $eject
```

```
LOC  OBJ                  LINE     SOURCE

                            42     cgroup  group   code
                            43     dgroup  group   data
                            44
----                        45     data    segment public  data
                            46
                            47             extrn   sec:byte
                            48
----                        49     data    ends
                            50
----                        51     code    segment public  code'
                            52             assume  cs:cgroup,ds:dgroup
                            53             extrn   gap0:near,gap1.near,sectorsequence:near
                            54
                            55     ;--------------------------------------------------
0000                        56     delay   proc    near
0000 B11E                   57             mov     cl,30           ;25 microsec delay
0002 D2E9                   58             shr     cl,cl
0004 C3                     59             ret
                            60     delay   endp
                            61     ;--------------------------------------------------
                          62 +1    $eject
```

```
8086/8087/8088 MACRO ASSEMBLER    **** Floppy Disk Controller - SAB 179X - Transfer Routines *


LOC  OBJ                  LINE     SOURCE

0005                        63     readsector proc near
                            64
0005 55                     65             push    bp
0006 8BEC                   66             mov     bp,sp
                            67
0008 C47E04                 68             les     di,[bp].mem     ; es:di points to byte in buffer
000B FC                     69             cld
000C BBF0FF                 70             mov     bx,s179x        ; bx = status reg. address
000F BEF6FF                 71             mov     si,d179x        ; si = data reg. address
0012 BAF0FF                 72             mov     dx,c179x        ; dx = port address for OUT instr.
0015 B098                   73             mov     al,r_s_com      ; read sectors command
0017 EE                     74             out     dx,al
0018 E8E5FF                 75             call    delay           ; delay before reading status
001B 8B4E08                 76             mov     cx,[bp].len     ; number of bytes to be read
001E 8BD3                   77     lab10:  mov     dx,bx
0020 EC                     78     lab11:  in      al,dx           ; read in SAB 179X status
0021 A801                   79             test    al,1            ; if BUSY bit reset
0023 7413                   80             jz      lab12           ; then jump
0025 A802                   81             test    al,2            ; if DREQ bit not set
0027 74F7                   82             jz      lab11           ; then jump and read status again
                            83
0029 8BD6                   84             mov     dx,si           ; if DREQ bit set
002B EC                     85             in      al,dx           ; read in data byte
002C AA                     86             stosb                   ; and store it in buffer, increment di
002D E2EF                   87             loop    lab10           ; decrement cx, jump back if not zero
                            88
002F BAF0FF                 89             mov     dx,c179x        ; force interrupt command to SAB 179X
0032 B0D0                   90             mov     al,f_i_com      ; to stop transfers
0034 EE                     91             out     dx,al
0035 E8C8FF                 92             call    delay           ; delay before fdc routine reads status
                            93
0038 5D                     94     lab12:  pop     bp
0039 C20600                 95             ret     6
                            96
                            97     readsector endp
                            98 +1  $eject
```

```
8086/8087/8088 MACRO ASSEMBLER    **** Floppy Disk Controller - SAB 179X - Transfer Routines *


LOC  OBJ                  LINE     SOURCE

003C                        99     writesector proc        near
                           100
003C 55                    101             push    bp
003D 8BEC                  102             mov     bp,sp
                           103
003F 1E                    104             push    ds              ; save ds
0040 C57604                105             lds     si,[bp].mem     ; ds:si = points to byte in buffer
0043 FC                    106             cld
0044 BBF0FF                107             mov     bx,s179x        ; bx = status reg. address
0047 BFF6FF                108             mov     di,d179x        ; di = data reg. address1
004A BAF0FF                109             mov     dx,c179x        ; dx = port address for OUT instr.
004D B0B8                  110             mov     al,w_s_com      ; write sector command
004F EE                    111             out     dx,al
0050 E8ADFF                112             call    delay           ; delay before reading status
0053 8B4E08                113             mov     cx,[bp].len     ; number of bytes to be read
0056 8BD3                  114     lab20:  mov     dx,bx
0058 EC                    115     lab21:  in      al,dx           ; read in SAB 179X status
0059 A801                  116             test    al,1            ; if BUSY bit reset
005B 7416                  117             jz      lab22           ; then jump
005D A802                  118             test    al,2            ; if DREQ bit not set
005F 74F7                  119             jz      lab21           ; then jump and read status again
                           120
0061 8BD7                  121             mov     dx,di           ; if DREQ bit set
0063 AC                    122             lodsb                   ; read byte from buffer, increment si
0064 EE                    123             out     dx,al           ; and write it out to fdc
0065 E2EF                  124             loop    lab20           ; decrement cx, jump back if not zero
                           125
0067 E896FF                126             call    delay           ; delay to complete writing of sector
                           127
006A BAF0FF                128             mov     dx,c179x        ; force interrupt command to SAB 179X
006D B0D0                  129             mov     al,f_i_com      ; to stop transfers
006F EE                    130             out     dx,al
0070 E88DFF                131             call    delay           ; delay before fdc routine reads status
                           132
0073 1F                    133     lab22:  pop     ds
0074 5D                    134             pop     bp
0075 C20600                135             ret     6
                           136
                           137     writesector endp
                           138 +1  $eject
```

```
8086/8087/8088 MACRO ASSEMBLER    **** Floppy Disk Controller - SAB 179X - Transfer Routines *


LOC  OBJ                  LINE     SOURCE

0078                       139     readtrack proc  near
                           140
0078 55                    141             push    bp
0079 8BEC                  142             mov     bp,sp
                           143
007B C47E04                144             les     di,[bp].mem     ; es:di points byte in buffer
007E FC                    145             cld
007F BBF0FF                146             mov     bx,s179x        ; bx = status reg. address
0082 BEF6FF                147             mov     si,d179x        ; si = data reg. address
0085 BAF0FF                148             mov     dx,c179x        ; dx = port address for OUT instr.
0088 B0E0                  149             mov     al,r_t_com      ; read track command
008A EE                    150             out     dx,al
008B E872FF                151             call    delay           ; delay before reading status
008E 8B4E08                152             mov     cx,[bp].len     ; number of bytes to be read
0091 8BD3                  153     lab30:  mov     dx,bx
0093 EC                    154     lab31:  in      al,dx           ; read in SAB 179X status
0094 A801                  155             test    al,1            ; if BUSY bit reset
0096 740A                  156             jz      lab32           ; then jump
0098 A802                  157             test    al,2            ; if DREQ bit not set
009A 74F7                  158             jz      lab31           ; then jump and read status again
                           159
009C 8BD6                  160             mov     dx,si           ; if DREQ bit set
009E EC                    161             in      al,dx           ; read in data byte
009F AA                    162             stosb                   ; and store it in buffer, increment di
00A0 E2EF                  163             loop    lab30           ; decrement cx, jump back if not zero
                           164
00A2 5D                    165     lab32:  pop     bp
00A3 C20600                166             ret     6
                           167
                           168     readtrack endp
                          169 +1   $eject
```

```
8086/8087/8088 MACRO ASSEMBLER    **** Floppy Disk Controller - SAB 179X - Transfer Routines *


LOC  OBJ                  LINE     SOURCE

00A6                       170     writetrack proc         near
                           171
00A6 55                    172             push    bp
00A7 8BEC                  173             mov     bp,sp
                           174
00A9 FC                    175             cld
00AA BFF6FF                176             mov     di,d179x        ; di = data reg. address
00AD BE0000          E     177             mov     si,offset gap0  , si = begining of track buffer
00B0 BB0000                178             mov     bx,0            ; bx = sector number index
00B3 BAF0FF                179             mov     dx,c179x        ; dx = port address for OUT instr.
00B6 B0F0                  180             mov     al,w_t_com      ; write track command
00B8 EE                    181             out     dx,al
00B9 E844FF                182             call  delay           ; delay before reading status
00BC B9E800                183             mov     cx,232          ; byte count = 232
00BF 8A870000        E     184     lab40.  mov     al,[bx+offset sectorsequence] , byte from sectorsequence
00C3 A20000          E     185             mov     sec,al          ; write into sector field of ID
                           186
00C6 BAF0FF                187     lab41:  mov     dx,s179x
00C9 EC                    188     lab42.  in      al,dx           ; read in SAB 179X status
00CA A801                  189             test    al,1            ; if BUSY bit reset
00CC 7430                  190             jz      lab46           , then jump
00CE A802                  191             test    al,2            ; if DREQ bit not set
00D0 74F7                  192             jz      lab42           , then jump and read status again
                           193
00D2 8BD7                  194             mov     dx,di           ; if DREQ bit set
00D4 AC                    195             lodsb                   ; read byte from buffer, increment si
00D5 EE                    196             out     dx,al           ; and write it out to fdc
00D6 E2EE                  197             loop    lab41           ; decrement cx, jump back if not zero
                           198
00D8 43                    199             inc     bx              ; increment sector index
00D9 83FB1A                200             cmp     bx,no_of_sect   ; if sector index = number of sectors
00DC 7408                  201             je      lab43           ; all sectors are written, jump out
00DE B9B900                202             mov     cx,185          ; new byte count = 185
00E1 BE0000          E     203             mov     si,offset gap1  ; byte address in buffer = gap1
00E4 EBD9                  204             jmp     lab40           ; jump and write next sector
                           205
00E6 B95E01                206     lab43:  mov     cx,350          ; if all sectors written
00E9 B4FF                  207             mov     ah,0ffh         ; write a max. of 350 trailing FF's
00EB BAF0FF                208     lab44:  mov     dx,s179x
00EE EC                    209     lab45:  in      al,dx           ; read in SAB 179X status
00EF A801                  210             test    al,1            ; if BUSY bit reset
00F1 740B                  211             jz      lab46           ; then jump
00F3 A802                  212             test    al,2            ; if DREQ bit not set
00F5 74F7                  213             jz      lab45           ; then jump and read status again
                           214
00F7 8BD7                  215             mov     dx,di           ; if DREQ bit set
00F9 8AC4                  216             mov     al,ah           ; load al with FF
00FB EE                    217             out     dx,al           ; write it
00FC E2ED                  218             loop    lab44           ; decrement cx, if not zero jump
                           219
00FE 5D                    220     lab46:  pop     bp
00FF C3                    221             ret
                           222
                           223     writetrack endp
                           224 +1  $eject
```

```
8086/8087/8088 MACRO ASSEMBLER    **** Floppy Disk Controller - SAB 179X - Transfer Routines *

LOC  OBJ                  LINE     SOURCE

0100                       225     verifytrack proc        near
                           226
                           227     ;-------------------------------------------------------------
                           228     ;The track is verified against CRC errors in ID and DATA field.
                           229     ;All the sectors are read with a multiple sector read command.
                           230     ;The bytes read are not written into a buffer, but are ignored.
                           231     ;A CRC error after verify indicates a bad track.
                           232     ;-------------------------------------------------------------
                           233
0100 55                    234             push    bp
0101 8BEC                  235             mov     bp,sp
                           236
0103 BAF4FF                237             mov     dx,se179x       ;save sector register
0106 EC                    238             in      al,dx
0107 50                    239             push    ax
0108 B001                  240             mov     al,1            ;first sector number = 1
010A EE                    241             out     dx,al
                           242
010B BBF0FF                243             mov     bx,s179x        ; bx = status reg. address
010E BEF6FF                244             mov     si,d179x        ; si = data reg. address
0111 BAF0FF                245             mov     dx,c179x        ; dx = port address for OUT instr.
0114 B098                  246             mov     al,r_s_com      ; read sectors command
0116 EE                    247             out     dx,al
0117 E8E6FE                248             call    delay           ; delay before reading status
011A B9000D                249             mov     cx,no_of_sect*sect_len  ; cx = no. of formatted bytes/track
011D 8BD3                  250     lab50:  mov     dx,bx
011F EC                    251     lab51:  in      al,dx           ; read in SAB 179X status
0120 A801                  252             test    al,1            ; if BUSY bit reset
0122 7412                  253             jz      lab52           ; then jump
0124 A802                  254             test    al,2            ; if DREQ bit not set
0126 74F7                  255             jz      lab51           ; then jump and read status again
                           256
0128 8BD6                  257             mov     dx,si           ; if DREQ bit set
012A EC                    258             in      al,dx           ; read in the byte
012B E2F0                  259             loop    lab50           ; decrement cx, jump back if not zero
                           260
```

```
LOC  OBJ                  LINE     SOURCE

012D BAF0FF                261             mov     dx,c179x        ; force interrupt command to SAB 179X
0130 B0D0                  262             mov     al,f_i_com      ; to stop transfers
0132 EE                    263             out     dx,al
0133 E8CAFE                264             call    delay           ; delay before fdc routine reads status
                           265
0136 BAF4FF                266     lab52:  mov     dx,se179x       ;restore sector register
0139 58                    267             pop     ax
013A EE                    268             out     dx,al
                           269
013B 5D                    270             pop     bp
013C C3                    271             ret
                           272
                           273     verifytrack     endp
                           274
----                       275     code    ends
                           276
                           277             end

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

```
PL/M-86 COMPILER     **** floppy disk controller - sab 179x - application ****

SERIES-III PL/M-86 V2.0 COMPILATION OF MODULE FLOPPYUSER
OBJECT MODULE PLACED IN :F1:FDC5.OBJ
COMPILER INVOKED BY:  PLM86.86 :F1:FDC5.P86


                $optimize(3) compact debug
                $title('**** floppy disk controller - sab 179x - application ****')

   1            floppy$user:
                do;

                /* small user program invoking the fdc driver */
                /*--------------------------------------------*/

                /* communications block between user and fdc */

   2    1       declare cbl structure(
                    track   byte,
                    sector  byte,
                    many    byte,
                    status  byte,
                    seek$status byte,
                    f   byte,
                    buff$ptr pointer) public;
                /*-------------------------------------------------*/
   3    1       declare buffer(4000h) byte at(8000h);

   4    1       fdc: procedure external;
   5    2       end;

   6    1       start:  disable;

   7    1           cbl.buff$ptr=@buffer;
   8    1       setf:   cbl.f=0fh;
   9    1           do while 1;
  10    2           do while cbl.f <= 8;
  11    3           call fdc;
  12    3           cbl.f=0fh;
  13    3           end;
  14    2           end;

  15    1       end floppy$user;


MODULE INFORMATION:

     CODE AREA SIZE     = 0020H     32D
     CONSTANT AREA SIZE = 0004H      4D
     VARIABLE AREA SIZE = 000AH     10D
     MAXIMUM STACK SIZE = 0002H      2D
     37 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION
```

# 8. Weitere Lösungsmöglichkeiten und Erweiterungen

Die Hard- und Software, die in dieser Applikation vorgestellt wird, stellt natürlich nur eine von vielen möglichen Lösungen für eine Floppy-Disk-Steuerung dar. Für jeden einzelnen Funktionsblock der Hard- und Software gibt es fast beliebig viele Alternativen. Einige Anregungen sollen in diesem Kapitel vermittelt werden.

## 8.1 Datentrennung (Data Separation)

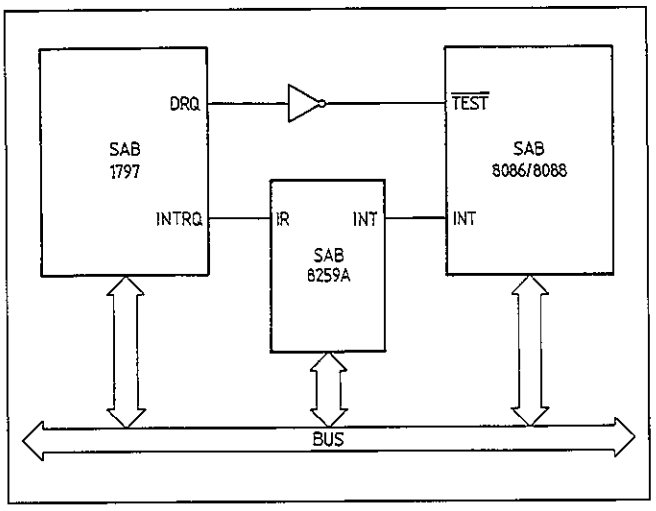
Es gibt eine fast unüberschaubare Anzahl von Schaltungen für die Trennung von Takt- und Dateninformation (Datentrennung). Die in der Applikation beschriebene Schaltung stellt die zur Zeit kompakteste Möglichkeit dar, alle üblichen Aufzeichnungsarten (5¼ und 8 Zoll, einfache und doppelte Schreibdichte) abzudecken. Einfachere Schaltungen sind z. B. dann möglich, wenn nur einfache Schreibdichte oder nur Minifloppies verwendet werden. Aufwendigere Schaltungen werden benötigt, wenn bestimmte Voraussetzungen erfüllt werden müssen. Ein Beispiel ist hier die Aufzeichnung mit doppelter Schreibdichte, ohne daß Schreibvorkompensation angewendet werden darf. Vorschläge für einige Alternativen finden Sie unter Punkt 3 und 4 in Anhang G (Literaturhinweise).

## 8.2 Verwendung von Anschluß $\overline{\text{TEST}}$ des SAB 8086/88

Die Prozessoren SAB 8086 und SAB 8088 haben einen Anschluß mit der Bezeichnung $\overline{\text{TEST}}$. Dieser Anschluß kann dazu verwendet werden, den SAB 179X die Datentransfers mit Hilfe des Signals DRQ steuern zu lassen. Die Prinzipschaltung zeigt Bild 8.1. Diese Anordnung hat außerdem den Vorteil, daß sehr schnelle Datentransfers ohne DMA möglich sind. Diese Lösung ist sogar schnell genug, eine 8-Zoll-Diskette mit doppelter Schreibdichte zu bedienen. Bild 8.2 zeigt am Beispiel einer Schreibroutine die notwendige Software. Der WAIT-Befehl entspricht einem „Springe auf der Stelle, bis der $\overline{\text{TEST}}$-Anschluß aktiv (low) ist". Auf diese Art wird der Ausgabebefehl ausgeführt, sobald DRQ aktiviert ist. Die gesamte Schleife dauert im ungünstigsten Fall 8 µs (bei einer Taktfrequenz von 5 MHz). Dies liegt weit unterhalb der Anforderungen einer 8-Zoll-Diskette mit doppelter Schreibdichte.

Diese Möglichkeit kann leicht in den vorgestellten Softwaretreiber eingebaut werden, wenn der $\overline{\text{TEST}}$-Anschluß im Anwendersystem zur Verfügung steht.

