PROFIBUCH

Jankowski / Rabich / Reschke

SYBEX

ATARI Profibuch ST-STE-TT

- Das Standardwerk in der 10., überarbeiteten Auflage
- Geballtes Insiderwissen auf über 1500 Seiten

SYBEX

ATARI Profibuch

Hans-Dieter Jankowski
Dietmar Rabich
Julian F. Reschke

SYBEX
Computerbücher & Software

DÜSSELDORF · SAN FRANCISCO · PARIS · SOEST (NL)

Titel der deutschen Original-Ausgabe: "ATARI ST Profibuch"
Original Copyright © 1987 by SYBEX-Verlag GmbH, Düsseldorf

Überarbeitung: Hans-Dieter Jankowski, Dietmar Rabich, Julian F. Reschke

Satz: text korrekt · Carola Richardt, Essen
Redaktion: Doris Heinzmann
Belichtung: Softype Computersatz-Service GmbH, Düsseldorf
Umschlaggestaltung: Lippert, Wilkens & Partner GmbH, Düsseldorf
Farbreproduktionen: Lettern Partners, Düsseldorf
Druck und buchbinderische Verarbeitung: Bercker, Kevelaer

ISBN 3-88745-888-5
10. Auflage 1991 (4. überarbeitete und erweiterte Ausgabe)
11. Auflage 1992

Inhaltsverzeichnis

Vorwort

Über vier Jahre sind vergangen, seitdem die erste Auflage des Profibuchs erschienen ist. In der Zwischenzeit hat sich vieles bei Hardware (MEGA ST, STE, MEGA STE, TT, Stacy, SCSI-Festplatten) und Software (neue TOS-Versionen, XCONTROL) getan. Auch über viele betriebssysteminterne Zusammenhänge wissen wir heute mehr als damals. Alle diese neuen Informationen haben wir in diese komplette Überarbeitung einfließen lassen.

Wie Sie sehen, haben wir dabei nunmehr das maximale Fassungsvermögen für ein einzelnes Buch erreicht. Mit künftigen Weiterentwicklungen – ob beim Betriebssystem (Multitasking?) oder bei der Hardware (ST-Book, 68040-Rechner) – werden wir uns daher voraussichtlich zu einem späteren Zeitpunkt in einem Ergänzungsband befassen.

Im vorliegenden Band sind wir unserem Motto treu geblieben, Fakten möglichst vollständig, aktuell, übersichtlich, präzise und preisgünstig zu präsentieren. Wir hoffen, daß Sie mit dem Ergebnis zufrieden sind.

Münster, im Oktober 1991

Hans-Dieter Jankowski, Dietmar Rabich und Julian Reschke

Elektronische Mail kann an folgende Adressen gerichtet werden:

Hardware: Hans-Dieter_Jankowski@un.maus.de

Software: Dietmar_Rabich@do.maus.de
Julian_Reschke@ms.maus.de

Hans-Dieter Jankowski

Nach nicht unerheblichem Arbeitsaufwand ist es endlich geschafft! Die Hardware-Kapitel zum ATARI ST/STE/TT Profibuch sind endlich fertig. Nach Erscheinen der ersten Auflage des ATARI ST Profibuchs hat sich ja doch einiges auf dem Hardware-Sektor bei Atari getan. So ist denn auch einiges an Informationen zusammengekommen, das hier niedergelegt wurde. Wie immer hat "unsereiner" sich mal wieder bezüglich des dazu erforderlichen Zeitaufwandes um ca. 300 Prozent verschätzt, aber dennoch: Es ist vollbracht!

"Unsereiner" ist jetzt Mitte 30, heißt Hans-Dieter Jankowski und ist von Beruf immer noch Ingenieur für Nachrichtentechnik. Während des Studiums in den Jahren 1974 bis 1977 ist

"unsereiner" kaum von der Datenverarbeitung "berührt" worden (eher abgeschreckt durch die Praktika an der DV-Anlage in unserer Hochschule, welche mit Lochkarten und in "PROSA 300" – ist auch eine Programmiersprache, wenn ich mich noch recht erinnere – programmiert wurde).

Nach dem Einstieg mit programmierbaren Taschenrechnern um 1979, mit ersten Erfolgserlebnissen in Form von selbsterstellten Programmen, erfolgte ein Wechsel auf einen "echten" Homecomputer mit Z80-CPU, 16 KByte RAM und Kassettenrekorder als "Massenspeicher" (vielleicht erinnnert sich ja der eine oder andere Leser noch an das "Video Genie"-System). Dann mußte Grafik und am besten auch Farbe her. Das führte zur Anschaffung eines "DAI"-Systems (8080 CPU, 48 KByte RAM und gute Farbgrafik-Eigenschaften), aber leider auch in eine Sackgasse. Es gab nur sehr wenige Anwender und kaum Software!

Aber dann (um 1983) entdeckte "unsereiner" den ATARI 800XL und in diesem Zusammenhang auch Julian Reschke (nein, nein, ich habe Julian Reschke nicht *entdeckt*, sondern lediglich kennen- und schätzengelernt!). Julian arbeitete ebenfalls auf einem 8-Bit-Atari und hatte schon bald Erfolge als Fachautor. Also warum nicht selbst einen Versuch starten; Ideen waren genug da. Zuerst kamen kleine Software-Tips, dann Utilities, die auch von den Verlagen angenommen wurden. Später kamen noch Erfahrungsberichte über Software dazu.

Aber letztlich kann keiner gegen seine Natur und "unsereiner" nicht gegen seine Leidenschaft für die Hardware und damit das "Innenleben" der Ataris. Drucker- und sonstige Interfaces wurden gebaut und auch ein Cartridge-Experimentiersystem für die 8-Bit-ATARIs entwickelt (erschien mal im ATARI-Sonderheft von "Happy Computer"). 1986 kam dann der "Umstieg" auf den ersten 16-Bit-Computer (520ST+) von Atari. Begeistert von der Bedienungsfreundlichkeit (erster Gedanke: Ein tolles Ding, diese Maus; zweiter Gedanke: Wie die wohl von innen aussieht?) und der großartigen Bildschirmdarstellung (erster Gedanke: Das Bild flimmert ja gar nicht; zweiter Gedanke: Da müßte man doch mal das Videosignal untersuchen), verschlang "unsereiner" alles, was an Informationen zu bekommen war. Besonders die Hardware interessierte natürlich, aber da waren die Infos doch sehr spärlich. Letztlich führte diese Leidenschaft dann dazu, die Kenntnisse über die Hardware der ST-Computer in einem Buch mit konzentrierten Infos über die Betriebssystemsoftware zusammenzufassen, und so wurde das ATARI ST Profibuch (Idee: Julian Reschke) "geboren".

Nachdem Atari nun im Herbst '90 mit der neuen Rechnergeneration TT auf dem Markt erschien, wurde natürlich auch eine Überarbeitung des "Profibuchs" erforderlich. Wobei wir aber ganz schnell zu dem Ergebnis kamen, daß es mit einigen weiteren Anhängen nicht getan sein würde. Also mußte ein komplettes, neues Buch her, mit natürlich bereits im ST Profibuch enthaltenen Infos, aber eben auch einer ganzen Menge an neuen Daten über den TT/STE. (Wenn wir noch lange warten, müssen wir den Umfang des Buches noch weiter "aufblasen", um auch den STBook und STPad mit aufzunehmen, ganz zu schweigen von Gerüchten um

ein Multitasking-TOS! Das führt dann letztlich dazu, daß dieses Buch nie erscheint, weil immer was Neues dazukommt.)

Nachdem der TT dann endlich für Entwickler lieferbar war (also wurde ich schnell "Entwickler", denn schließlich sollte man so ein Gerät, über dessen Innenleben man zu schreiben gedenkt, schon besitzen!), wurde auf dem schnellsten Wege ein TT geordert. Dank Herrn Henseleits Einsatz (Dank, Dank, Dank!) von Atari-Deutschland kam "mein" Exemplar auch *sehr* schnell. Karton aufgemacht, und "Überraschung!": Es war scheinbar ein Bausatz, denn alle Komponenten (ST-RAM-Erweiterung, FAST-RAM usw.) waren separat geliefert worden. Der nette Kommentar von Herrn Henseleit dazu: "Sie wollen ja das "Ding" sowieso auseinandernehmen! Wir hatten es hier im Dauertest und haben es, da Sie es ja schnell haben wollten, sofort zerlegt versendet."

Aber nun genug der Historie und Philosophiererei! Ein besonders großes Dankeschön geht natürlich noch an meine Frau Marianne. Sie hat es nicht immer leicht gehabt, mich aus dem Innenleben der Rechner und Peripherie und dem Gewirr von Meßleitungen von Oszilloskop und Logikanalyzer (Originalton: "Mein Gott, hoffentlich kriegst du den (gemeint war der zerlegte ST) noch einmal wieder zusammen!") herauszulösen. Verbunden ist damit auch eine Bitte um Entschuldigung an sie und den Rest der Familie (inzwischen drei prächtige Ausgaben "selbstgemachter Hardware" männlichen Geschlechts, acht Monate, acht und zehn Jahre alt und mit gesunder "natürlicher Intelligenz" ausgestattet. Der Jüngste heißt übrigens auch Julian, mal sehen, ob es was nützt!) für die viele Zeit, die ich statt mit ihnen an den Computern (vor allen Dingen bei der Erstellung des Buches) verbracht habe. Nachdem das Manuskript des Hardware-Teils nun fertig ist, verspreche ich, mich mehr um sie (die Familie) zu kümmern (hm, da war doch noch was von wegen Harddisktreiber usw...).

Ein weiteres Dankeschön an Herrn Henseleit und Herrn Jurkat von Atari-Deutschland, die für mich immer ein offenes Ohr und reichlich Informationen hatten.