**Bild 8.1**
**Verwendung des $\overline{\text{TEST}}$-Anschlusses**




## 8.3 Einsatz von DMA oder nicht?

In vielen Systemen wird DMA verwendet, um die Daten zwischen der Floppy und dem Speicher zu transportieren. In manchen Fällen ist dies notwendig, da der Prozessor den Transfer nicht schnell genug abwickeln kann (z. B. Pollingverfahren bei 8-Zoll-Disketten mit doppelter Schreibdichte). Oft ist DMA aus Gründen der CPU-Verfügbarkeit wünschenswert. In den meisten Systemen, die DMA benutzen, wartet der Prozessor, bis der Datentransfer abgeschlossen ist. Hier ist das DMA-Verfahren eigentlich nicht notwendig. Häufig wird der DMA-Transfer jedoch gewählt, um eine Aufgabenteilung durchzuführen. Der Transport großerer Datenmengen wird durch die DMA-Steuerung ausgeführt. Dies gilt vor allem in Multitasking-Systemen. Hier kann der Prozessor eine andere Task bedienen, während der Datentransfer für die vorhergehende Task von der DMA-Steuerung durchgeführt wird.

Allerdings darf man dabei nicht vergessen, daß die freie Prozessorzeit durch den DMA-Transfer sich um die Zeit verkürzt, die der zweimalige Taskwechsel benötigt. Zusätzlich dazu ist die Verarbeitungsgeschwindigkeit des Prozessors herabgesetzt, da die DMA-Steuerung regelmäßig den Bus belegt.

Ob eine DMA-Steuerung eingesetzt wird, ist je nach Anwendungsfall zu entscheiden. Man muß dabei beachten, ob bei der verwendeten Kombination von Prozessor, Floppy Disk, Betriebssystem und der notwendigen Systemverfügbarkeit der Einsatz einer DMA-Steuerung sinnvoll ist.

**Bild 8.2**
**Übertragungsroutine zum Schreiben eines Sektors (mit WAIT)**

```
load_reg:
        lds     si,[bp].mem     ;buffer pointer
        mov     cx,[bp].len     ;byte count
        mov     dx,d179x        ;data reg. address

trans:  lodsb                   ;read byte from buffer, increment si
        wait                    ;wait till DRQ active
        out     dx,al           ;write byte to SAB 179X
        loop    trans           ;decrement cx, loop if cx not 0

finish:                         ;sector(s) written
```

# 9. Die Floppy-Disk-Steuerbausteine der SAB 279X-Familie

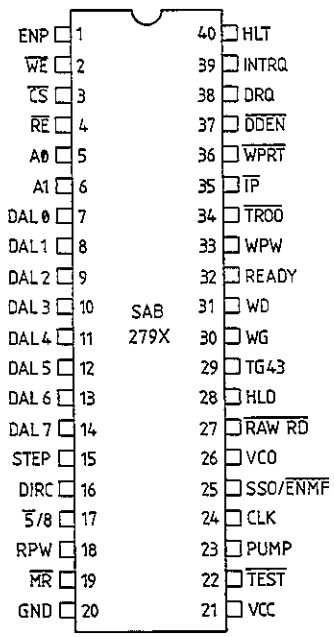
Die SAB 279X Floppy-Disk-Steuerbausteine sind das Ergebnis der konsequenten Weiterentwicklung der SAB 179X-Familie. Zusätzlich zu den bisherigen Funktionen wurden der Datenseparator (PLL-Prinzip) und die Schreibvorkompensationslogik (für Aufzeichnungen mit doppelter Schreibdichte) auf dem Chip integriert. Als Versorgungsspannung ist nur noch +5 V notwendig. Trotzdem sind die Bausteine der SAB 279X-Familie voll software-kompatibel zu den entsprechenden Bausteinen der SAB 179X-Familie. Das Bild 9.1 zeigt die Anschlußbelegung und eine Übersicht über die einzelnen Bausteine mit ihren unterschiedlichen Eigenschaften.

Die Rechnerschnittstelle ist ebenfalls pin-kompatibel mit den SAB 179X-Bausteinen, Unterschiede gibt es beim Interface zum Laufwerk.

Die Tabelle 9.2 enthält eine Übersicht über die Anschlüsse, die eine unterschiedliche Funktion haben.

Der $\overline{\text{ENMF}}$-Eingang bei SAB 2791 und SAB 2793 erlaubt den einfachen Wechsel zwischen 8-Zoll- und 5-Zoll-Laufwerken. Liegt ein „Ø-Pegel" an, wird der 2 MHz Eingangstakt (notwendig für 8-Zoll-Laufwerke) intern auf den für 5-Zoll-Laufwerke nötigen 1 MHz Takt heruntergeteilt. Eine „1" an $\overline{\text{ENMF}}$ läßt den Takt unverändert.

**Bild 9.1**
**Übersicht über die SAB 279X-Bausteinfamilie und Anschlußbelegung**




- Datenseparator auf dem Baustein
- Schreibvorkompensationslogik auf dem Baustein
- Versorgungsspannung +5 V
- Unterstützung der IBM Formate IBM 3740 (FM) und IBM 34 (MFM)
- TTL-kompatibel
- Voll software-kompatibel zu SAB 179X

| Eigenschaften | SAB 2791 | SAB 2793 | SAB 2795 | SAB 2797 |
|---|---|---|---|---|
| Einfache Schreibdichte (FM) | X | X | X | X |
| Doppelte Schreibdichte (MFM) | X | X | X | X |
| Nichtinvertierter Datenbus | | X | | X |
| Invertierter Datenbus | X | | X | |
| Doppelseitige Disketten | | | X | X |
| Interner Taktvorteiler | X | X | | |

**Tabelle 9.2**
**Unterschiedliche Anschlußbelegung zwischen SAB 179X und SAB 279X**

| SAB 179X | | SAB 279X | |
|---|---|---|---|
| PIN | Name | Name | Funktion |
| 1 | nicht belegt | ENP | Freigabe der Vorkompension |
| 17 | EARLY | 5/8 | Auswahl 5-Zoll-, 8-Zoll-Laufwerke |
| 18 | LATE | RPW | Breite des Leseimpulses |
| 22 | $\overline{\text{TEST}}$ | $\overline{\text{TEST}}$ | Freigabe des Abgleichmodus |
| 23 | HLT | PUMP | Anschluß für externes Filter |
| 25 | RG (SAB 1791/93) | $\overline{\text{ENMF}}$ (SAB 2791/93) | Freigabe des CLK-Vorteilers für Minifloppys |
| | SSO (SAB 1793/97) | SSO (SAB 2795/97) | Seitenauswahl |
| 26 | RCLK | VCO | Oszillatorabgleich |
| 33 | $\overline{\text{WF/VFOE}}$ | WPW | Schreibimpulsbreite |
| 38 | DRQ (Open Collector-Ausgange) | DRQ | (Totem-pole-Ausgänge) |
| 39 | INTRQ (Open Collector-Ausgange) | INTRQ | (Totem-pole-Ausgange) |
| 40 | VDD | HLT | |

## 9.1 Hardware einer Floppy-Disk-Steuerung mit SAB 279X

In Bild 9.4 ist das Blockschaltbild einer Floppy-Disk-Steuerung mit einem Baustein der SAB 279X-Familie dargestellt. Sie kann mit drei Bausteinen aufgebaut werden: SAB 279X + 2 TTL-Treiberbausteine. Im Gegensatz dazu benötigt man beim SAB 179X mindestens 6 Bausteine (siehe Bilder 9.3 und 5.1).

Bild 9.5 zeigt die komplette Schaltung einer Steuerung mit dem SAB 2797. Sie ist bei der zusätzlich notwendigen Hardware (Takterzeugung, Laufwerkauswahl) so ausgelegt, daß die in dieser Applikation beschriebenen Softwaretreiber unverändert übernommen werden können. In den Datenleitungen kann unter Umständen ein zusätzlicher Treiberbaustein notwendig sein. Dies hängt vom verwendeten Floppy-Disk-Steuerbaustein zusammen mit der Busstruktur (invertierend – nicht invertierend) sowie vom Systemaufbau und der Systemgröße ab.

Die Anmerkungen zur Schaltung für die Laufwerkauswahl und den Leitungstreibern zum Floppy-Disk-Laufwerk aus Kapitel 5 sind für die SAB 279X ebenfalls gültig. Nicht mehr notwendig sind die Pull-Up-Widerstände bei den Anschlüssen DRQ und INTRQ.

**Bild 9.3**
**Blockschaltbild für eine Floppy-Disk-Steuerung mit SAB 179X**

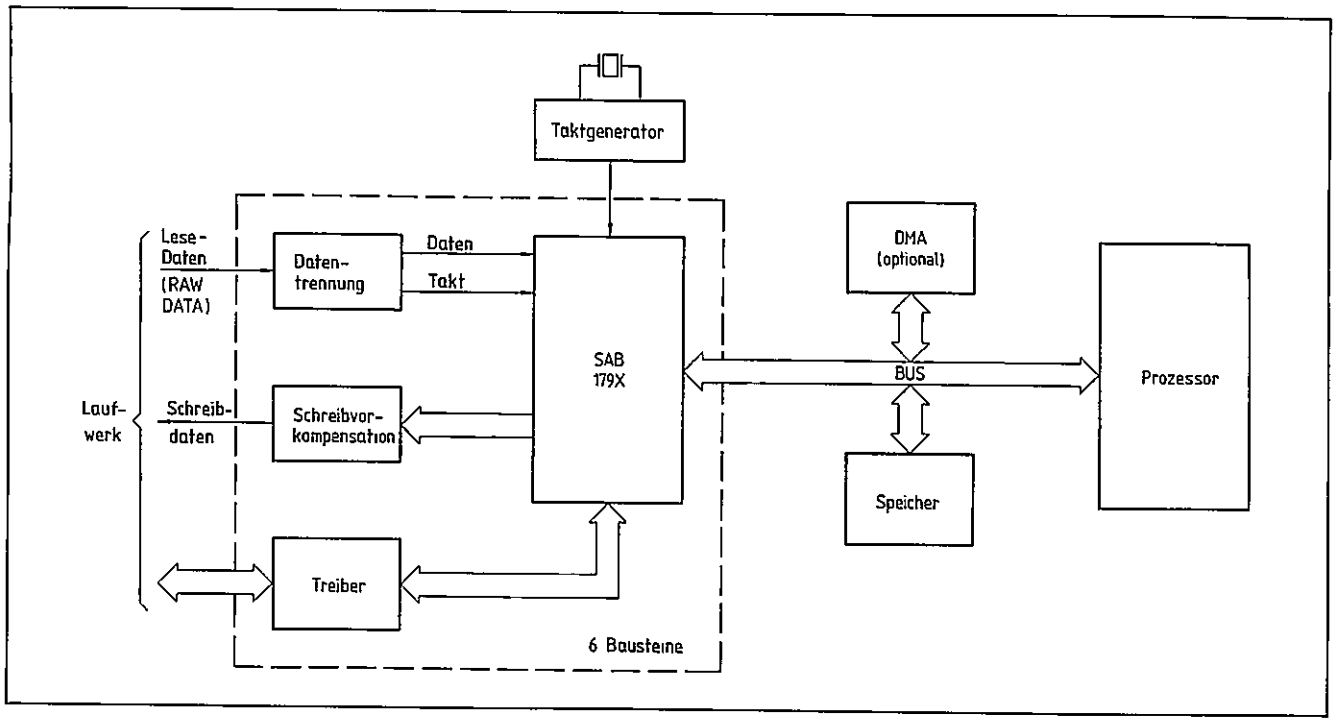



**Bild 9.4**
**Blockschaltbild für eine Floppy-Disk-Steuerung mit SAB 279X**

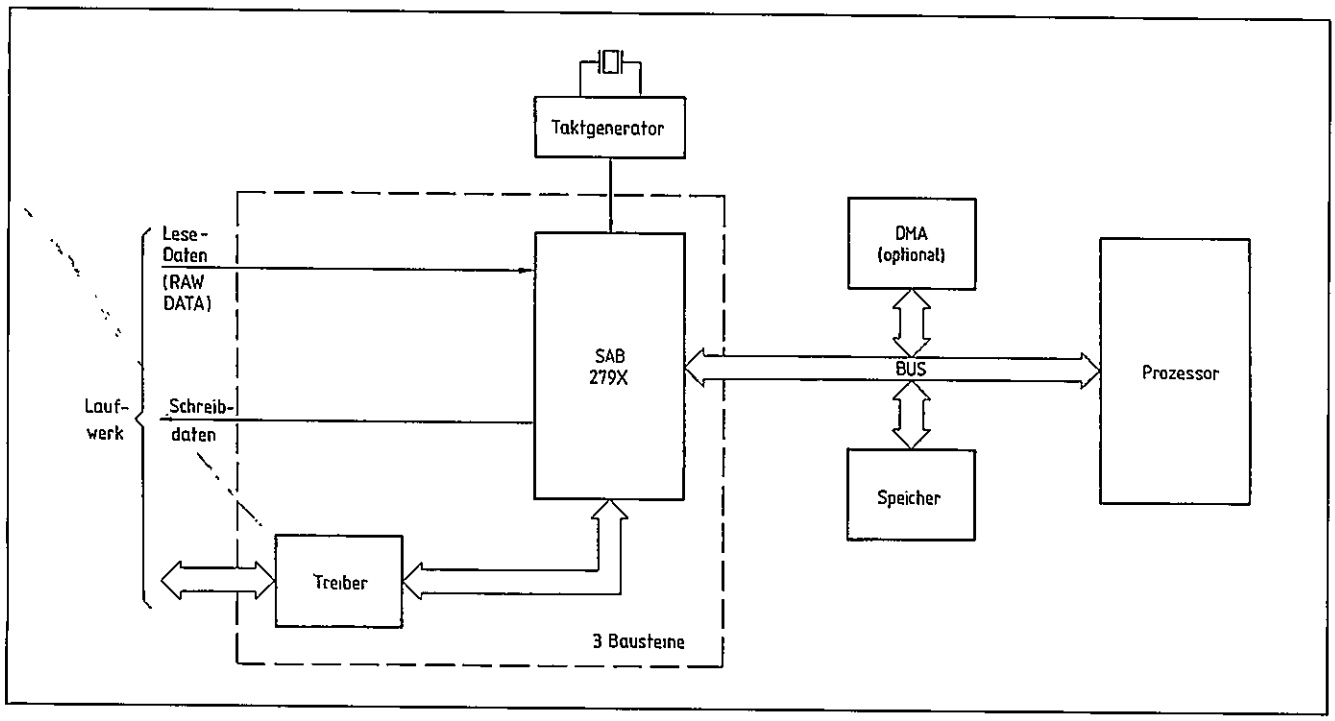
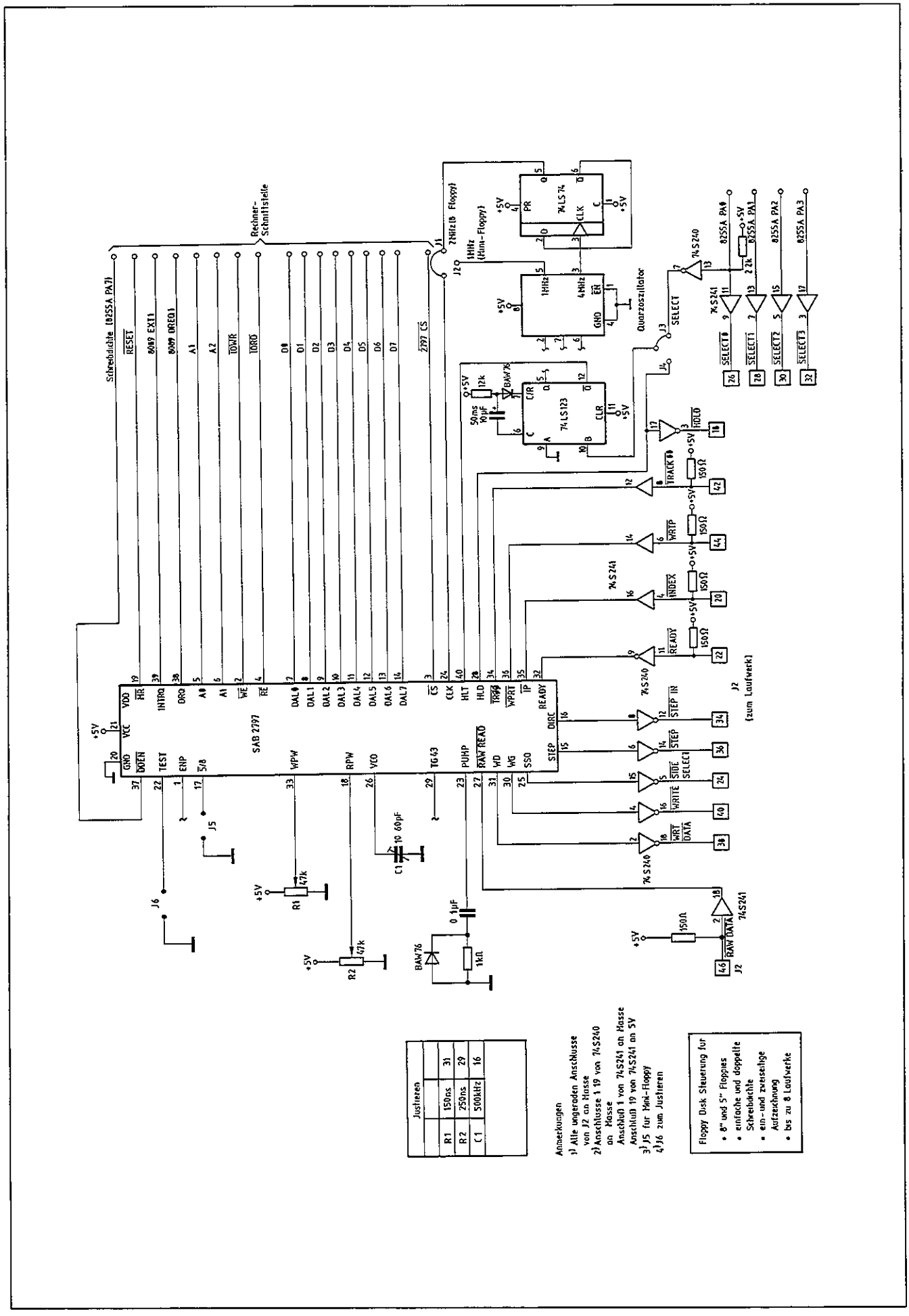


**Bild 9.5**
**Floppy-Disk-Steuerung mit SAB 2797**




| Justieren | | |
|---|---|---|
| R1 | 150ns | 31 |
| R2 | 250ns | 29 |
| C1 | 500kHz | 16 |

Anmerkungen
1) Alle ungeraden Anschlüsse von J2 an Masse
2) Anschlüsse 1 19 von 74S240 an Masse
Anschluß 1 von 74S241 an Masse
Anschluß 19 von 74S241 an 5V
3) J5 für Mini-Floppy
4) J6 zum Justieren

Floppy Disk Steuerung für
- 8" und 5" Floppies
- einfache und doppelte Schreibdichte
- ein- und zweiseitige Aufzeichnung
- bis zu 8 Laufwerke

Die SAB 279X-Bausteine benötigen eine geringfügige externe Beschaltung, die zum Betrieb des Datenseparators und der Schreibvorkompensation erforderlich sind.