Dietmar Rabich

Als im Jahre 1962 eine Computergeneration begann, erblickte auch ich das Licht der Welt. Nach den vielen Jahren der Schule und einem Studium der Mathematik und Physik gehe ich seit 1989 als Diplom-Mathematiker dem Beruf eines Programmierers in einem großen Konzern in Dortmund nach.

Der erste Kontakt mit einem Computer ergab sich im neunten Schuljahr. Auf einem Wang 2200 mit 4 (vier) Kilobyte Hauptspeicher, einer Schreibmaschine als Ausgabemedium und einem eingebauten Kassettenrecorder als Massenspeicher wurde eifrig mit der ganzen Klasse BASIC gelernt. Und, wie sollte es anders sein, gerade dieser Rechner übte einen so starken Reiz auf mich aus, daß ich mich nicht mehr davon lösen konnte.

Dem Wang folgte daheim ein vergleichsweise komfortabler Rechner: ein CBM 3032 mit 32 KByte RAM und zwei Diskettenlaufwerken. Auch hier hieß die Stammprogrammiersprache BASIC.

1983 kamen dann mit dem Studium andere Programmiersprachen hinzu (Pascal, ForTran 77, ForTran 200, PL/I, ...). Die Kenntnisse konnten jedoch nicht auf den heimischen Rechner übertragen werden, also war eine Vertiefung der Kenntnisse nicht möglich.

In der Erwachsenenbildung war ich regelmäßig als Kursleiter tätig. Unterrichtet wurde fast ausschließlich BASIC auf Apple IIe- oder Toshiba 1500-Rechnern. Von 1989 an bot ich nur noch kleine Kurse an.

Das Jahr 1986 brachte mir endlich einen komfortablen, preisgünstigen Rechner: den Atari ST! Nach einer kurzen Eingewöhnungsphase – die Maus war bis dahin ein unbekanntes Peripheriegerät gewesen – versuchte ich, meinen ersten ST (einen 520 ST+) erst noch in BASIC zu programmieren, was ich aber wegen der bekannten Qualitäten des damaligen BASIC schnell aufgab... Ich wechselte zu dem CCD-Pascal-Compiler. Der alte CBM geriet mit der Begeisterung für den ST schnell in Vergessenheit.

Da neben dem BASIC noch ein LOGO ausgeliefert wurde, ergab sich die Gelegenheit, auch noch diese Programmiersprache zu erlernen. Doch trotz überaus interessanter Details konnte sich LOGO keinen Platz als Leib- und Magenprogrammiersprache ergattern. Der Interpreter war doch zu wenig leistungsfähig.

Die Jahre gingen ins Land, der 520 ST+ wich einem MEGA ST2 mit Harddisk, und diesen ergänzt mittlerweile ein TT030/6. Pascal mußte Modula-2 weichen und Modula-2 wieder C, so daß ich heute fast nur noch in C programmiere und entwickle. C zeigte sich als zuverlässig, mit allen Eigenschaften einer modernen Programmiersprache und guter Code-Erzeugung ausgerüstet.

Über die Jahre erschienen auch einige Programme: die Mathematik-Bibliothek MATHLIB bei Creative Computer Design, das Disk-Utility und der SIGNUM!-Manager bei Application Systems Heidelberg sowie ein paar kleine Utilities. Weitere Programme – zumeist Utilities wie DISKINFO – wurden als Sharewareprogramme realisiert und auch so vertrieben. Wenn der ST oder der TT gerade mal nicht zum Programmieren herhalten müssen, stehen sie für Textverarbeitungszwecke oder ein gutes Spiel bereit.

An dieser Stelle möchte ich mich bei allen bedanken, die mich unterstützt haben und mir das Hobby und den Beruf, der in die gleiche Richtung geht – jedoch keine Atari ST/TT-Programmierung, sondern eher Anwendungsprogrammierung unter Betriebssystem OS/2 und dem Großrechnerbetriebssystem MVS –, ermöglicht haben.

Ein besonders herzliches Dankeschön möchte ich auch an meine geduldige Frau Heike richten, die mich nur allzu oft am Rechner vorfindet, ständig für mich Korrektur liest und immer Verständnis für meine zeitraubende Beschäftigung zeigt.

Julian F. Reschke

Geboren 1965 in Bremen, war ich noch gerade rechtzeitig dabei, um den Beginn der Computer-Revolution im Heimbereich mitzuerleben. Nachdem programmierbare Taschenrechner (wer erinnert sich noch an das HI-LO-Game auf dem TI 57?) ihren ersten Reiz verloren hatten, mußte ein "richtiger" Computer her. Es machten aber nicht etwa ein Tandy, ein Apple oder ein AIM das Rennen, sondern es war ein Sinclair ZX81, den man zum Schleuderpreis von 648 Mark mit sage und schreibe 16 KByte RAM bekam. Das war im Herbst 1981. Doch schon ein knappes halbes Jahr später hatte die Computeszene ihren neuen Star: den Atari 800. Mit 256 Farben, vier Tonkanälen, einem vernünftigen Betriebssystem und – natürlich – phantastischen Spielen (Ballblazer-Fans, hört Ihr mich?) war der Heimcomputer von heute geboren. Daß er mit 48 KByte RAM und einem Diskettenlaufwerk mit 90 KByte Speicherkapazität die ungeheure Summe von 3200 Mark kostete, konnte mich nur kurz aufhalten.