Es handelt sich dabei um zwei Trimmerwiderstände, einen Trimmerkondensator und ein einfaches RC-Filter. Außerdem wird eine Steckbrücke zum Umschalten des Bausteines in den Einstellmodus benötigt (siehe Bild 9.6).

Das Vorgehen beim Abgleichen wird im folgenden Kapitel beschrieben.

## 9.2 Abgleich der SAB 279X-Hardware

Um den Abgleich vornehmen zu können, muß der $\overline{\text{TEST}}$-Eingang auf Ø-Pegel gelegt werden. Dadurch werden drei Anschlüsse umdefiniert und liefern die internen Signale, die zur Einstellung notwendig sind. Es handelt sich dabei um folgende Anschlüsse:

| Anschluß | Normale Betriebsart | Betriebsart im Einstellmodus |
|---|---|---|
| 31 | WD: Schreibdaten | Ausgabe von Pulsen für Einstellung der Schreibvorkompensationszeit: Pulsbreite = Kompensationszeit |
| 29 | TG43: Spurnummer im Spurregister ist größer als 43 | Ausgabe von Pulsen für Einstellung des Lesetaktes (RCLK): Pulsbreite = Lesepulsbreite |
| 16 | DIRC: Bewegungsrichtung des Kopfes | Ausgabe des Taktes des internen VCO's (Voltage Controlled Oscillator) = Lesetakt |

In Bild 9.6 sind die zur Einstellung des Datenseparators und der Schreibvorkompensationsschaltung notwendigen Beschaltungen und Signale zusammengefaßt.

Der Abgleich der beiden Funktionseinheiten läuft wie folgt ab:

1. Schreibvorkompensation
   - 1.1 $\overline{\text{TEST}}$ = 1 (Brücke offen), Baustein rücksetzen (Impuls von mindestens 50 µs an $\overline{\text{MR}}$ [$\overline{\text{MASTER RESET}}$]).
   - 1.2 $\overline{\text{TEST}}$ = Ø (Brücke geschlossen), Oszilloskop an Anschluß 31 (WD) anschließen.
   - 1.3 Mit Trimmer an Anschluß 33 (WPW) die gewünschte Schreibvorkompensationszeit (z.B. 150 ns) einstellen (= Breite der Impulse).
   - 1.4 $\overline{\text{TEST}}$ = 1 (für normalen Betrieb).
2. Datenseparator
   - 2.1 An $\overline{\text{DDEN}}$ und $\overline{5/8}$ die notwendigen logischen Pegel anlegen.
   - 2.2 $\overline{\text{TEST}}$ = 1, Baustein rücksetzen (Impuls von mindestens 50 µs an $\overline{\text{MR}}$).
   - 2.3 $\overline{\text{TEST}}$ = Ø, Oszilloskop an Anschluß 29 (TG43) anschließen.
   - 2.4 Mit Trimmer an Anschluß 18 (RPW) eine Pulsbreite von 1/8 der Datenrate einstellen (siehe Tabelle 9.7, RPW).
   - 2.5 Oszilloskop an Anschluß 16 (DIRC) anschließen.
   - 2.6 Mit Trimmerkondensator an Anschluß 26 (VCO) die Oszillatormittenfrequenz auf die notwendige Datenrate einstellen (siehe Tabelle 9.7, F).
   - 2.7 $\overline{\text{TEST}}$ = 1 (für normale Betriebsart).

**Tabelle 9.7**
**Einstellungen für verschiedene Laufwerke**

| 5/8 \ DDEN | Ø | 1 |
|---|---|---|
| Ø | P = 150 ns*)<br>RPW = 500 ns<br>F = 250 kHz | P = 150 ns*)<br>RPW = 1 µs<br>F = 125 kHz |
| 1 | P = 150 ns*)<br>RPW = 250 ns<br>F = 500 kHz | P = 150 ns<br>RPW = 500 ns<br>F = 250 kHz |

P = Schreibvorkompensationszeit (150 ns haben sich als günstigster Wert erwiesen)
RPW = Leseimpulsbreite
F = Frequenz des VCO (RCLK)
*) wenn erforderlich

**Anmerkungen:**

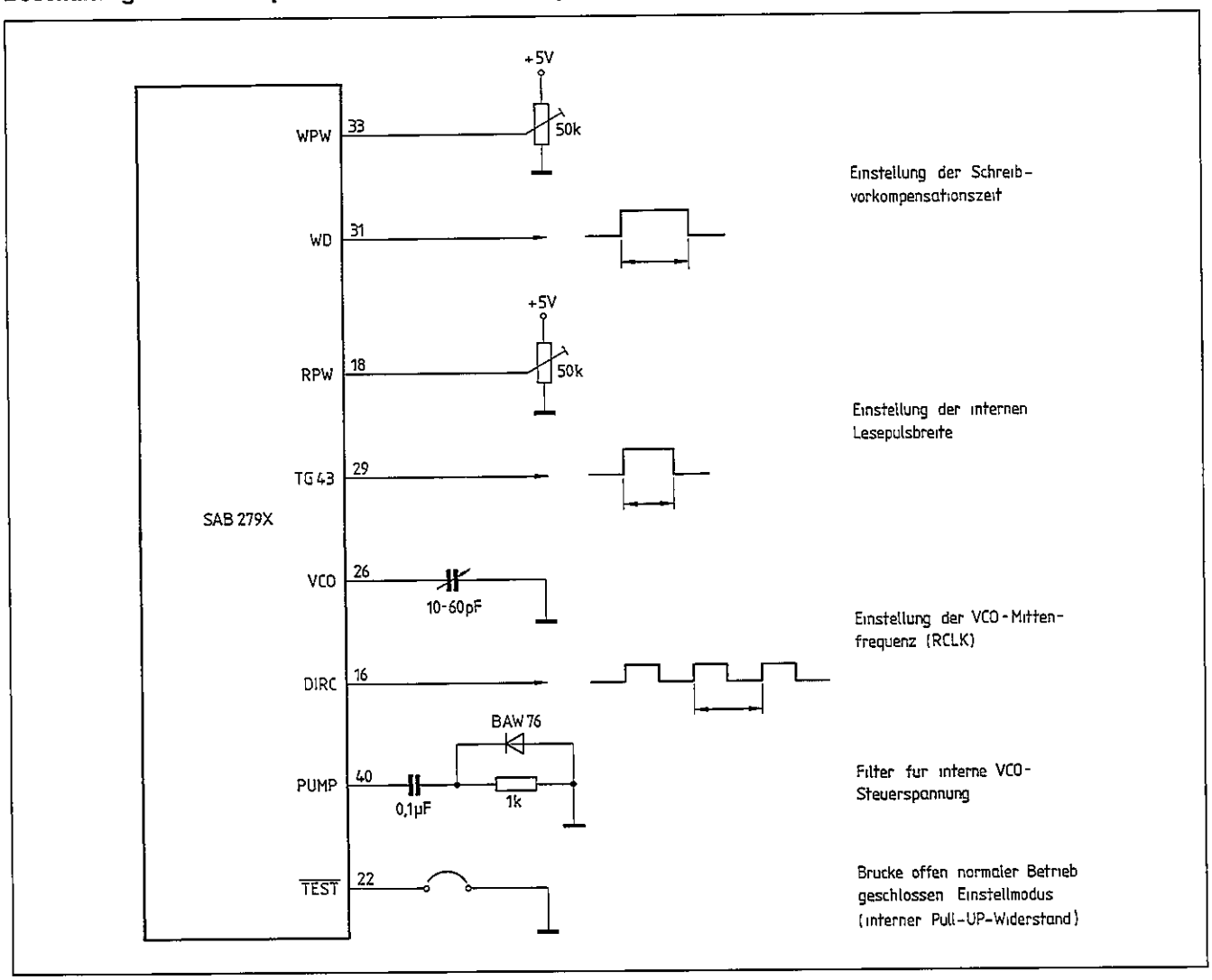
1. Die Beschaltung $\overline{\text{ENMF}}$ bei SAB 2791 und SAB 2793 ist für den Einstellvorgang ohne Bedeutung (natürlich nicht für den normalen Betrieb!).
2. Bei der Einstellung der Schreibvorkompensationszeit (WPW) für Aufzeichnungen mit doppelter Schreibdichte (MFM) und Vorkompensation wird gleichzeitig die Schreibimpulsbreite für diese Betriebsart eingestellt. Die folgende Tabelle zeigt die Schreibimpulsbreiten für die verschiedenen Betriebsarten:

| Aufzeichnungsart | ENP | Laufwerk | |
|---|---|---|---|
| | | 5¼ Zoll | 8 Zoll |
| MFM | Ø | 500 ns | 250 ns |
| | 1 | 2 × WPW | 1 × WPW |
| FM | X | 1 µs | 500 ns |

3. Wird der Datenseparator z.B. für 8-Zoll-Laufwerke mit doppelter Schreibdichte abgeglichen, ist er auch für die restlichen möglichen Laufwerkstypen richtig eingestellt (und umgekehrt). Welche Einstellung vorgenommen werden muß, hängt nur von der Laufwerksbestimmung an $\overline{\text{DDEN}}$ und $\overline{5/8}$ sowie der Taktversorgung (Einstellung von $\overline{\text{ENMF}}$ bei SAB 2791 und SAB 2793) ab.
4. Um ein einwandfreies Arbeiten des internen VCO (Voltage Controlled Oscillator) zu gewährleisten, muß der $\overline{\text{TEST}}$-Eingang auf „1"-Pegel liegen (= offenem Eingang), wenn ein Resetimpuls angelegt wird.

Die Inbetriebnahme der Floppy-Disk-Steuerung sollte in derselben Reihenfolge vorgenommen werden, wie es bei den Bausteinen SAB 179X beschrieben ist (siehe Kapitel 6). Die Erfahrung zeigt, daß die Inbetriebnahme der Steuerung keinerlei Probleme bereitet. Allerdings muß man darauf achten, daß der $\overline{\text{TEST}}$-Anschluß nicht noch versehentlich auf „Ø-Pegel" gelegt ist. In diesem Fall können nicht vorhersehbare Effekte auftreten.

**Bild 9.6**
**Beschaltung von Datenseparator und Schreibvorkompensator, Signale beim Abgleich**



# Anhang A

## Beschreibung eines typischen FD-Laufwerkes

Als Beispiel für die Eigenschaften eines typischen 8-Zoll-Floppy-Disk-Laufwerkes ist ein Auszug aus der Beschreibung des Siemens-Laufwerkes FD 200-8 beigefügt. Ebenfalls enthalten ist eine Beschreibung der Interfacesignale.

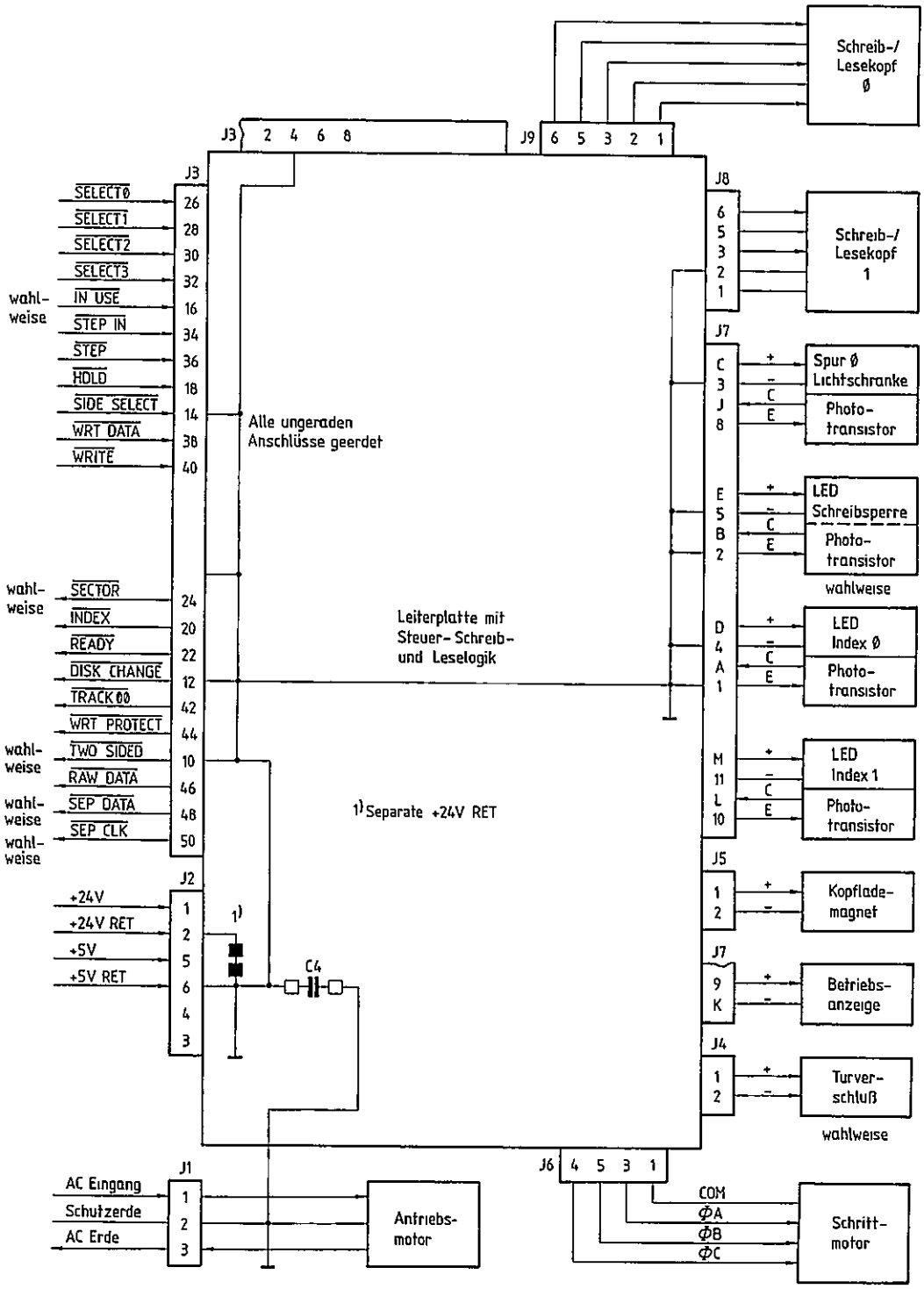
## Technische Daten

| Funktion | Technische Daten | |
|---|---|---|
| | Schreibdichte | |
| | Einfach | Doppelt |
| Diskettenart | DIN 66237 | DIN 66237 |
| Speicherkapazität, unformiert<br>pro Diskette<br>pro Spur<br>pro Diskettenseite | <br>6,4 MBits<br>41,7 KBits<br>3,2 MBits | <br>12,8 MBits<br>83,4 KBits<br>6,4 MBits |
| Anzahl der Spuren | 154 | 154 |
| Spurdichte pro mm | 1,89 (48 Spuren/Zoll) | 1,89 (48 Spuren/Zoll) |
| Schreibdichte pro mm<br>äußere Spur<br><br><br>innere Spur | <br>72,28 Bits<br>(1836 Bits/Zoll)<br>(144,56 Flußwechsel)<br>128,66 Bits<br>(3268 Bits/Zoll)<br>(257,32 Flußwechsel) | <br>144,56 Bits<br>(3672 Bits/Zoll)<br>(144,56 Flußwechsel)<br>257,32 Bits<br>(6536 Bits/Zoll)<br>(257,32 Flußwechsel) |
| Aufzeichnungsverfahren | FM | MFM |
| Drehzahl | 360 U/min. ± 2,5 % | 360 U/min. ± 2,5 % |
| Zugriffszeit pro Spur<br>Durchschnitt<br>Maximum | <br>83,33 ms<br>175,6 ms | <br>83,33 ms<br>175,6 ms |
| Zugriffszeit<br>Spur zu Spur<br>Spur –00– zu Spur –76–<br>über 38 Spuren | <br>3 – 6 ms<br>456 ms<br>226 ms | <br>3 – 6 ms<br>456 ms<br>226 ms |
| Kopfberuhigungszeit | 8 ms | 8 ms |
| Kopfladezeit | 25 ms | 25 ms |
| Datenübertragungsrate | 250 KBit/s | 500 KBit/s |
| Lösch-/Schreib-Umschaltzeit<br>Sattellöschen<br>Tunnellöschen | <br>50 µs<br>580 µs<br>(erforderlich für einwandfreies Lesen nach dem Schreiben) | <br>50 µs<br>580 µs |
| Schreib-/Lesekopf (Einspaltkopf mit . . .) | Tunnellöschspule | Sattellöschspule |
| Abstand zwischen Schreib-/Lesespalt und Löschspalt | 0,914 mm | entfällt |
| Spurbreite | 0,33 mm<br>(0,013 Zoll) | 0,33 mm<br>(0,013 Zoll) |
| Löschbreite | 0,152 mm<br>(0,006 Zoll)<br>an beiden Seiten der Spur | 0,177 mm<br>(0,007 Zoll)<br>an beiden Seiten der Spur |
| Spurabstand | 0,529 mm<br>(0,02083 Zoll) | 0,529 mm<br>(0,02083 Zoll) |