Verschiedene Programme wurden fertiggestellt und mit unterschiedlichem Erfolg verkauft ("HIGHWAY DUEL", "NADRAL" und "MEMO-BOX"). Letztes Projekt war das "ATARI Profibuch", in das ich meine langjährigen Erfahrungen mit dieser Rechnergeneration einfließen ließ.

Zum passenden Zeitpunkt betrat der Atari ST die Szene – nachdem die 8-Bitter von Atari ausgelotet waren, gab es wieder viel zu entdecken. Wegen meines Studiums der Mathematik und Informatik an der Westfälischen Wilhelms-Universität Münster mußten die Erkundungsreisen in die Tiefen des ST jedoch in erster Linie in den Semesterferien stattfinden. Doch im Herbst 1987 war es dann soweit – mit dem "ATARI ST Profibuch" erschien gewissermaßen der Urahn des nun vorliegenden Buchs.

Seit Januar 1988 erscheint im "ST-Magazin" eine monatliche Kolumne, in der ich über die neuesten Entwicklungen bei der TOS-Programmierung berichte. Dort werden auch Korrekturen und Ergänzungen zu diesem Werk zuerst erscheinen.

Neben vielerlei kleinen Hilfsprogrammen (wie "BigScreen", siehe Anhang) entstanden einige Libraries (wie die "FlyDials") sowie die Festplattensoftware "SCSI-Tool" (Hard & Soft) und etliche Neuauflagen des "ATARI ST Profibuchs". Als Haupt-Entwicklungssprache findet ANSI-C Verwendung – keine andere für TOS erhältliche Hochsprache ist vergleichbar ausgereift und verfügt über ebenso umfangreiche wie standardisierte Bibliotheken – sehr wichtig bei Softwareportierungen. Assembler zu verstehen bleibt bei der Fehlersuche dennoch nützlich.

Mittlerweile steht ein TT030/6 mit Großbildschirm auf dem Tisch – Arbeitsgeschwindigkeit und Bildschirmformat sind eine Riesenerleichterung gegenüber früheren Tagen.

Ich bedanke mich bei allen denen, die ich durch das gemeinsame Hobby kennengelernt habe und die mich mit ihrem Know-how unterstützt haben. Stellvertretend für viele seien da nur Arnd Beißner, Stefan Eissing und Gereon Steffens genannt. Auch alle Kollegen "vom Fach" und die Atari-Mitarbeiter in Raunheim (Normen Kowalewski und seine Vorgänger) und Sunnyvale (Leonard Tramiel, Bill Rehbock, Allan Pratt, Ken Badertscher und Mike Fulton) dürfen sich angesprochen fühlen. Ebenso ein Hallo an die Redaktion des "ST-Magazins", die sich Monat für Monat mit meiner "schweren Kost" beschäftigen darf.

Schließlich noch ein Gruß an meine Freunde und Kommilitonen, die mich oft genug vom heimischen Bildschirm losgeeist haben, und ebenso an die Besatzung der Informatik der WWU, die ungeduldig auf die Ergebnisse gewisser objektorientierter Forschungen wartet.

Geleitwort zur TOS-Dokumentation

Programmiersprachen

Die folgenden Kapitel dokumentieren das "Application Programming Interface" (API) von TOS. Für fast alle Funktionen ist der entsprechende ANSI-C-Prototyp (also die Funktionsdeklaration) angegeben – meistens, aber nicht immer in Übereinstimmung mit den mit "Turbo C" mitgelieferten Bibliotheken. Änderungen haben wir genau dann vorgenommen, wenn die vorliegenden Bindings entweder falsch oder zumindest "unglücklich" abgefaßt waren.

Natürlich ist durch die Angabe einer C-Funktionsdeklaration noch lange nicht die Parameterübergabe geklärt. Daher finden Sie bei den BIOS-, XBIOS- und GEMDOS-Funktionen zusätzlich die genaue Stackbelegung und bei den GEM-Funktionen die Belegung der einzelnen Ein- und Ausgabefelder.

Damit sollte die Dokumentation auch für Benutzer anderer Hochsprachen oder Assembler möglich sein. Eine Kurzeinführung in die C-Syntax und die benutzten Datentypen, die natürlich nur die allerwichtigsten Informationen aufführen kann, finden Sie im Anhang. Wir können jedem professionellen Entwickler die Benutzung eines ANSI-C-Systems nur ans Herz legen!