## **Technische Daten** (Fortsetzung)

| Funktion | Technische Daten | |
|---|---|---|
| Spurmittenradius | $Rn = (51{,}537 + \frac{76 - N}{48} \times 25{,}4)$ mm<br>$Rn = (2{,}029 + \frac{76 - N}{48})$ Zoll<br>N = Spurnummer (0 – 76) | |
| Logikpegel<br>Laufwerk<br><br>Schnittstelle | <br>1 = +2,5 bis +5,5 V<br>0 = 0,0 bis +0,4 V<br>1 = 0,0 bis +0,4 V<br>0 = +2,5 bis +5,5 V | |
| Netzspannung | 230 V (180 – 253 V), 50 Hz ± 0,5 Hz<br>230 V (180 – 253 V), 60 Hz ± 0,5 Hz<br>115 V (90 – 127 V), 50 Hz ± 0,5 Hz<br>115 V (90 – 127 V), 60 Hz ± 0,5 Hz | |
| Spannungsausfall | 100 %, einmal 10 ms pro 600 Sekunden | |
| **Motorstrom (max.)** | | |
| Anlauf<br><br>Betrieb | 1,0 A bei 115 V<br>0,6 A bei 230 V<br>0,5 A bei 115 V<br>0,3 A bei 230 V | |
| Gleichspannungen | +24 V ± 5 %, 1,7 A max<br>+ 5 V ± 5 %, 1,2 A max | |
| **Zuverlässigkeit** | | |
| MTBF<br>MTTR | 6000 Stunden (nach anfänglichen 200 Stunden)<br>weniger als 20 Minuten | |
| Lesefehler<br>behebbare<br>nicht behebbare (nach 10 Versuchen) | <br>weniger als 1 auf $10^9$<br>weniger als 1 in $10^{12}$ | |
| **Umgebungstemperaturen** | **Betrieb** | **Transport/Lagerung** |
| Temperaturbereich | 4,4 – 46° C | 0 – 65° C |
| Relative Feuchte | 20 – 80 %<br>keine Betauung | 5 – 90 %<br>keine Betauung |
| Höhe | – 300 m bis 3000 m | – 300 m bis 14000 m |
| Wärmeabgabe | max. 88 W | — |
| Gewicht | 5,7 kg | |

## Verbindungsdiagramm



## Eingangsschnittstellensignale

| Signal | Bedeutung |
|---|---|
| $\overline{\text{SELECT Ø}}$ bis $\overline{\text{SELECT 3}}$ | In der Grundausführung ist jedem Laufwerk eine Auswahlleitung zugeordnet Logisch 1 (niedrig) wählt das Laufwerk aus und gibt die Schnittstellenschaltungen frei Logisch Ø (hoch) sperrt alle Eingänge außer den Auswahlleitungen und alle Ausgangsleitungen außer $\overline{\text{READY}}$<br>Wenn die Binärauswahlfunktion verwendet wird, gibt ein gesetztes (niedriges) Signal $\overline{\text{SELECT Ø}}$ die Laufwerkauswahl frei. $\overline{\text{SELECT 1}}$, $\overline{\text{SELECT 2}}$ und $\overline{\text{SELECT 3}}$ übertragen einen aus drei Bits bestehenden binären Code, mit dem das Laufwerk adressiert wird.<br>Die Betriebsanzeige wird eingeschaltet und die Türverschlußfunktion aktiviert (falls verwendet), wenn letztere die $\overline{\text{SELECT}}$-Leitung überwacht |
| $\overline{\text{HDLD}}$ | Logisch 1 (niedrig) erregt den Kopflademagnet Dieser gibt dann den Andruckarm frei, der den Kontakt zwischen der Diskette und dem Schreib-/Lesekopf herstellt Nach dem Befehl $\overline{\text{HDLD}}$ und vor einer Lese- oder Schreiboperation muß eine Wartezeit von 25 ms eingehalten werden<br>Das Laufwerk muß ausgewählt sein, bevor $\overline{\text{HDLD}}$ gegeben werden kann $\overline{\text{HDLD}}$ muß nicht gesetzt werden, wenn die entsprechende HDLD-Option verwendet wird<br>Die Betriebsanzeige wird eingeschaltet und die Türverschlußfunktion (falls verwendet) wird aktiviert, falls die Anzeigelogik das Signal $\overline{\text{HDLD}}$ überwacht |
| $\overline{\text{STEP}}$ | Die Rückflanke eines jeden gesetzten (niedrigen) $\overline{\text{STEP}}$-Impulses bewirkt eine Kopfbewegung um eine Spurbreite<br>Die Impulse müssen über eine Dauer von mindestens 1 µs gehalten werden Der Zeitabstand zwischen den Impulsen muß mindestens 3 ms und darf höchstens 6 ms betragen, wenn eine Bewegung über mehrere Spuren erfolgen soll<br>Voraussetzung für Schreib-/Lesekopfbewegungen sind<br>1. Schreiboperationen gesperrt<br>2. Laufwerk selektiert oder Radialschrittfunktion angeschaltet<br>3 $\overline{\text{HDLD}}$ gesetzt |
| $\overline{\text{STEP IN}}$ | Bestimmt die Bewegungsrichtung Logisch 1 = in Richtung auf Spur 76, logisch Ø = in Richtung auf Spur ØØ<br>Der Signalzustand darf bei weniger als 1 µs vor der Rückflanke des $\overline{\text{STEP}}$-Impulses nicht mehr verändert werden $\overline{\text{STEP IN}}$ wird nur freigegeben, wenn das Laufwerk ausgewählt ist oder wahlweise die Radialschrittfunktion verwendet wird |
| $\overline{\text{WRITE}}$ | Logisch 1 (niedrig) sperrt die Leseschaltungen, die Schreib-/Lesekopfbewegungen und schaltet den Schreibstrom im Schreib-/Lesekopf ein<br>$\overline{\text{WRITE}}$ muß eine Bitperiode vor dem ersten Schreibdatenimpuls gesetzt sein und bis 2 Bitperioden nach dem letzten Schreibdatenimpuls gesetzt bleiben<br>Der Löschstrom wird bei Sattellöschung von WRITE GATE ein- und ausgeschaltet Wahlweise wird bei Tunnellöschung der Löschstrom 200 µs nach $\overline{\text{WRITE}}$ eingeschaltet und 530 µs nach dem Rücksetzen von $\overline{\text{WRITE}}$ ausgeschaltet<br>Voraussetzungen für die Freigabe von $\overline{\text{WRITE}}$ sind<br>1 Laufwerk selektiert<br>2 $\overline{\text{HDLD}}$ gesetzt<br>3 Diskette ohne Schreibsperrkerbe eingelegt (falls die Schreibsperrfunktion verwendet wird). |
| $\overline{\text{WRT DATA}}$ | Die Übergänge des $\overline{\text{WRT DATA}}$-Impulses von hoch zu niedrig bzw niedrig zu hoch ändern die Polarität des Schreibstromflusses durch den Schreib-/Lesekopf (siehe Bild 5 2)<br>$\overline{\text{WRT DATA}}$ bleibt 150 bis 1100 ns lang niedrig, um die nominelle Dauer der Daten- und Taktimpulse festzulegen<br>Zur Freigabe von $\overline{\text{WRT DATA}}$ muß das Laufwerk selektiert sein |
| $\overline{\text{IN USE}}$ (Zusatz) | Kann für das Einschalten der Betriebsanzeige und die Aktivierung der Türanschlußfunktion verwendet werden, falls letztere das Signal $\overline{\text{IN USE}}$ überwacht |
| $\overline{\text{SIDE SELECT}}$ | Logischer Pegel „Ø" (+5 V) gibt Kopf Ø frei<br>Logischer Pegel „1" (0 V) gibt Kopf 1 frei |

## Ausgangsschnittstellensignale

| Signal | Bedeutung |
|---|---|
| $\overline{\text{READY}}$ | Wenn dieses Signal gesetzt (niedrig) ist, sind alle Spannungen vorhanden und 2 Spuranfangsimpulse erkannt worden. |
| $\overline{\text{WRT PROTECT}}$ | Wird gesetzt (niedrig), wenn eine schreibgeschützte Diskette eingelegt wurde. Die Schreibschaltungen werden gesperrt. Das Laufwerk muß ausgewählt sein. |
| $\overline{\text{TRACK ØØ}}$ | Wird gesetzt (niedrig), wenn der Schreib-/Lesekopf auf Spur ØØ positioniert ist. Nur verfügbar, wenn das Laufwerk ausgewählt ist. |
| $\overline{\text{INDEX}}$ | Dieser 1,7 ms niedrige Impuls tritt einmal pro Diskettenumdrehung auf.<br>Bei Verwendung der Hardsektorfunktion wird die Impulsdauer auf nominal 0,4 ms verringert. Dieser Impuls wird für die Synchronisierung der Datenübertragung durch die Steuerung verwendet. |
| $\overline{\text{DISK CHANGE}}$ | Ein gesetztes (niedriges) Signal besagt, daß das Laufwerk gerade eingeschaltet wurde oder daß die Ladeklappe geöffnet und wieder geschlossen wurde, während das Laufwerk nicht selektiert war. |
| $\overline{\text{SELECTOR}}$<br>(Zusatz) | Diese Leitung wird nur bei Hardsektorfunktion verwendet. Die 0,4 ms niedrigen Impulse treten in einem Abstand von 5,2 ms auf und markieren den Anfang der 32 Sektoren einer hardsektorierten Diskette. Bei Verwendung der 16/8-Sektor-Funktion wird der Impuls bei jedem zweiten Sektor (16 Sektoren) bzw. jedem vierten Sektor (8 Sektoren) erzeugt. |
| $\overline{\text{RAW DATA}}$ | Für jeden von der Diskette gelesenen Flußwechsel wird ein (niedriger) Impuls erzeugt.<br>Der so erzeugte Impulszug wird als Lesedaten zur Steuerung übertragen. Jeder Impuls ist 200 ns breit. |
| $\overline{\text{FM SEP DATA}}$<br>$\overline{\text{FM SEP CLOCK}}$<br>(Zusatz) | Diese Leitungen werden nur verwendet, wenn die Datentrennfunktion aktiviert ist.<br>Bei jedem Datenübergang bzw. Taktübergang im Rohdatenimpulszug wird ein 200 ns langer Impuls ($\overline{\text{FM SEP DATA}}$) bzw. ($\overline{\text{FM SEP CLOCK}}$) erzeugt. |
| $\overline{\text{ILLEGAL PACK}}$<br>(Zusatz) | Dieses Signal wird aktiv (niedrig), wenn eine einseitige Floppy eingelegt ist und der Kopf 1 ausgewählt wird. |
| $\overline{\text{TWO-SIDED}}$<br>(Zusatz) | Ist der Zusatz eingebaut, wird dieses Signal aktiv (niedrig), wenn eine zweiseitige Floppy eingelegt ist und die Köpfe geladen werden. Es wird inaktiv, wenn die Köpfe nicht geladen sind oder wenn bei geladenen Köpfen eine einseitige Floppy eingelegt ist. Dieser und der $\overline{\text{ILLEGAL PACK}}$-Zusatz schließen sich gegenseitig aus. |

Zwischen einer Schreiboperation und einer folgenden Leseoperation müssen 50 µs liegen.

# Anhang B

## Datentrennung (Data Separation)

Die Information, die auf einer Floppy Disk aufgezeichnet ist, besteht aus einer Mischung aus Daten- und Taktsignalen. Wird diese Information nun von der Diskette gelesen, muß ein Referenztakt (Lesetakt, RCLK) erzeugt werden. Mit Hilfe dieses Taktes kann der SAB 179X die Rohdaten (RAW DATA) in Takt- und Dateninformation trennen. Bild B.1 zeigt das vom Laufwerk kommende Datensignal RAW DATA und den zur Trennung benötigten Lesetakt RCLK. Das RCLK-Signal muß so erzeugt werden, daß die fallende Flanke des Rohdatenimpulses in die Mitte einer RCLK-Phase fällt. Dabei ist es ohne Bedeutung, ob es sich um den „1"- oder den „Ø"-Zustand handelt. Jede Phase des Taktes sollte etwa gleich lang sein (Zeiten siehe Datenblatt). Die vom Laufwerk kommenden Signale sind gegenüber ihrer Sollposition etwas verschoben. Dies liegt unter anderem an Schwankungen der Umdrehungsgeschwindigkeit und der begrenzten Flankensteilheit der Änderungen des magnetischen Flusses. Deshalb ist eine Rückkopplung bei der Erzeugung von RCLK notwendig. Bild B.2 zeigt das Prinzip.

**Bild B.1**
**Zeitverhältnisse zwischen den $\overline{\text{RAW DATA}}$-Signalen und dem notwendigen RCLK-Signal**

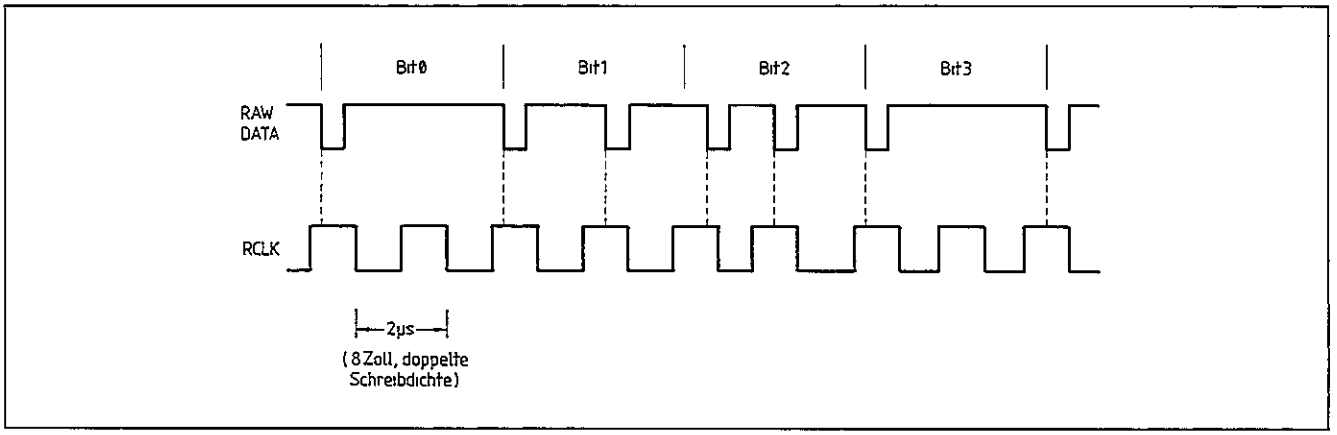
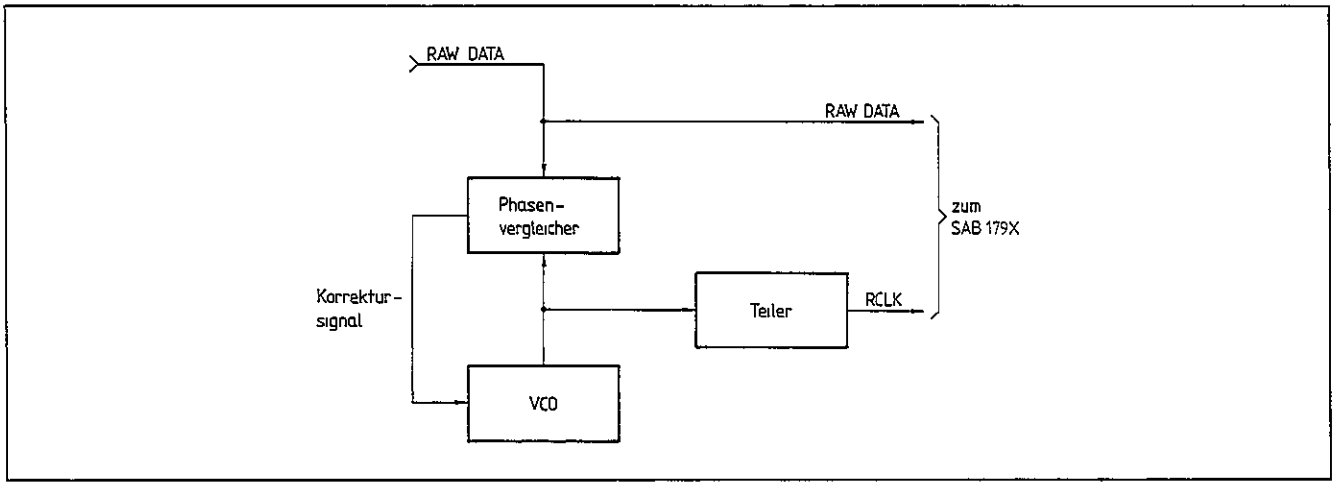
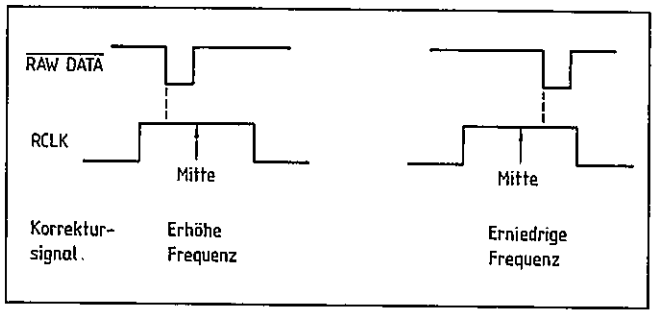



**Bild B.2**
**RCLK Erzeugung**



Bei jedem vom Laufwerk kommenden Impuls wird seine Lage in bezug auf die Phasenmitte des augenblicklichen RCLK-Signals überprüft. Liegt er links von der Mitte, muß ein Korrektursignal vom Phasenvergleicher erzeugt werden, das die RCLK-Frequenz etwas anhebt. Dadurch liegt der nächste Impuls wieder näher zur Mitte. Liegt er rechts von der Mitte, muß die Frequenz entsprechend erniedrigt werden (siehe Bild B.3). Das RCLK-Signal wird also durch den Phasenvergleicher in seiner Frequenz moduliert. Diese ständige Korrektur gewährleistet, daß der Datenimpuls immer innerhalb einer RCLK-Phase bleibt (auch Fenster genannt).