Schichtenmodell

TOS ist ziemlich sauber in mehrere Schichten gegliedert. Die folgenden Kapitel sind entsprechend dieser Hierarchie aufsteigend sortiert. Einige Faustregeln zur sauberen Programmierung lauten:

1. *Niemals* für eine Aufgabe Aufrufe verschiedener Betriebssystemschichten vermischen. Beispiel: In einem GEM-Programm fragt man Maus und Tastatur per AES, nicht etwa per BIOS, ab. Alles andere kann zu Konflikten zwischen den verschiedenen Schichten (hier: Tastenpufferung) führen.

2. *Niemals* von irgendwelchen unsicheren Annahmen über interne Zusammenhänge zwischen den einzelnen Schichten ausgehen. Beispiel: Ein GEMDOS-Laufwerk kann sowohl auf einem BIOS- als auch auf einem Meta-DOS-Gerät liegen. Die Maus hängt normalerweise am IKBD-Chip, muß es aber nicht (externe Tastaturinterfaces, neue Hardware von Atari).

3. *Immer* nach Möglichkeit die höchste Betriebssystemschicht benutzen.

   Beispiel: Ein XControl-Modul kann die Länderkennung aus dem Betriebssystemheader extrahieren – besser aber ist es, dazu die entsprechende XControl-Datenstruktur zu bemühen.

Dokumentiert oder nicht dokumentiert?

Wir haben uns bemüht, immer möglichst präzise das wiederzugeben, was die Originaldokumentation von Atari beschreibt. Leider war das nicht immer möglich:

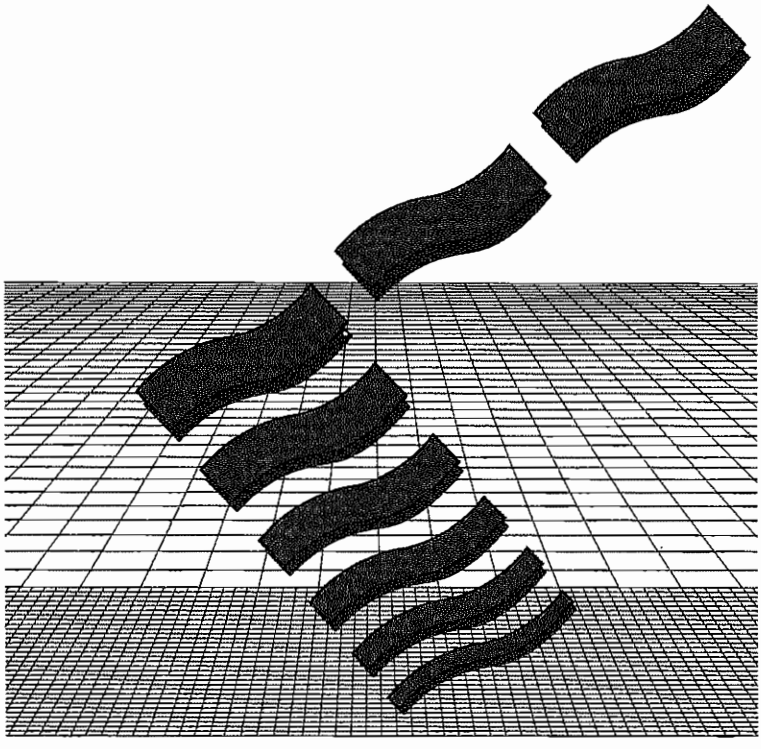
- Manche Funktionen sind in der Originaldokumentation offensichtlich falsch beschrieben; einige Dinge sind in verschiedenen Originaltexten widersprüchlich dokumentiert.

- Manche Funktionen sind dort entweder gar nicht oder unvollständig beschrieben; in solchen Fällen haben wir unsere eigenen Erkenntnisse wiedergegeben und diese als solche gekennzeichnet.

Für die Korrektheit unserer Beschreibungen können wir natürlich keinerlei Gewähr übernehmen. Ferner kann ein Buch natürlich niemals genauso aktuell wie die offizielle Dokumentation des Herstellers sein. Jeder professionelle Softwareentwickler sollte sich daher auch bei der entsprechenden Atari-Niederlassung als Entwickler registrieren lassen, um neben diesem Buch auch die Originaldokumentation und den Entwicklersupport in Anspruch nehmen zu können.

Künftige Weiterentwicklungen werden entweder in ein überarbeitetes "Profibuch" oder einen Ergänzungsband Eingang finden. Stets aktuelle Informationen zu diesem Thema finden Sie in den diversen Fachzeitschriften, insbesondere in der monatlichen Kolumne "Atarium" im "ST-Magazin".

Teil I



TOS – Das Betriebssystem des Atari

Kapitel 1: BIOS und XBIOS

Einleitung

BIOS und XBIOS sind enge Verwandte. Sie sind in der gleichen "Betriebssystemschicht" angeordnet und teilen sich sogar denselben TRAP-Dispatcher.