**Bild B.3**
**Regelung des RCLK-Taktes über VCO**




Allerdings dürfen die kurzzeitigen Veränderungen der Signalabstände nicht zu groß sein. Nur dann kann die PLL-Schaltung (phase locked loop) aus VCO (voltage controlled oscillator) und Phasenvergleicher schnell genug reagieren und die Datenimpulse innerhalb des Fensters halten. Die Abstandsschwankungen liegen bei den gebräuchlichen Laufwerken unter ±350 ns.

In Anhang G (Literaturhinweise, Pkt. 3 und 4) sind verschiedene Schaltungen für die Datentrennung angegeben.

# Anhang C

## Formatieren einer Floppy Disk

Das Formatieren einer Diskette ist das Beschreiben mit der Information uber die Aufteilung der einzelnen Spuren in Sektoren. Eine „soft-formatierte" Diskette kann erst nach der Formatierung fur normale Schreib- und Leseoperationen verwendet werden

Beim Formatieren muß jede Spur der Diskette innerhalb einer Umdrehung der Diskette geschrieben werden Dies beinhaltet neben den Datenfeldern die Lucken (Gaps), Adreßmarken, ID-Felder usw Wenn man sich eine Spur als lineare Folge von Bytes vorstellt, folgen die einzelnen Teile aufeinander, wie in Bild C 1 dargestellt.

Der physikalische Beginn wird durch den Indexpuls (IP) markiert (= Loch in der Diskette) Nach einer Lucke kommt die Index-Adreßmarke Sie wird von SAB 179X nicht unbedingt benotigt. Nach einer weiteren Lucke folgt die ID-Adreßmarke, die das ID-Feld kennzeichnet. Im ID-Feld werden im wesentlichen die Adresse und der Typ des Sektors angegeben. Im einzelnen sind das Spurnummer, Sektornummer, Seitennummer und kodiert die Sektorlange Unmittelbar daran schließt sich ein CRC-Feld an Nach einer Lucke und der Daten-Adreßmarke folgt das erste Datenfeld Dieses wird ebenfalls von einem CRC-Feld und einer anschließenden Lucke abgeschlossen Danach kommen die restlichen 25 Sektoren, jeweils aus dem ID-Feld und dem Datenfeld bestehend Nach dem 26. Sektor folgt wiederum eine Lücke, die bis zum Ende der Spur reicht Das Ende wird wie der Anfang durch den Indexpuls markiert

Bild C 2 zeigt die Realisierung der einzelnen Teile bei den IBM-Formaten 3740 (einfache Schreibdichte) und SYSTEM 34 (doppelte Schreibdichte). Die Bilder C 3 und C.4 zeigen die Bytes am Anfang der Spur 1ØH in beiden Formaten Hier kann man die genaue Folge der Bytes auf einer Spur leicht nachvollziehen

Um eine Spur zu formatieren, bereitet man die zu schreibenden Bytes am besten im Speicher auf Mit einem „Schreibe Spur"-Kommando können sie dann der Reihe nach ubertragen werden. Dabei muß die Information fur die ganze Spur im Speicher nicht vorhanden sein, da der großte Teil identisch ist Wie das Schreiben einer Spur ablauft, kann man dem entsprechenden Assemblerprogramm in Kapitel 7 entnehmen

**Bild C.1**
**Aufteilung einer Spur**

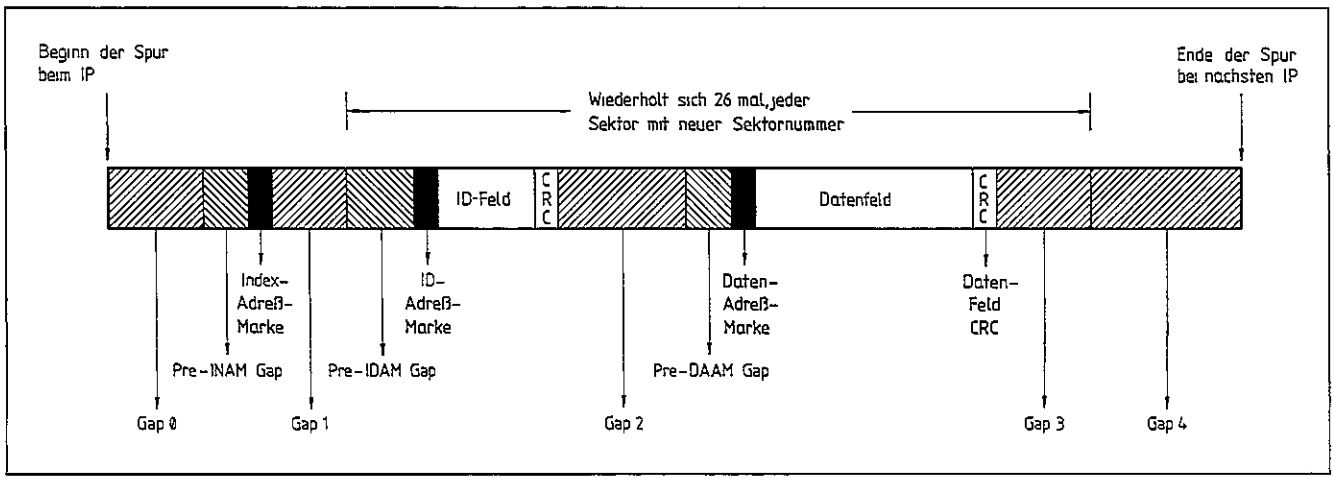
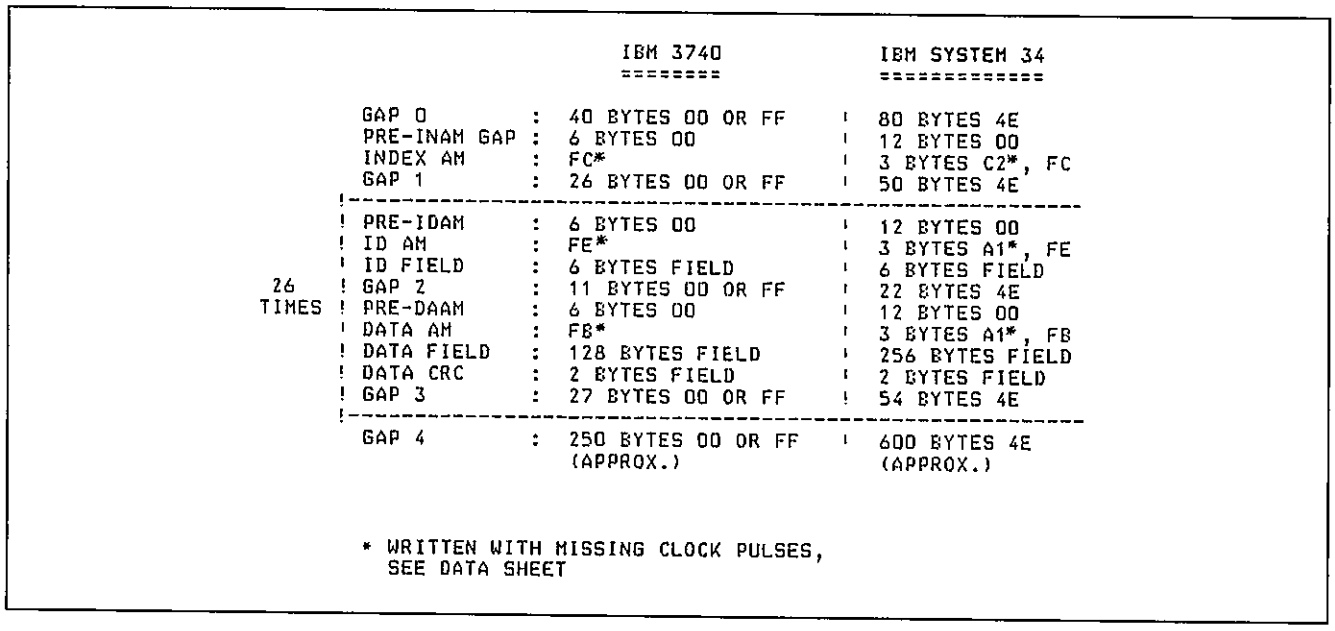


**Bild C.2**
**Spurformat für**
**IBM 3740 (einfache Schreibdichte) und**
**IBM SYSTEM 34 (doppelte Schreibdichte)**



| | | IBM 3740 | IBM SYSTEM 34 |
|---|---|---|---|
| | GAP 0 | 40 BYTES 00 OR FF | 80 BYTES 4E |
| | PRE-INAM GAP | 6 BYTES 00 | 12 BYTES 00 |
| | INDEX AM | FC* | 3 BYTES C2*, FC |
| | GAP 1 | 26 BYTES 00 OR FF | 50 BYTES 4E |
| 26 TIMES | PRE-IDAM | 6 BYTES 00 | 12 BYTES 00 |
| | ID AM | FE* | 3 BYTES A1*, FE |
| | ID FIELD | 6 BYTES FIELD | 6 BYTES FIELD |
| | GAP 2 | 11 BYTES 00 OR FF | 22 BYTES 4E |
| | PRE-DAAM | 6 BYTES 00 | 12 BYTES 00 |
| | DATA AM | FB* | 3 BYTES A1*, FB |
| | DATA FIELD | 128 BYTES FIELD | 256 BYTES FIELD |
| | DATA CRC | 2 BYTES FIELD | 2 BYTES FIELD |
| | GAP 3 | 27 BYTES 00 OR FF | 54 BYTES 4E |
| | GAP 4 | 250 BYTES 00 OR FF (APPROX.) | 600 BYTES 4E (APPROX.) |

* WRITTEN WITH MISSING CLOCK PULSES,
SEE DATA SHEET

**Bild C.3**
**IBM 3740 Spurformat (die ersten Bytes nach dem Indexpuls)**

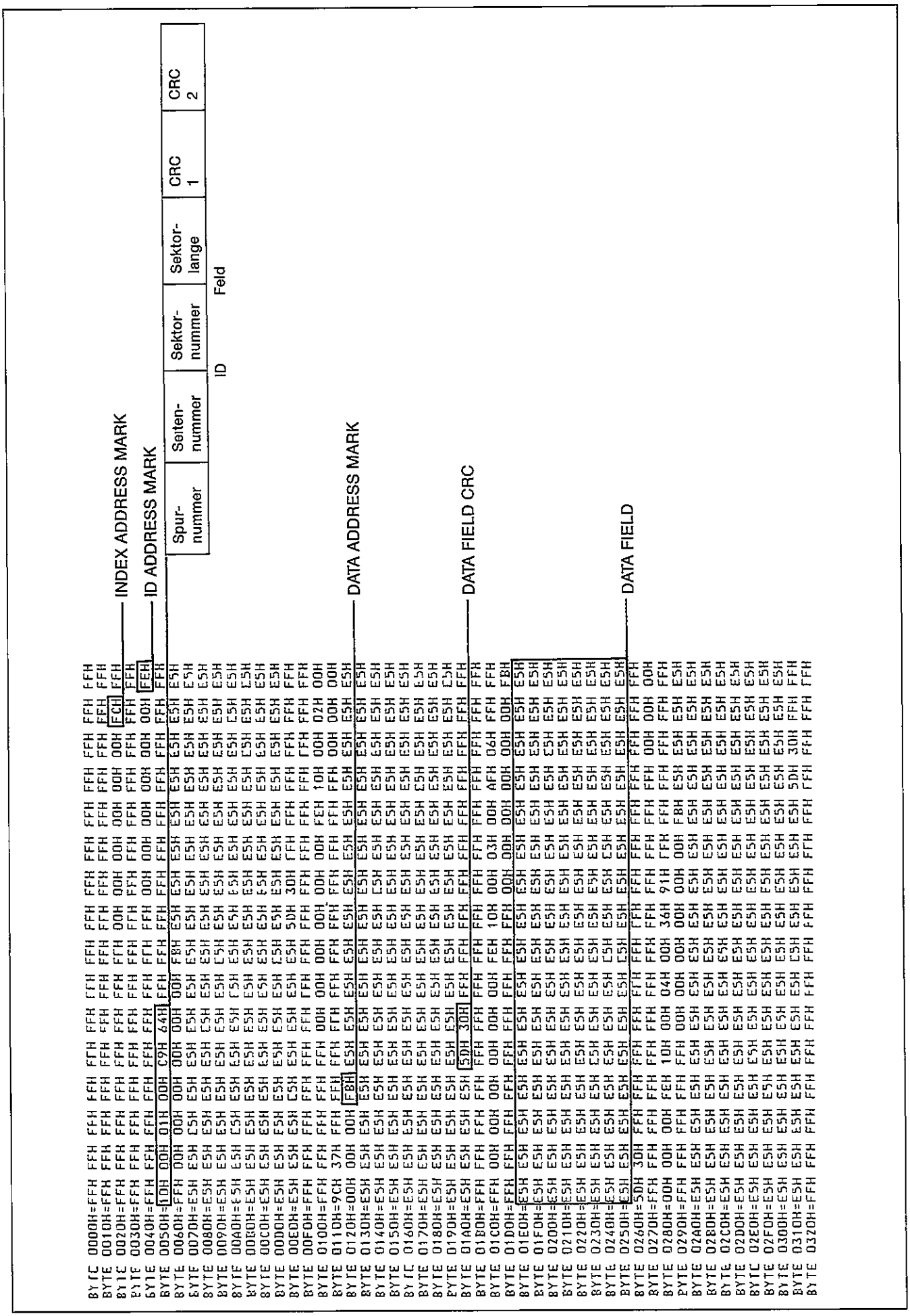


**Bild C.4**
**IBM SYSTEM 34 Spurformat (die ersten Bytes nach dem Indexpuls)**

```
BYTE 0000H=4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH
BYTE 0010H=4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH
BYTE 0020H=4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH
BYTE 0030H=4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH
BYTE 0040H=4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH
BYTE 0050H=4EH 4EH 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H C2H C2H
BYTE 0060H=C2H FCH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH
BYTE 0070H=4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH
BYTE 0080H=4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH
BYTE 0090H=4EH 4EH 4EH 4EH 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
BYTE 00A0H=A1H A1H A1H FEH 10H 00H 01H 01H E1H ABH 4EH 4EH 4EH 4EH 4EH 4EH
BYTE 00B0H=4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH
BYTE 00C0H=00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H A1H A1H A1H FBH
BYTE 00D0H=E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H
BYTE 00E0H=E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H
BYTE 00F0H=E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H
BYTE 0100H=E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H
BYTE 0110H=E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H
BYTE 0120H=E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H
BYTE 0130H=E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H
BYTE 0140H=E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H
BYTE 0150H=E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H
BYTE 0160H=E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H
BYTE 0170H=E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H
BYTE 0180H=E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H
BYTE 0190H=E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H
BYTE 01A0H=E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H
BYTE 01B0H=E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H
BYTE 01C0H=E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H
BYTE 01D0H=78H 27H 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH
BYTE 01E0H=4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH
BYTE 01F0H=4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH
BYTE 0200H=4EH 4EH 4EH 4EH 4EH 4EH 4EH 00H 00H 00H 00H 00H 00H 00H 00H 00H
BYTE 0210H=00H 00H 00H 00H A1H A1H A1H FEH 10H 00H 02H 01H B4H F8H 4EH 4EH
BYTE 0220H=4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH
BYTE 0230H=4EH 4EH 4EH 4EH 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
BYTE 0240H=A1H A1H A1H FBH E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H
BYTE 0250H=E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H
BYTE 0260H=E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H
BYTE 0270H=E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H
BYTE 0280H=E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H
BYTE 0290H=E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H
BYTE 02A0H=E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H
BYTE 02B0H=E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H
BYTE 02C0H=E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H
BYTE 02D0H=E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H
BYTE 02E0H=E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H
BYTE 02F0H=E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H
BYTE 0300H=E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H
BYTE 0310H=E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H
BYTE 0320H=E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H
BYTE 0330H=E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H E5H
BYTE 0340H=E5H E5H E5H E5H 78H 27H 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH
BYTE 0350H=4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH 4EH
```

INDEX ADDRESS MARK

ID ADDRESS MARK

DATA ADDRESS MARK

DATA FIELD CRC

DATA FIELD

| Spur-nummer | Seiten-nummer | Sektor-nummer | Sektor-länge | CRC 1 | CRC 2 |
|---|---|---|---|---|---|

ID Feld

# Anhang D

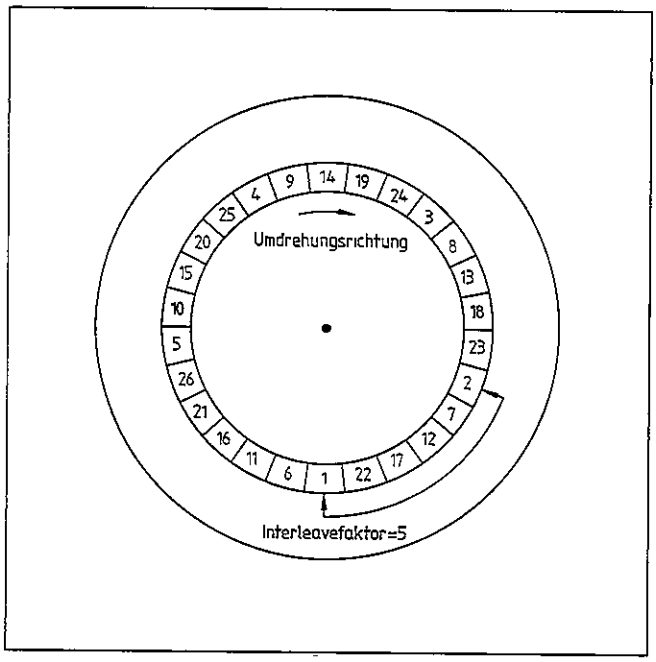
## Versetztes Aufzeichnen der Sektoren (Sector Interleaving)

Beim ersten (oder erneuten) Formatieren einer Diskette wird festgelegt, wo welcher Sektor innerhalb einer Spur zu liegen kommt. Dabei ist es nicht notwendig, daß sie in ihrer numerischen Reihenfolge in der Spur plaziert werden (z. B. 1, 2, 3 . 26). Tatsächlich ist es oft vorteilhafter, die Sektoren nicht sequentiell anzuordnen. Die sequentielle Reihenfolge ist dann optimal, wenn mehrere Sektoren (maximal eine Spur) ohne Unterbrechung während einer Umdrehung der Diskette übertragen werden können. Ein Beispiel dafür ist das Laden eines Programmes. Die Technik des versetzten Aufzeichnens von Sektoren („Sector Interleaving") optimiert die Zugriffszeiten, wenn die einzelnen Sektoren zwar nacheinander gelesen werden, zwischen dem Lesen/Schreiben aber eine kürzere Bearbeitungszeit notwendig ist. Ein Beispiel hierfür ist ein Texteditor. Hier kommt es vor, daß ein Sektor gelesen wird und der Inhalt mit einem bestimmten Suchkriterium bearbeitet wird. Erfüllt sich das Kriterium nicht, wird der nächste Sektor von der Diskette geholt. Da sich die Diskette aber weiterdreht, ist (bei sequentieller Aufzeichnung) der Anfang des nächsten Sektors schon am Schreib-/Lesekopf vorbei. Der Prozessor muß also fast eine ganze Umdrehung (ca. 166 ms) warten, bis er auf den nächsten Sektor zugreifen kann. Beim Interleaving folgt der nächste Sektor nicht unmittelbar anschließend, sondern es liegt eine gewisse Anzahl anderer Sektoren dazwischen (siehe Bild D 1). Diese Zahl wird als Interleavingfaktor bezeichnet. Auf diese Weise steht dem Prozessor Arbeitszeit zwischen dem Ende eines Sektors und dem Anfang des nächsten zur Verfügung. Bei einer Diskette mit 26 Sektoren/Spur (128 Bytes/Sektor) und einem Interleavingfaktor von 2 sind dies beispielsweise 6,4 ms.