Das BIOS ("Basic Input/Output System") ist das Bindeglied zwischen GEMDOS und Hardware (im "Hitchhiker's Guide to the BIOS" ist das entsprechende Kapitel nicht umsonst "GEMDOS BIOS Calls" betitelt). Dazu stellt es Funktionen zur zeichenorientierten und blockorientierten Kommunikation, zur Speicherverwaltung und für den Zugriff auf Systemvektoren zu Verfügung. Bis auf einige wenige Ausnahmen (bei der Behandlung der Zeitfunktionen) greift GEMDOS ausnahmslos auf BIOS-Funktionen zu.

Insgesamt gilt die Faustregel: das BIOS ist für all das verantwortlich, was unterhalb der Ebene von GEMDOS und GEM stattfindet und nichts mit dem XBIOS zu tun hat – beispielsweise auch die Systeminitialisierung.

Das XBIOS ("eXtended BIOS") hingegen enthält eine große Anzahl hardwareabhängiger Erweiterungen – meist vollständig unabhängig von anderen Betriebssystemschichten. Faustregel sollte sein: XBIOS-Funktionen möglichst nur dann benutzen, wenn es keine Funktion einer "höheren" Betriebssystemschicht gibt, die man statt dessen benutzen könnte.

- Viele der Funktionen dienen der Konfiguration von Hardware oder allerunterster Betriebssystem-Schichten. Gute Beispiele dafür sind "Cursconf()" und "Setprt()". Einstellungen dieser Art werden normalerweise vom Kontrollfeld vorgenommen, in eigenen Programmen sollte man immer nur die Einstellungen abfragen.

- Funktionen wie "Kbdvbase()" sind eigentlich nur für residente Programme interessant.

- Eine Funktion, die man in Anwendungsprogrammen *braucht*, ist "Keytbl()". Anders ist es nicht möglich, länderunabhängig Tastatur-Scancodes auszuwerten.

- Die Grafikfunktionen sind Paradebeispiele für Funktionen, die man nicht in Anwendungsprogrammen einsetzen sollte. Grund: Sie sind *nur* für die ST-, STE- und TT-Grafikhardware definiert und funktionieren im allgemeinen für Grafikkarten nicht oder nur eingeschränkt. Das VDI-Konzept erlaubt prinzipiell Grafikkarten, deren Bildspeicher für die CPU überhaupt nicht zugänglich ist. Auch die Farbpaletten-Funktionen des XBIOS sind nur sehr eingeschränkt für "farbigere" Grafikkarten einsetzbar. Anwendungsprogramme sollten daher *ausschließlich* GEM-Funktionen einsetzen.

Zeichenorientierte Funktionen

Welche Kanäle gibt es?

Das BIOS kennt von Haus aus folgende sechs Ein- und Ausgabekanäle:

Kanal	Name	Belegung
0	PRT	Parallele Schnittstelle
1	AUX	Serielle Schnittstelle
2	CON	Konsole (Bildschirm mit VT52-Sequenzen)
3	MIDI	MIDI-Schnittstelle
4	IKBD	Intelligent Keyboard Processor
5	RAWCON	Konsole (ohne Steuersequenzen)

Auf neueren TOS-Versionen können über die XBIOS-Funktion "Bconmap()" (siehe unten) weitere Kanäle installiert werden. Zur Bedienung dieser Kanäle bietet BIOS vier Ein- und Ausgabefunktionen:

Name	Funktion
Bconstat()	Status des Eingabegeräts feststellen
Bconin()	Zeichen einlesen
Bconout()	Zeichen ausgeben
Bcostat()	Status des Ausgabegeräts feststellen

Vorsicht: für die Funktion "Bcostat()" sind die Kanäle 3 und 4 vertauscht.

Installation eigener Treiber

Seit TOS 1.02 benutzt das BIOS eine dokumentierte Tabelle mit Zeigern auf die einzelnen Funktionen. Insgesamt gibt es Vektoren für acht Kanäle, doch nur sechs von ihnen können benutzt werden:

Adresse	Name	Belegung
$51E	xconstat	Acht Zeiger auf Eingabestatus-Routinen
$53E	xconin	Acht Zeiger auf Eingabe-Routinen
$55E	xcostat	Acht Zeiger auf Ausgabestatus-Routinen
$57E	xconout	Acht Zeiger auf Ausgabe-Routinen

Wer eigene Routinen installieren möchte, sollte folgendes beachten:

Die Funktionen werden mit den gleichen Parametern wie die entsprechenden BIOS-Funktionen aufgerufen (nur BIOS-Funktionsnummer und Kanalnummer fehlen). Der Return-Wert wird in Register D0 zurückgeliefert. Alle Register dürfen benutzt werden, da der BIOS-Dispatcher selbst für das Retten und Wiederherstellen der "garantierten" Register sorgt.