Um den im Einzelfall notwendigen Interleavefaktor zu berechnen, muß man die Zeit, die der Prozessor zwischen den Sektorzugriffen benötigt, durch die Zeit teilen, die ein Sektor braucht, um den Schreib-/Lesekopf zu passieren. Mit Hilfe des Faktors kann nun die physikalische Reihenfolge bestimmt werden, mit der die Sektoren auf der Diskette aufgezeichnet werden (siehe Bild D 1). Beim Formatieren sind die Sektornummern in dieser Reihenfolge an den Steuerbaustein zu übergeben.

In Bild D 2 finden Sie ein Programm, das ein Feld mit dem Namen „sector sequence" mit der richtigen physikalischen Reihenfolge für einen gegebenen Interleavingfaktor I ausfüllt. Zusätzlich kann noch die Nummer des ersten Sektors in der Spur angegeben werden. In manchen Betriebssystemen wird sie in Abhängigkeit von der Spur verändert. Dadurch kann die Zugriffszeit bei einem Spurwechsel optimiert werden.

**Bild D.1**
**Sektorfolge bei versetzter Aufzeichnung (Interleaving)**



# Anhang F

## Softwaretreiber für ein System, bestehend aus: SAB 179X, SAB 8086 und SAB 8089

### F.1 Einleitung

In diesem Abschnitt wird ein vollständiger Softwaretreiber (FDC) für die SAB 179X-Steuerbausteine in einem SAB 8086/8089 System beschrieben. Das Blockschaltbild der verwendeten Hardware ist in Kapitel 5, Bild 5.3, zu finden. Der Softwaretreiber kann, in Verbindung mit der zugehörigen Steuerschaltung (siehe Bild 5.1), bis zu acht Laufwerke mit einfacher oder doppelter Schreibdichte in beliebiger Kombination bedienen.

Der grundsätzliche Aufbau dieses Softwaretreibers ist analog zu dem in Kapitel 7 beschriebenen. Die programmtechnische Lösung ist allerdings anders, da der SAB 8089 zur Verfügung steht. Er ist ein Ein-/Ausgabeprozessor, der u. a. zwei DMA-Kanäle besitzt. In dieser Applikation wird er speziell wegen dieser Eigenschaft eingesetzt, da er bis jetzt der einzige DMA-Baustein für den SAB 8086 ist.

### F.2 Beschreibung der Funktionen des Floppy-Disk-Treibers (FDC)

Der Softwaretreiber FDC stellt folgende Befehle bzw. Funktionen zur Verfügung:

0. Initialisieren
1. Restore (Kopf auf Spur Ø positionieren)
2. Kopf positionieren (Seek)
3. Lese Sektor(en) [Read Sector(s)]
4. Schreibe Sektor(en) [Write Sector(s)]
5. Lese Spur (Read Track)
6. Schreibe Spur (Write Track)
7. Diskette formatieren (Format Disk)
8. Verify (Diskette auf CRC-Fehler überprüfen)

Die notwendigen Parameter müssen in einem Datenfeld vom Anwender an den FDC übergeben werden.

### F.2.0 Initialisieren

Durch dieses Kommando wird der Floppy-Disk-Treiber initialisiert. Dabei werden die Bausteine (SAB 8089, SAB 8259A und SAB 8255A) und die Datenblöcke in einen Grundzustand versetzt. Dieser Befehl muß erteilt werden, bevor eine der übrigen Funktionen aufgerufen wird. Er darf allerdings nur einmal nach einem Hardwarereset ausgeführt werden (Einschränkung durch den SAB 8089). Dies ist auch das einzige Kommando, das ausgeführt werden kann, ohne daß eines der Laufwerke betriebsbereit ist. Bei jeder anderen Funktion muß das ausgewählte Laufwerk funktionsbereit sein.

### F.2.1 Restore

Der Schreib-/Lesekopf des ausgewählten Laufwerkes wird auf Spur Ø positioniert. Durch Lesen der Spurnummer im ID-Feld wird die korrekte Ausführung überprüft.

### F.2.2 Kopf positionieren (Seek)

Der Schreib-/Lesekopf des ausgewählten Laufwerkes wird auf die gewünschte Spur positioniert. Durch Lesen der Spurnummer wird die Position überprüft.

### F.2.3 Lese Sektor(en)

Der Inhalt der gewünschten Zahl von Sektoren auf der angegebenen Spur wird in einen Stack (Pufferspeicher) übertragen.

Die Befehlsausführung beginnt mit einem impliziten Positionieren des Kopfes auf die gewünschte Spur. Danach werden die spezifizierten Sektoren ausgelesen und die Daten im Speicher abgelegt. Die Spurnummer, die Nummer des ersten zu lesenden Sektors, die Anzahl der Sektoren sowie die Anfangsadresse des Pufferspeichers müssen vom Anwender im Parameterblock übergeben werden (siehe nächsten Abschnitt).

### F.2.4 Schreibe Sektor(en)

Der Ablauf und die notwendigen Parameter sind analog zum Lesen von Sektoren. Die Daten aus dem Stack werden in die Sektoren geschrieben.

### F.2.5 Lese Spur

Sämtliche Bytes der angegebenen Spur werden gelesen und in den Puffer übertragen. Allerdings kann es vorkommen, daß nicht alle Gap- und Adreßmarkenbytes richtig übertragen werden, da die Synchronisation jeweils mit den Adreßmarken erfolgt. Dadurch ist aber sichergestellt, daß die Inhalte der ID- und Datenfelder korrekt wiedergegeben werden.

Dieses Kommando ist für Diagnosezwecke gedacht, nicht jedoch um im Normalfall Daten von der Floppy Disk zu lesen. Auch bei diesem Befehl wird der Kopf automatisch positioniert.

### F.2.6 Schreibe Spur

Dieses Kommando formatiert die gewünschte Spur im IBM 3740- (einfache Schreibdichte) bzw. IBM-System 34-(doppelte Schreibdichte)Format. Die Gaps werden im ersten Fall mit FFH, im zweiten mit 4EH beschrieben. In die Datenfelder wird E5 geschrieben. Bei diesem Befehl kann außer der zu formatierenden Spur noch die Nummer des physikalischen ersten Sektors sowie der Interleavefaktor angegeben werden.

Vor Ausführung der Schreiboperation wird automatisch auf die richtige Spur positioniert. Die fehlerfreie Übertragung der Daten wird durch eine nachfolgende Verify-Operation überprüft.

## F.2.7 Diskette formatieren

Dieses Kommando beschreibt alle Spuren der Diskette (siehe Abschnitt 2 7) Vor dem Beginn der Formatierung wird der Kopf auf Spur Ø positioniert Auch bei diesem Kommando kann man die Nummer der ersten Spur und den Interleavefaktor angeben Diese Einstellungen gelten dann fur alle Spuren. (Soll die Diskette mit von Spur zu Spur unterschiedlichen Anfangssektornummern und/ oder Interleavefaktoren formatiert werden, muß man jede Spur einzeln formatieren )

Nach Beendigung der Formatierung wird der Kopf wieder auf Spur Ø positioniert

## F.2.8 Verify

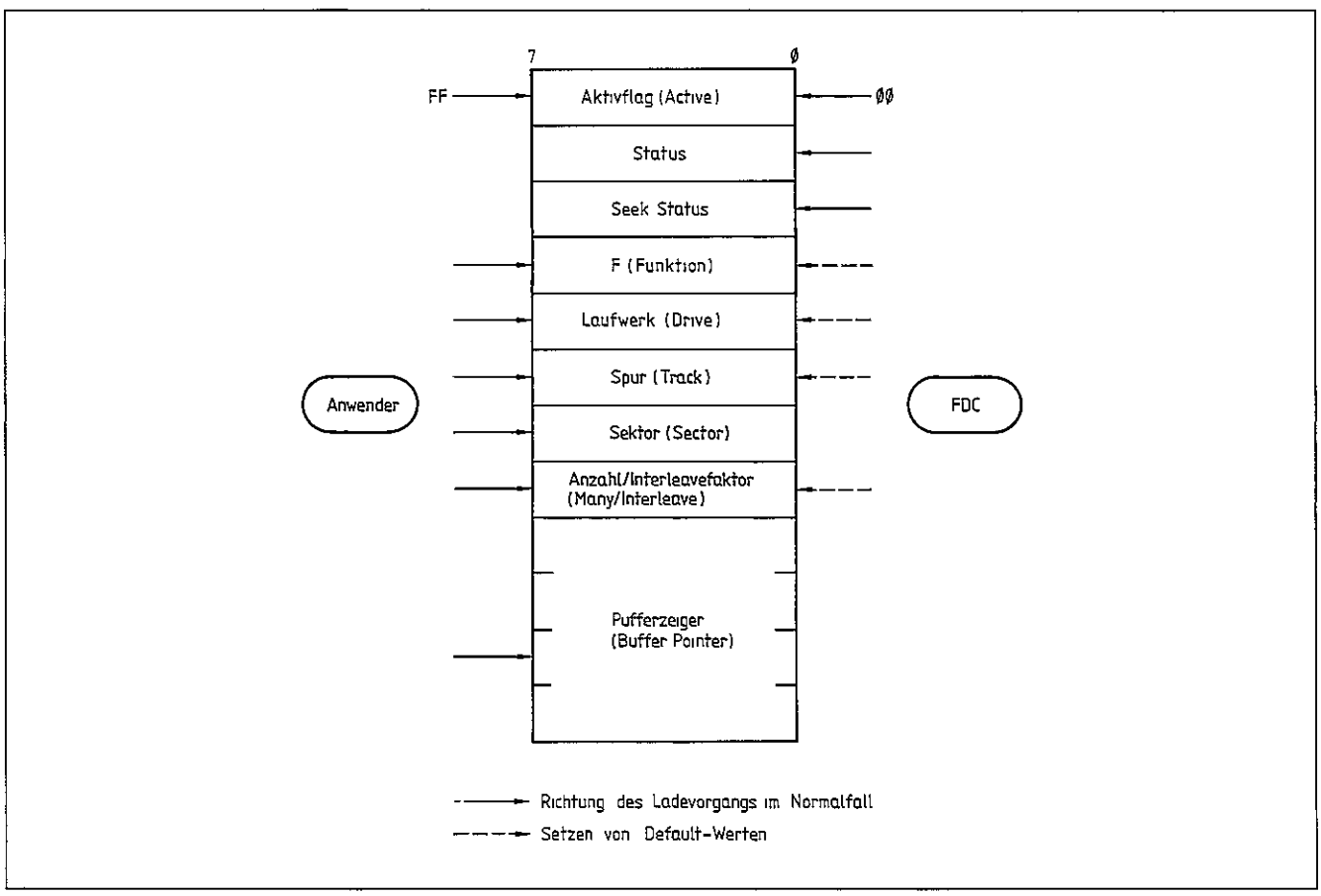
Mit diesem Kommando kann man die Diskette auf CRC-Fehler uberprufen Es wird jeder Sektor von jeder Spur gelesen und dabei auf CRC-Fehler getestet Dies ist die einfachste und schnellste Moglichkeit, um festzustellen, ob die Diskette in Ordnung ist oder nicht Die Uberprufung beginnt mit Spur Ø Nach der Kommandoausfuhrung wird wieder auf Spur Ø positioniert

Jedes dieser Kommandos benotigt bestimmte Parameter, die der Anwender ubergeben muß Nach der Befehlsausführung erhalt er Statusmeldungen und andere Daten vom FDC zuruck. Die notwendige Kommunikation wird über einen Speicherbereich abgewickelt, der im folgenden als Datenblock bezeichnet wird.

## F.3 Die Schnittstelle zwischen Anwender und FDC

Der Anwender muß die fur die jeweilige Funktion notwendigen Parameter in einem Datenblock (DBL) ubergeben. Er besteht aus 12 Byte, der Aufbau ist in Bild F 1 gezeigt. Nachdem der Anwender den Datenblock geladen hat, kann er den Softwaretreiber FDC aufrufen, wobei er die Adresse des Datenblocks als Parameter ubergeben muß. FDC kopiert dann den DBL in einen internen Kommunikationsblock CBL, der in seinem Aufbau genau dem DBL entspricht (siehe Bild F 2) Bei der weiteren Kommandobearbeitung werden nur noch die Werte in CBL verwendet Nach der Durchfuhrung der Funktion wird CBL nach DBL kopiert und das Aktivflag in DBL zuruckgesetzt. Danach wird die Kontrolle an den Anwender zuruckgegeben. Das Aktivflag zeigt dem Anwenderprogramm, wann der FDC die Bearbeitung des ubergebenen Kommandos beendet hat Dazu muß der Anwender das Flag vor dem Aufruf setzen (ØFFH) Durch diese Kommunikationssteuerung ist es moglich, daß mehrere Datenblocke gleichzeitig vorhanden sind (z B einer fur jedes Laufwerk) Es darf jeweils nur ein DBL aktiviert sein Dadurch kann das Anwenderprogramm auch in einer Multitasking-Umgebung mit Hilfe des Aktivflags feststellen, welcher DBL gerade bearbeitet wird

**Bild F.1**
**Aufbau des Datenblockes (DBL) zur Kommunikation zwischen Anwender und FDC**



## F.4 Programmbeschreibung

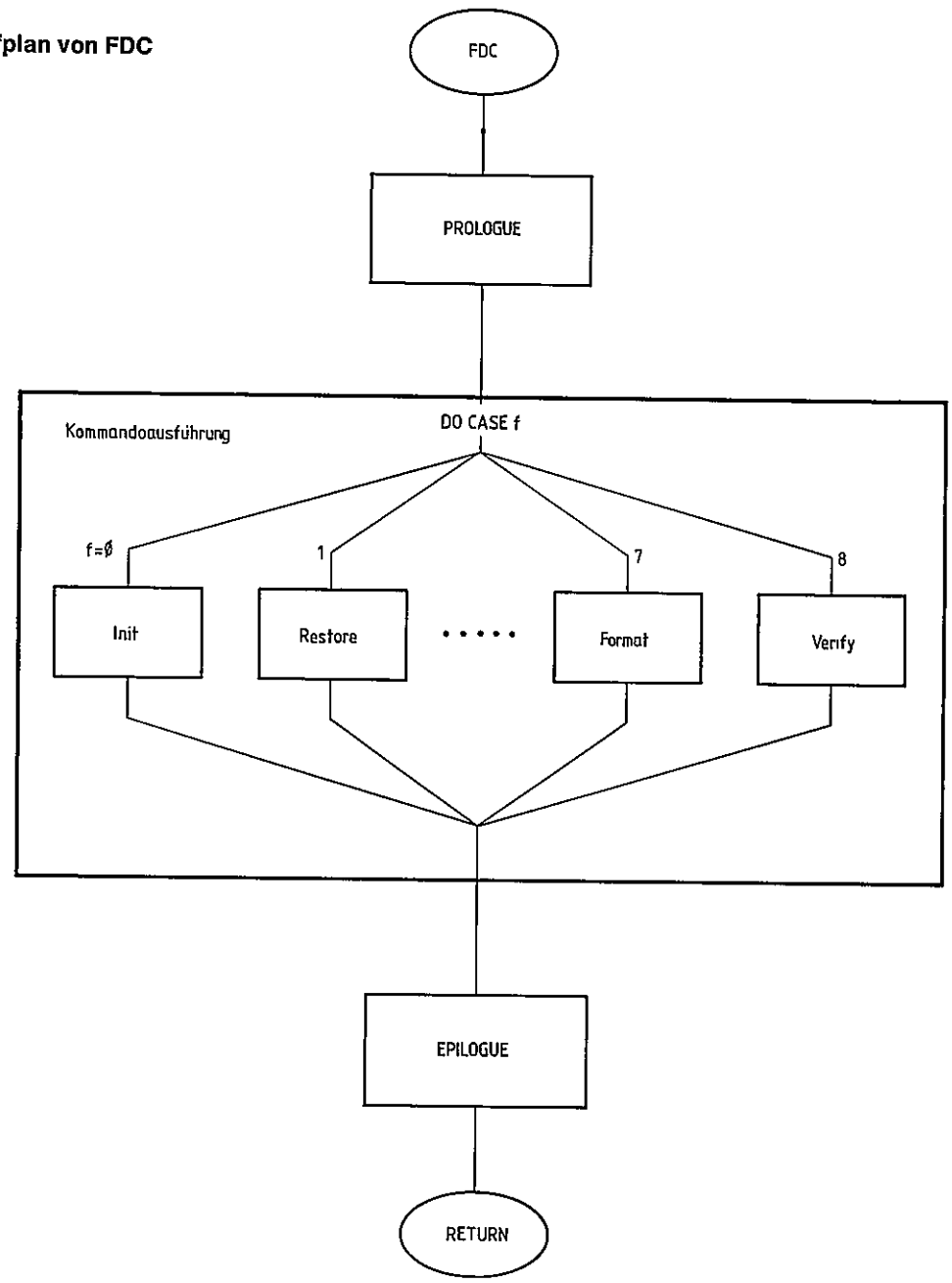
Das Floppy-Disk-Treiberprogramm (FDC) ist nach dem Schichtenmodell von Bild F.3 aufgebaut. Die oberste Ebene läuft auf dem SAB 8086 ab und ist in PL/M-86 geschrieben. In ihr sind ca. 90 % des Softwaretreibers realisiert. Die Kommunikation zwischen Anwender und FDC erfolgt, wie in Kapitel F.3 beschrieben, über den Datenblock DBL. Die mittlere Ebene stellt die Verbindung zum SAB 179X (innerste Ebene) her. Sie ist mit dem Ein-/Ausgabeprozessor SAB 8089 realisiert. Die Kommunikation zwischen der SAB 8086-Ebene und der SAB 8089-Ebene erfolgt über zwei Datenblöcke und zwei Hardwaresignale. Die Datenblöcke sind der Kanal-Kommando-Block CCB und der Parameterblock PB, die Signale sind Kanalaufruf CA (Channel Attention) und Interrupt INT. Die Kommandos und Parameter, die der SAB 8089 benötigt, werden in CCB und PB geschrieben und die Programmausführung des SAB 8089 über CA gestartet. Hat der SAB 8089 alle gewünschten Aktionen beendet, meldet er dies über das INT-Signal an den SAB 179X.

Das Verbindungsglied zwischen dem SAB 8089 und SAB 179X sind die Register des Floppy-Disk-Steuerbausteines sowie die Signale Datenanforderung (DRQ) und Bedienanforderung (IRQ).

Nachdem der SAB 8086 den SAB 8089 aktiviert hat, führt er das SLEEP-Programm aus. In dem vorliegenden Beispiel für einen Softwaretreiber ist diese Routine ein Dummyprogramm, das auf den Interrupt des SAB 8089 wartet. Das Interruptprogramm setzt ein Flag, das im SLEEP-Programm ständig abgefragt wird. Der SAB 8086 fährt dann mit der Ausführung des Hauptprogrammes des Softwaretreibers fort. Dieser Mechanismus kann in einem echten Multitaskingbetriebssystem verwendet werden, um die Task FDC in einen Wartezustand zu versetzen.

Im folgenden sollen nicht alle Einzelheiten eines jeden Programms besprochen werden. Es werden der Grobablauf und die wichtigsten verwendeten Algorithmen erläutert, um das Verständnis der Listings zu erleichtern.

**Bild F.4**
**Grobablaufplan von FDC**



Der Treiber FDC laßt sich in drei große Blocke gliedern (siehe Bild F 4)

- den Prolog
- die Kommandoausfuhrung und
- den Epilog

Der Beginn des FDC-Hauptteils (= Beginn des Prologes ist Anweisung 252). Im Prolog werden vor jedem Kommando folgende Aktionen durchgeführt

- Kopieren von DBL nach CBL
- Kopieren von DBL nach CBL
- Art des Laufwerkes und die Nummer des Laufwerkes werden aus CBL entnommen, wenn notwendig korrigiert und das entsprechende Laufwerk selektiert.
- Der Parameter „Anzahl" wird uberpruft und wenn notwendig korrigiert
- Das Spurregister des SAB 179X wird mit der Spurnummer geladen, auf der der Kopf des selektierten Laufwerks steht Diese Information wird zusatzlich in einem Feld von 8 Bytes „current$track" gespeichert
- Es wird uberpruft, ob das selektierte Laufwerk betriebsbereit ist.
- Es wird uberpruft, ob die Nummer der gewunschten Funktion F = 8 ist. Wenn nicht, wird sie auf 1 gesetzt (= Restore)

Nun wird das gewunschte Kommando ausgefuhrt. Dies geschieht in einem DO CASE-Block mit der Funktionsnummer F (= cbl f) als Auswahlvariable

Nach der Kommandoausfuhrung werden im Epilog noch folgende Aktionen ausgefuhrt

- Das Spurregister des SAB 179X wird im „current$track"-Feld abgespeichert.
- Das Aktivflag in CBL wird zurückgesetzt.
- CBL wird nach DBL kopiert.
- Alle Laufwerke werden deselektiert.

Der oben erwähnte DO CASE-Block hat 9 Falle (Nummer Ø bis 8) Jeder Fall besteht aus einem oder mehreren Unterprogrammaufrufen

Als Beispiel wollen wir den Fall 3, das Lesen eines Sektors, betrachten (Die Ausfuhrung dieses Kommandos beginnt bei Anweisung 125 )

Zuerst wird der Schreib-/Lesekopf auf die gewunschte Spur positioniert. Anschließend wird die Nummer des ersten zu lesenden Sektors in das Sektorregister des SAB 179X geladen (= Parameter cbl sector) Da die eigentliche Leseoperation vom SAB 8089 gesteuert wird, mussen ihm die notwendigen Daten über den Parameterblock (PBL) geliefert werden Im einzelnen sind dies: der Pufferzeiger fur Datenubergaben, die Konfiguration des Kommandobytes für den SAB 179X (fur Sektor lesen), die Anzahl der zu ubertragenden Bytes und ein Byte, das den SAB 8089 uber die auszufuhrende Funktion informiert Danach wird das Signal „Kanalaufruf" (CA) aktiviert (Anweisung 132) und damit der SAB 8089 gestartet Der SAB 8086 wartet nun auf den Interrupt vom SAB 8089 (SLEEP-Programm), der den Abschluß der Operation meldet. Jetzt kann die Statusinformation des SAB 179X eingelesen werden und damit die Leseoperation abgeschlossen werden

Die übrigen Funktionsprogramme sind analog dazu aufgebaut. So unterscheidet sich das „Schreibe-Sektor-Kommando" vom Lesekommando nur durch die an den SAB 8089 ubergebenen Kommandobytes (fur SAB 179X und SAB 8089)

Fur den SAB 8089 sind zwei Programme vorhanden: IOPTASK1 und TRACKTASK Das zweite wird beim Formatieren einer Spur (Schreiben einer ganzen Spur) und der Diskette benotigt Alle ubrigen Funktionen werden mit IOPTASK1 durchgefuhrt

Das Programm IOPTASK1 beginnt in Zeile 59 (SAB 8089 – Teil des Listings) Die Register werden mit den entsprechenden Werten des Parameterblockes PBL geladen und dann das notwendige Kommando an den SAB 179X ausgegeben. Danach wird der SAB 8089 in den DMA-Ubertragungszustand versetzt (= nach Ausführung der Anweisung in Zeile 69). Diesen Zustand kann er durch zwei Ereignisse verlassen

1. Der SAB 179X aktiviert die IRQ-Leitung (Beendigung des Kommandos oder bei einem Fehler) In diesem Fall fuhrt der SAB 8089 die Anweisung in Zeile 71 aus
2. Der Bytezahler fur die DMA-Ubertragung erreicht Ø (moglich beim Lesen oder Schreiben mehrerer Sektoren) In diesem Fall fahrt der SAB 8089 in Zeile 73 mit der Programmausfuhrung fort Hier wird, nach einer Verzogerung, die Kommandoausfuhrung des SAB 179X durch einen „Force Interrupt"-Befehl abgebrochen.

In beiden Fallen wartet der SAB 8089 bis das Busybit in der Statusanzeige des SAB 179X zuruckgesetzt ist Anschließend wird der Interrupt für den SAB 8086 generiert.

Das Programm IOPTASK1 wird auch bei Befehlen wie RESTORE und SEEK benutzt, die eigentlich keine DMA-Ubertragung benotigen In diesen Fallen ist der SAB 8089 im DMA-Ubertragungszustand inaktiv und wartet auf den Interrupt vom SAB 179X