Ab TOS 2.00 werden die Gerätenummern über 5 ("RAWCON:") mittels "Bconmap()" verwaltet. Daher können die freien Einträge in den Systemvektoren – wenn überhaupt – nur unter älteren TOS-Versionen benutzt werden.

Anmerkungen zu CON: und RAWCON:

Der VT52-Emulator

Beim ursprünglichen Entwurf des ST hat Atari deutlich in Richtung "Digital Equipment" geschielt und einige Anleihen beim verbreiteten Terminal "VT52" genommen. Davon ist nicht nur die Tastatur, sondern auch die für "CON:" benutzte Terminalemulation betroffen.

Der große Vorteil liegt darin, daß genau diese Steuersequenzen auch auf vielen anderen Systemen benutzt werden. Gerade beim Anruf von Mailboxen mit Terminalemulation macht sich das positiv bemerkbar.

Alle Steuersequenzen werden durch ESC (also den ASCII-Wert 27) eingeleitet. Daher nennt man sie auch (genau wie bei Druckern) üblicherweise Escape-Sequenzen. Dem ESC-Zeichen folgen eines oder auch mehrere Zeichen, die dann die eigentliche Funktion auslösen.

Cursor up (VT52 ESC A)
Bewegt den Cursor um eine Zeile nach oben. War der Cursor bereits in der obersten Zeile, passiert nichts.

Cursor down (VT52 ESC B)
Bewegt den Cursor um eine Zeile tiefer. War der Cursor bereits in der untersten Zeile, passiert nichts.

Cursor forward (VT52 ESC C)
Bewegt den Cursor um eine Position nach rechts, höchstens aber bis zum rechten Bildschirmrand.

Cursor backward (VT52 ESC D)
Bewegt den Cursor um eine Spalte nach links, höchstens aber bis zum linken Bildschirmrand.

Clear screen (and home cursor) (VT52 ESC E)
Löscht den gesamten Bildschirm und setzt den Cursor in die linke obere Ecke.

Home cursor (VT52 ESC H)
Setzt den Cursor in die linke obere Ecke.

Reverse index (VT52 ESC I)
Bewegt den Cursor um eine Zeile nach oben. War der Cursor bereits in der obersten Zeile, wird oben eine Leerzeile eingefügt und der Rest des Bildschirms um eine Zeile nach unten verschoben.

Erase to end of page (VT52 ESC J)
Löscht ab der aktuellen Cursorposition (einschließlich) den gesamten Bildschirm.

Clear to end of line (VT52 ESC K)
Löscht ab der aktuellen Cursorposition bis zum Ende der Zeile.

Insert line (VT52 ESC L)
Fügt an der aktuellen Cursorposition eine leere Zeile ein. Der Cursor wird an den Anfang dieser Zeile gesetzt und der Rest des Bildschirms um eine Zeile nach unten verschoben.

Delete line (VT52 ESC M)
Löscht die aktuelle Zeile und verschiebt den Rest des Bildschirms um eine Zeile nach oben. Dabei entsteht am unteren Rand eine leere Zeile. Der Cursor bleibt in der gleichen Zeile und wird an den linken Rand bewegt.

Position cursor (VT52 ESC Y)
Positioniert den Cursor frei auf dem Bildschirm. Dazu übergibt man zusätzlich die gewünschte Y- und X-Position (jeweils um 32 erhöht).

Set foreground color (VT52 ESC b)
Setzt die Schriftfarbe. Dazu übergibt man als zusätzlichen Wert die gewünschte Farbnummer (Anzahl der Farben vom Bildschirmmodus abhängig) als Byte (von dem nur die unteren vier Bits ausgewertet werden).

Set background color (VT52 ESC c)
Setzt die Hintergrundfarbe. Dazu übergibt man als zusätzlichen Wert die gewünschte Farbnummer (Anzahl der verfügbaren Farben vom Bildschirmmodus abhängig).

Erase beginning of display (VT52 ESC d)
Löscht alle Zeichen zwischen dem Bildschirmanfang und der Cursorposition.

Enable cursor (VT52 ESC e)
Macht den Cursor sichtbar.

Disable cursor (VT52 ESC f)
Macht den Cursor unsichtbar.

Save cursor position (VT52 ESC j)
Speichert intern die momentane Cursorposition.

Restore cursor position (VT52 ESC k)
Setzt den Cursor wieder auf die mittels "Save cursor" gespeicherte Position. Die dabei gesicherte Position wird gelöscht!

Erase entire line (VT52 ESC l)
Löscht die aktuelle Zeile und setzt den Cursor an den linken Bildschirmrand.

Erase beginning of line (VT52 ESC o)
Löscht alle Zeichen zwischen dem Zeilenanfang und der Cursorposition.

Enter reverse video mode (VT52 ESC p)
Schaltet für alle folgenden Bildschirmausgaben auf inverse Schrift (Vertauschen von Schrift- und Hintergrundfarbe).

Exit reverse video mode (VT52 ESC q)
Schaltet wieder auf normale Bildschirmdarstellung um.