Zusatzlich zu den Treiberprogrammen ist noch ein Dummy-Anwenderprogramm vorhanden Hier soll verdeutlicht werden, wie der Datenblock DBL vereinbart werden kann (Anweisung 2), wie man den Floppy-Disk-Treiber FDC aufruft (Anweisung 12) und wie er als externe Prozedur vereinbart wird (Anweisungen 4, 5 und 6)

```
                /* communication block containing data for current job. */
                /* it is a copy of dbl, received from the user. */

 32   1         declare cbl structure(

                    active      byte,
                    status      byte,
                    seek$status byte,
                    f           byte,

                    drive       byte,
                    track       byte,
                    sector      byte,
                    many        byte,

                    buff$ptr    pointer);   /* 12 byte block */

                /*-------------------------------------------------------------*/

                /* SAB 8089 communication data structure */

 33   1         declare scb structure (     /* system configuration block */
                    conf    byte,
                    null    byte,
                    ccwp    pointer) at (0ff0h);

 34   1         declare ccb structure (     /* channel command block */
                    ccw1    byte,
                    busy1   byte,
                    pbp1    pointer,
                    null1   word,
                    ccw2    byte,
                    busy2   byte,
                    pbp2    pointer,
                    null2   word);

 35   1         declare pb1 structure (     /* parameter block for iop$task1 */
                    tp      pointer,
                    mem     pointer,
                    comm    byte,
                    null    byte,
                    byt$cnt word,
                    cmd     word);

 36   1         declare pb2 structure (     /* parameter block for track$task */
                    tp                  pointer,
                    track$buff          pointer,
                    gap1                pointer,
                    sector              pointer,
                    bc1                 word,
                    bc2                 word,
                    sectors(no$sect)    byte);

                /*-------------------------------------------------------------*/
                $eject
```

```
37   1      declare track$buff(470) byte;          /* for writing track format */
38   1      declare current$track(no$drives) byte;  /* current head position for 8 drives */
39   1      declare drive   byte,
                   density byte,
                   intr$flag byte,

             /*-------- IBM 3740 track format ----------*/

40   1      declare sd$track$form structure(
                 g0(40) byte,            /* ff */
                 pinam(6) byte,          /* 00 */
                 inam byte,              /* fc */
                 g1(27) byte,            /* ff */
                 pidam(6) byte,          /* 00 */
                 idam byte,              /* fe */
                 track byte,             /* 00 */
                 side byte,              /* 00 */
                 sector byte,            /* 01 */
                 seclen byte,            /* 00 */
                 idcrc byte,             /* f7 */
                 g2(11) byte,            /* ff */
                 pdtam(6) byte,          /* 00 */
                 dtam byte,              /* fb */
                 dat(128) byte,          /* e5 */
                 dtcrc byte,             /* f7 */
                 g3(2) byte)             /* ff */
                 at (@track$buff);

             /*-------- IBM SYSTEM 34 track format ---------*/

41   1      declare dd$track$form structure(
                 g0(80) byte,    /* 4e */
                 pg0(12) byte,   /* 00 */
                 pinam(3) byte,  /* f6 */
                 inam byte,      /* fc */
                 g1(54) byte,    /* 4e */
                 pg1(12) byte,   /* 00 */
                 pidam(3) byte,  /* f5 */
                 idam byte,      /* fe */
                 track byte,     /* 00 */
                 side byte,      /* 00 */
                 sector byte,    /* 01 */
                 seclen byte,    /* 01 */
                 idcrc byte,     /* f7 */
                 g2(22) byte,    /* 4e */
                 pg2(12) byte,   /* 00 */
                 pdtam(3) byte,  /* f5 */
                 dtam byte,      /* fb */
                 dat(256) byte,  /* e5 */
                 dtcrc byte,     /* f7 */
                 g3(2) byte)     /* 4e */
                 at (@track$buff);

             /*-----------------------------------------------------*/
             $eject
```

```
42   1      int8089: procedure interrupt 22h;
            /* invoked on interrupt from SAB 8089 which, in general,
               signals completion of a SAB 179X operarion  */
43   2          intr$flag = 1;                  /* wake up fdc */
44   2          return;
45   2      end;
            /*================================================*/

46   1      fdc: procedure(dbl$ptr) public;
47   2      declare dbl$ptr pointer;
            /*------------------------------------------------*/

48   2      sleep: procedure;

            /* for the want of something better to do, fdc sleeps
               here  till an interrupt from SAB 8089 comes and wakes it. */

49   4          do while intr$flag = 0; end; /* wait for interrupt */
51   3          intr$flag=0;        /* intr$flag reset for subsequent use */
52   3          return;

53   3      end;
            /*------------------------------------------------*/

54   2      interleave: procedure;

            /* fills in a field of 26 bytes (pb.sectors) with a sequence of numbers
               from 1 to 26. first entry is the first sector number, specified by
               user in the dbl.sector field and the subsequent numbers satisfy the
               interleave factor specified in the dbl.many field.    */

55   3          declare strt$sec$nr literally 'cbl.sector'; /* starting sector number */
56   3          declare intlev literally 'cbl.many';          /* inter-leave factor */
57   3          declare sec$mark(no$sect) byte,act$sec$nr byte, tbl$id byte, s byte;

58   3          call setb(0,@sec$mark,no$sect);
59   3          tbl$id = 0;
60   3              act$sec$nr,pb2.sectors(tbl$id)=strt$sec$nr;
61   3          sec$mark(tbl$id)=1;
62   3          do s = 2 to no$sect;
63   4           tbl$id = ((tbl$id + intlev) mod no$sect);  /* next sector no. location */
64   4               act$sec$nr = act$sec$nr +1;            /* next sector number */
65   4               if act$sec$nr > no$sect then act$sec$nr = act$sec$nr mod no$sect;
67   4          do while sec$mark(tbl$id) <> 0;         /* check if location free */
68   5           tbl$id =  (tbl$id +1) mod no$sect;     /* if not, offset by 1    */
69   5          end;

70   4          sec$mark(tbl$id) = 1;               /* declare new location as not free */
71   4          pb2.sectors(tbl$id) = act$sec$nr;   /* write next sector no. in location */
72   4          end;
73   3          return;
74   3      end;
            /*------------------------------------------------*/
            $eject
```

```
 75   2      init$sd$track: procedure;

             /* initialize ibm 3740 track$format */

 76   3          call setb(0ffh,@sd$track$form,96);
 77   3          call setb(0,@sd$track$form.pinam(0),6);
 78   3          call setb(0,@sd$track$form pidam(0),6);
 79   3          call setb(0,@sd$track$form.pdtam(0),6);
 80   3          call setb(0,@sd$track$form.track,4);
 81   3          call setb(0e5h,@sd$track$form.dat(0),128);
 82   3          call setb(0ffh,@sd$track$form.g3(0),3);
 83   3          sd$track$form.inam = 0fch;
 84   3          sd$track$form.idam = 0feh;
 85   3          sd$track$form.dtam = 0fbh;
 86   3          sd$track$form.idcrc,sd$track$form.dtcrc = 0f7h;
 87   3          return;
 88   3      end,

             /*-------------------------------------------------------*/
 89   2      init$dd$track: procedure;

             /* initialize IBM SYSTEM 34 track$format */

 90   3          call setb(04eh,@dd$track$form,468);
 91   3          call setb(0f6h,@dd$track$form.pinam(0),3);
 92   3          call setb(0f5h,@dd$track$form.pidam(0),3);
 93   3          call setb(0f5h,@dd$track$form.pdtam(0),3);
 94   3          call setb(0e5h,@dd$track$form.dat(0),256);
 95   3          call setb(0,@dd$track$form.pg0,12);
 96   3          call setb(0,@dd$track$form.pg1,12);
 97   3          call setb(0,@dd$track$form.pg2,12);
 98   3          dd$track$form.inam = 0fch;
 99   3          dd$track$form.idam = 0feh;
100   3          dd$track$form.dtam = 0fbh;
101   3          dd$track$form.track = 0;
102   3          dd$track$form.side = 0;
103   3          dd$track$form.seclen = 1;
104   3          dd$track$form.idcrc,dd$track$form.dtcrc = 0f7h;
105   3          return;
106   3      end;

             /*-------------------------------------------------------*/
             $eject
```



```
107   2      restore: procedure;

108   3          pb1.comm = res$comm;
109   3          pb1.cmd = step$seek$cmd;
110   3          output(addr$8089) = 0;  /* start iop channel 1 */
111   3          call sleep;

112   3          cb1.seek$status=input(stat$179x);
113   3          cb1.track=input(track$179x);
114   3          return;
115   3      end;
             /*-------------------------------------------------------*/
116   2      seek:   procedure;

117   3          output(data$179x)=cb1.track;

118   3          pb1.comm = seek$comm;
119   3          pb1.cmd = step$seek$cmd;
120   3          output(addr$8089) = 0;  /* start iop channel 1 */
121   3          call sleep;

122   3          cb1.seek$status=input(stat$179x);
123   3          return;
124   3      end,
             /*-------------------------------------------------------*/
             $eject
```

```
125   2     rd$sector: procedure;

126   3         call seek;

127   3         output(sector$179x)=cb1.sector;

128   3         pb1.mem = cb1.buff$ptr;
129   3         pb1.comm = r$s$comm;
130   3         pb1.byt$cnt = sector$length * cb1.many;
131   3         pb1.cmd = read$sector$cmd;
132   3         output(addr$8089) = 0;  /* start iop channel 1 */
133   3         call sleep;

134   3         cb1.status=input(stat$179x);
135   3         return;
136   3     end;
          /*-------------------------------------------------------------*/
137   2     wr$sector: procedure;

138   3         call seek;

139   3         output(sector$179x)=cb1.sector;

140   3         pb1.mem = cb1.buff$ptr;
141   3         pb1.comm = w$s$comm;
142   3         pb1.byt$cnt = sector$length * cb1.many;
143   3         pb1.cmd = write$sector$cmd;
144   3         output(addr$8089) = 0;  /* start iop channel 1 */
145   3         call sleep;

146   3         cb1.status=input(stat$179x);

147   3         return;
148   3     end;
          /*-------------------------------------------------------------*/
          $eject
```



```
149   2     rd$track: procedure;

150   3         call seek;

151   3         pb1.mem = cb1.buff$ptr;
152   3         pb1.comm = r$t$comm;
153   3         pb1.byt$cnt = 5000h;
154   3         pb1.cmd = read$track$cmd;
155   3         output(addr$8089) = 0;  /* start iop channel 1 */
156   3         call sleep;

157   3         cb1.status=input(stat$179x);

158   3         return;
159   3     end;
          /*-------------------------------------------------------------*/

160   2     verifytrack: procedure;

161   3         output(sector$179x) = 1;
162   3         pb1.mem = @track$buff(468);
163   3         pb1.comm = r$s$comm;
164   3         pb1.byt$cnt = no$sect * sector$length;
165   3         pb1.cmd = verify$track$cmd;
166   3         output(addr$8089) = 0;  /* start iop channel 1 */
167   3         call sleep;
168   3         return;
169   3     end verifytrack;
          /*-------------------------------------------------------------*/
          $eject
```

```
PL/M-86 COMPILER    ** Floppy Disk Controller - SAB 179X - Software Driver **


170   2      wr$track: procedure;

171   3          do case density;
172   4           do;
173   5              sd$track$form.track=cb1.track;
174   5              pb2.gap1 = @sd$track$form.g1(0);
175   5              pb2.sector = @sd$track$form.sector;
176   5              pb2.bc1 = 233;
177   5              pb2.bc2 = 186;
178   5           end;
179   4           do;
180   5              dd$track$form.track = cb1.track;
181   5              pb2.gap1 = @dd$track$form.g1(0);
182   5              pb2.sector = @dd$track$form.sector;
183   5              pb2.bc1 = 466;
184   5              pb2.bc2 = 370;
185   5           end;
186   4          end;

187   3          ccb.pbp1 = @pb2;        /* switch to parameter block 2 */
188   3          output(addr$8089) = 0;  /* start iop channel 1 */
189   3          call sleep;

190   3          ccb.pbp1 = @pb1;        /* switch back to parameter block 1 */
191   3          cb1.status=input(stat$179x) or cb1.status;

192   3          call verifytrack;       /* auto-verify */
193   3          cb1.status=input(stat$179x) or cb1.status;

194   3          return;
195   3      end;
          /*-------------------------------------------------------------*/
          $eject
```

```
PL/M-86 COMPILER    ** Floppy Disk Controller - SAB 179X - Software Driver **


196   2      format$disk: procedure;

197   3          call    restore;

198   3          cb1.status = 0;

199   3          do cb1.track=0 to 76;
200   4          call wr$track;

201   4          pb1.comm = stepin$comm;     /* step in by 1 track */
202   4          pb1.cmd = step$seek$cmd;
203   4          output(addr$8089) = 0;  /* start iop channel 1 */
204   4          call sleep;
205   4          end;

206   3          call    restore;

207   3          return;
208   3      end;
          /*-------------------------------------------------------------*/
209   2      verify$disk: procedure;

210   3          call    restore;

211   3          cb1.status=0;

212   3          do cb1.track=0 to 76;
213   4          call verifytrack;
214   4          cb1.status = input(stat$179x) or cb1.status;

215   4          pb1.comm = stepin$comm;     /* step in by 1 track */
216   4          pb1.cmd = step$seek$cmd;
217   4          output(addr$8089) = 0;  /* start iop channel 1 */
218   4          call sleep;
219   4          end;

220   3          call    restore;

221   3          return;

222   3      end;
          /*-------------------------------------------------------------*/
          $eject
```

```
223   2       init:   procedure;

224   3           call set$interrupt(22h,int8089);

                /* initialize 8255a */

225   3           output(addr$8255+6)=init$ports;
226   3           output(addr$8255)=0ffh;

                /* initialize 8259a */

227   3           output(addr$8259)=13h;
228   3           output(addr$8259+2)=20h;
229   3           output(addr$8259+2)=3h;
230   3           output(mask$8259)=11111011b;

                /* initialize 8089  */

231   3           scb.ccwp=@ccb;
232   3           scb.conf=1;
233   3           ccb.busy1,ccb.busy2=0ffh;
234   3           ccb.ccw1=00h;
235   3           ccb.pbp1 = @pb1;
236   3           pb1.tp = @iop$task1;
237   3           pb2.tp = @track$task;
238   3           pb2.track$buff = @track$buff;

239   3           output(addr$8089+2)=0;  /* first channel attention */
240   3           call time(1);
241   3           do while ccb.busy1 <> 0; /* wait till iop is initialized */
242   4           end;

243   3           ccb.ccw1=10h;       /* enable 8089 interrupt */
244   3           output(addr$8089)=0;    /* channel attention */

                /* initialize parameters */

245   3           call setb(0,@current$track,8);
246   3           intr$flag=0;

247   3           ccb.ccw1 = start$ch;
248   3           ccb.busy1 = 0;

249   3           enable;         /* enable SAB 8086 interrupt */
250   3           return;

251   3       end;

                /*============================================================*/
                $eject
```

```
                /* entry point to fdc */

 252    2       begin$fdc:

                    call movb(dbl$ptr,@cbl,12); /* copy dbl to cbl */

 253    2           cbl.drive = cbl.drive and 00010111b;/* auto-parameter-correction */
 254    2           drive = cbl.drive and 7;    /* values between 0 and 7 */
 255    2           density = shr(cbl.drive,4); /* density = 0 (Single), = 1 (Double) */

 256    2           if cbl.many = 0 then cbl.many = 1;  /* auto-parameter-correction */
 258    2           if cbl.many > no$sect then cbl.many = no$sect;

                    /* select drive and density
                        8255 port for single density = 1111ddd0
                                for double density = 0111ddd0, ddd = drive no. */
 260    2           do case density;
 261    3           output(addr$8255) = shl (not(drive),1);     /* single density */
 262    3           output(addr$8255) = shl (not(drive),1) and 7fh; /* double density */
 263    3           end;

 264    2           output(track$179x) = current$track(drive);
 265    2           call time(10);
 266    2           if (cbl.f * (input(stat$179x) and 10000000b)) = 0 then
                    /* if drive ready or f = 0 */

 267    2            do;
 268    3             if cbl.f > max$f then cbl.f = 1;  /* auto-parameter-correction */
 270    3             do case cbl.f;
 271    4       c0:    call init;
 272    4       c1:    call restore;
 273    4       c2:    call seek;
 274    4       c3:    call rd$sector;
 275    4       c4:    call wr$sector;
 276    4       c5:    call rd$track;
 277    4       c6:    do;
 278    5               cbl.status=0;
 279    5               do case density,
 280    6                 call init$sd$track;
 281    6                 call init$dd$track;
 282    6               end;
 283    5              call interleave;
 284    5              call seek;
 285    5              call wr$track;
 286    5              end;
 287    4       c7:    do;
 288    5               do case density;
 289    6                 call init$sd$track;
 290    6                 call init$dd$track;
 291    6               end;
 292    5              call interleave;
 293    5              call format$disk;
 294    5              end;
 295    4       c8:    call verifydisk;
 296    4             end;
 297    3             if cbl.f <> 0 then current$track(drive) = input(track$179x);
```

```
PL/M-86 COMPILER    ** Floppy Disk Controller - SAB 179X - Software Driver **


 299   3           end;
 300   2           else cbl.seek$status = input(stat$179x);

 301   2          cbl.active = 0;
 302   2          call movb(@cbl,dbl$ptr,12);      /* copy cbl to dbl */
 303   2          output(addr$8255)=11111111b;     /* de-select drives */

 304   2          return;

 305   2      end fdc;
 306   1      end;


MODULE INFORMATION:

     CODE AREA SIZE     = 0666H   1638D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 0256H    598D
     MAXIMUM STACK SIZE = 001EH     30D
     627 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION
```

```
8089 MACRO ASSEMBLER     '* Floppy Disk Controller -SAB 179X- Software Driver *'


ISIS-II 8089 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE FDC62
OBJECT MODULE PLACED IN :F1:FDC62.OBJ
ASSEMBLER INVOKED BY:  asm89 :f1:fdc62.a89


LOC    OBJECT CODE             TIMING    INC MAC  LINE SOURCE

                                                    1 $debug
                                                    2 $title('* Floppy Disk Controller -SAB 179X- Software Driver *')
                                                    3
                                                    4 ; Sharad Gandhi, Mikrocomputer-Systemberatung, Siemens, Munich
                                                    5
                                                    6 ;*************************************************************
                                                    7 ;
                                                    8 ;       name    fdcio
                                                    9 ;
                                                   10 ;8089 part of the software driver contains routines for doing
                                                   11 ;data transfers and also for doing dummy dma while waiting
                                                   12 ;for interrupt from SAB 179X.
                                                   13 ;It consists of two procedures:
                                                   14 ;          1. Tracktask - for writing a new track(format)
                                                   15 ;          2. Ioptask1  - for all other functions
                                                   16 ;
                                                   17 ;-------------------------------------------------------------
                                                   18
0000                                               19 iop     segment
                                                   20
                                                   21         public  ioptask1
                                                   22         public  tracktask
                                                   23
                                                   24
                                                   25
  FFF0                                             26 s179x   equ     0fff0h          ;sab 179x status reg. address
  FFF0                                             27 c179x   equ     0fff0h          ;sab,179x command reg. address
  FFF6                                             28 d179x   equ     0fff6h.         ;sab 179x data reg. address
  1020                                             29 dummy_dma equ   1020h           ;wait for intr. on ext pin
  00D0                                             30 force_intr equ 0d0h
  00F0                                             31 write_track_comm equ    0f0h
                                                   32
                                                   33 ;*************************************************************
                                        +1         34 $eject
```

```
8089 MACRO ASSEMBLER     '* Floppy Disk Controller -SAB 179X- Software Driver *'

LOC    OBJECT CODE              TIMING    INC MAC  LINE SOURCE

                                                    35 ;
                                                    36 ;IOPTASK1 - is used for dma transfers for the operations:
                                                    37 ;                1. Read sector(s)
                                                    38 ;                2. Write sectors(s)
                                                    39 ;                3. Read track
                                                    40 ;                4. Verify track
                                                    41 ;
                                                    42 ;It is also used, doing a dummy DMA, while waiting for an
                                                    43 ;interrupt form SAB 179X for the operations:
                                                    44 ;                1. Restore
                                                    45 ;                2. Seek
                                                    46 ;These are essentially busy waits.
                                                    47 ;
                                                    48 ;-------------------------------------------------------------
                                                    49
0000                                                50 pb1      struc
0000                                                51 tp1:     ds      4
0004                                                52 mem:     ds      4
0008                                                53 comm:    ds      1
0009                                                54 null:    ds      1
000A                                                55 bytcnt:  ds      2
000C                                                56 cmd:     ds      2
000E                                                57 pb1      ends
                                                    58
0000                                                59 ioptask1:
0000   038B 04                    34    46         60          lpd     ga,[pp1.mem     ;memory pointer in ga
0003   6383 0A                    53    76         61          mov     bc,[pp1.bytcnt  ;byte count
0006   C383 0C                    75   106         62          mov     cc,[pp1.cmd     ;dma command
                                                    63
0009   3130 F6FF                  93   131         64          movi    gb,d179x        ;179x data register address
000D   5130 F0FF                 111   156         65          movi    gc,c179x        ;179x command register address
0011   0293 08 00CE              151   206         66          movb    [gc],[pp1.comm  ;command to sab 179x
                                                    67
0016   6000                     162   224         68          xfer
0018   8000                     173   242         69          wid     8,8             ;start dma with bus widths = 8
                                                    70
001A   9120 0C00                 190   267         71 ext_term: ljmp   finis           ;interrupt received from SAB 179X
                                                    72
001E   B130 1400                 207   292         73 bc_end:  movi    ix,20           ;delay before force interrupt
0022   A03C                     217   309         74 delay:   dec     ix
0024   A840 FB                  236   332         75          jnz     ix,delay
0027   084E D0                  259   362         76          movbi   [gc],force_intr ;stop multiple sector command
                                                    77
002A   5130 F0FF                 276   387         78 finis:   movi    gc,s179x        ; status reg.
002E   08BE FD                   304   419         79 lab0:    jbt     [gc],0,lab0     ; wait till SAB 179X not busy
0031   4000                     319   437         80          sintr                   ; generate interrupt
0033   2048                     341   462         81          hlt
                                                    82
                                                    83 ;*************************************************************
                                          +1        84 $eject
```

```
8089 MACRO ASSEMBLER      * Floppy Disk Controller -SAB 179X- Software Driver *'

LOC    OBJECT CODE            TIMING    INC MAC  LINE SOURCE

                                                    85
                                                    86 ;TRACKTASK is used to write a complete track on the disk.
                                                    87
                                                    88 ;-------------------------------------------------------------
                                                    89
                                                    90 pb2      struc
0000                                                91 tp2:             ds      4
0004                                                92 trackbuff:       ds      4
0008                                                93 gap1:            ds      4
000C                                                94 sector:          ds      4
0010                                                95 bc1:             ds      2
0012                                                96 bc2:             ds      2
0014                                                97 sectors:         ds      26
002E                                                98 pb2      ends
                                                    99
0035   8820 00                                     100 even
                                                   101
0038                                               102 tracktask:
0038   B130 1400                358   487         103         movi    ix,sectors      ; offset to sector secquence
003C   F130 1A00                375   512         104         movi    mc,26           ; number of sectors
0040   038B 04                  409   558         105         lpd     ga,[pp].trackbuff       ; start of track data
0043   6383 10                  428   588         106         mov     bc,[pp].bc1     ; first byte count
0046   438B 0C                  462   634         107         lpd     gc,[pp].sector  ; address of id sector byte
0049   D130 0850                480   659         108         movi    cc,5008h        ; dma with bc terminate
004D   3130 F0FF                498   684         109         movi    gb,c179x
0051   8000                     513   702         110         wid     8,8
0053   084D F0                  536   732         111         movbi   [gb],write_track_comm
0056   3130 F6FF                553   757         112         movi    gb,d179x
                                                   113
005A   6000                     564   775         114 lab1:   xfer
005C   0693 00CE                596   821         115         movb    [gc],[pp+ix+]   ; next sector number loaded
                                                   116
0060   038B 08                  630   867         117         lpd     ga,[pp].gap1    ; start address for next dma
0063   6383 12                  649   897         118         mov     bc,[pp].bc2     ; byte count
0066   E03C                     659   914         119         dec     mc              ; decrement sector count
0068   E840 EF                  678   937         120         jnz     mc,lab1         ; jump if all sectors not done
                                                   121
006B   D130 2010                696   962         122         movi    cc,1020h        ; mem -> port, without inc.
                                                   123
006F   6000                     711   980         124         xfer
0071   7130 0010                729  1005         125         movi    bc,1000h        ; dma till intr. from sab 179x
                                                   126
0075   5130 F0FF                747  1030         127         movi    gc,a179x        ; status reg.
0079   08BE FD                  772* 1062         128 lab2:   jbt     [gc],0,lab2     ; wait till SAB 179X not busy
007C   4000                     783  1080         129         sintr
007E   2048                     801  1105         130         hlt
                                                   131
0080                                               132 iap     ends
                                                   133
                                                   134         end


ASSEMBLY COMPLETE, NO ERRORS FOUND
```

# Anhang G

## Literaturhinweise


1. SAB 179X – Data Sheet
2. Western Digital 1981 Product Handbook
3. 179X Application Note
4. 1771 Application Note
5. Beschreibung des Floppy-Disk-Laufwerkes Siemens FDD 200-8
6. iAPX 86, 88 User's Manual, July 1981

# Mikrocomputer-Beratung

**Ansprechpartner in allen Mikrocomputerfragen – Technik, Lieferbedingungen, MC-Schule – finden Sie unter folgenden Rufnummern:**

## Bundesrepublik Deutschland und Berlin (West)

Siemens AG
Salzufer 6–8
**1000 Berlin 10**
☎ (030) 3939-2694
-2384
-3268

Siemens AG
Schweriner Straße 1
**4800 Bielefeld 1**
☎ (0521) 291-207
-205
-204

Siemens AG
Contrescarpe 72
**2800 Bremen**
☎ (0421) 364-2251
-2255

Siemens AG
Lahnweg 10
**4000 Düsseldorf 1**
☎ (0211) 399-690
-540
-932

Siemens AG
Rödelheimer Landstraße 5–9
**6000 Frankfurt 1**
☎ (0611) 797-2715
-3376
-2882

Siemens AG
Habsburgerstraße 132
**7800 Freiburg**
☎ (0761) 2712-348
-307
-305

Siemens AG
Lindenplatz 2
**2000 Hamburg 1**
☎ (040) 282-2861

Siemens AG
Am Maschpark 1
**3000 Hannover 1**
☎ (0511) 199-2741
-2740
-2256

Siemens AG
N 7, 18 (Siemenshaus)
**6800 Mannheim 1**
☎ (0621) 296-312
-218

Siemens AG
Richard-Strauß-Straße 76
**8000 München**
☎ (089) 9221-2552
-2647

Siemens AG
Von-der-Tann-Straße 30
**8500 Nürnberg 1**
☎ (0911) 654-3421
-3803
-3313

Siemens AG
Geschwister-Scholl-Str. 24
**7000 Stuttgart 1**
☎ (0711) 2076-742
-362
-771

Siemens AG
Nicolaus-Otto-Straße 4
**7900 Ulm**
☎ (0731) 499-252
-251

## Europa

**Niederlande**
Siemens Nederland N.V.
Wilhelmina van
Pruisenweg 26
**NL-2595 AN Den Haag**
☎ (070) 782697

**Österreich**
Siemens AG
Göllnergasse 15
**A-1030 Wien**
☎ (0222) 7293-5883
-5448

**Schweiz**
Siemens AG
Freilagerstraße 28
**CH-8047 Zürich**
☎ (01) 495-4185
-4314

---


**Für Kunden in der Bundesrepublik Deutschland**

Mit diesem Stempel möchten wir Ihre Aufmerksamkeit auf den Siemens Bauteile Service – SBS – lenken, der mehr als 12.000 Schwerpunkttypen an Elektronik-Bauelementen ständig für Sie versandbereit hält. Die Preis- und Lagerliste erhalten Sie kostenlos auf Anruf vom Siemens Bauteile Service (siehe Stempel) oder von der

SIEMENS AG
ZVW 85
Postfach 1500
8510 Fürth-Bislohe

**Für Kunden im Ausland**

dienen als Bezugsquellen die Bauteile-Vertriebe unserer Landesgesellschaften oder Vertretungen.

**Herausgegeben von Siemens AG, Bereich Bauelemente, Produkt-Information, Balanstraße 73, D-8000 München 80**



Fragen über Technik, Preise und Liefermöglichkeiten richten Sie bitte an unsere Zweigniederlassungen im Inland, Abteilung VB oder an unsere Landesgesellschaften im Ausland

2.83

Siemens Aktiengesellschaft

Bestell-Nr. B/2677
Printed in Germany
PA 03832.