Wrap at end of line (VT52 ESC v)
In diesem Modus wird beim Erreichen des Zeilenendes automatisch die nächste Zeile begonnen.

Discard at end of line (VT52 ESC w)
In diesem Modus wird erst dann eine neue Zeile begonnen, wenn ein "Carriage Return" (13) und ein "Line Feed" (10) ausgegeben werden.

Intern benutzt der VT52-Emulator die Systemvariablen "conterm" (diverse Einstellungen), "con_state" (wird bei der Umschaltung zwischen "normalen" Zeichen und Steuersequenzen benutzt) und "sav_row" (zum Speichern der Cursorposition bei ESC "j").

Laut "Hitchhiker's Guide to the BIOS" ist allerdings nur die erste von ihnen offiziell dokumentiert!

Wie groß ist der Bildschirm?

Von einem Programm, das VT52-Sequenzen für seinen Bildschirmaufbau benutzt, erwartet man zu Recht, daß es die gesamte verfügbare Fläche nutzt. Doch wie stellt man fest, wie viele Zeichen auf den Bildschirm passen?

In der geringen Auflösung des ST könnten es gerade mal 25 * 40 sein, während auf dem TT in der hohen Auflösung beeindruckende 60 * 160 Zeichen Platz finden. Noch schwieriger wird es bei anderen Grafikkarten und erst recht dann, wenn ausgefeilte grafische Oberflächen TOS-Programme in Fenstern flexibler Größe ablaufen lassen.

Leider gibt es keine Möglichkeit, den verfügbaren Platz abzufragen, ohne die Betriebssystemschichten BIOS, XBIOS und GEMDOS zu verlassen. Daher sei an dieser Stelle folgender Vorschlag gemacht: falls die Environmentvariablen ROWS und COLUMNS existieren, dann enthalten sie die maximale Zeilen- bzw. Spaltenzahl.

Vorteile dieses Verfahrens:

- Es werden keine undokumentierten Betriebssystemeigenschaften benötigt.
- Jede tastaturorientierte Shell (und an deren Anwenderkreis wendet sich ja dieser Vorschlag) kann Environmentvariablen setzen.
- Das Desktop "Gemini" bzw. die UNIX-Shell "Mupfel" nutzen diese Methode seit geraumer Zeit.

Ausgaben ohne Terminalemulation

Mit "RAWCON:" kann man alle Zeichen gewissermaßen ungefiltert auf dem Bildschirm ausgeben. Vorteile sind daher nicht nur der größere Zeichenvorrat (anstelle der Steuersequenzen), sondern auch eine deutlich höhere Ausgabegeschwindigkeit als bei "CON:".

BIOS- und VDI-Escapefunktionen

Interessanterweise benutzen die VDI-Escapefunktionen des ROM-Bildschirmtreibers den gleichen Code und die gleichen Variablen wie der VT52-Emulator des BIOS.

Dennoch sollte man innerhalb eines Programms keinesfalls beide Betriebssystemteile gemischt aufrufen. Der Grund: die BIOS-Ausgaberoutinen gehen ab TOS 1.02 über Vektortabellen und können damit umgelenkt werden. Die VT52-Escapefunktionen hingegen sind ein Teil des VDI-Bildschirmtreibers und könnten daher durch einen anderen Bildschirmtreiber ersetzt worden sein!

Benutzte Systemvariablen

Über "conterm" kann man einige Attribute verstellen (Tastenklick, Tastenwiederholung, "Ping" bei Ausgabe von CTRL-G, Rückliefern von Scancodes bei "Bconin()"). Ab TOS 1.06 gibt es die zwei zusätzlichen Vektoren "bell_hook" und "kcl_hook", über die man eigene Routinen für das "Ping"-Geräusch und den Tastenklick installieren kann.

Über den Status der Umschalttasten wird in einer BIOS-internen Variable Buch geführt. Ihre Adresse ist eigentlich nur für Interruptroutinen interessant und kann mit Hilfe von "_sysbase" ermittelt werden. Andere Programme sollten statt dessen "Kbshift()" verwenden.

Anmerkungen zu PRT:

Auswahl parallel/seriell

Die BIOS-Routinen zur Druckerausgabe geben normalerweise auf der parallelen Schnittstelle aus. "Bconout()" ist prinzipiell darauf vorbereitet, die Drucker-Voreinstellungen des XBIOS ("Setprt()") in Hinsicht auf die zu benutzende Schnittstelle (parallel/seriell) zu berücksichtigen. Leider ist die Abfrage in allen bekannten TOS-Versionen fehlerhaft, so daß Anwendungsprogramme die Einstellungen selbst auswerten sollten (hinzu kommt, daß "Bcostat()" die Einstellungen völlig ignoriert).

Der DIABLO-Treiber

Die Atari-Laserdrucker SLM 804 und 605 erfreuen sich aufgrund der hohen Ausgabegeschwindigkeit und des günstigen Preises großer Beliebtheit. Zum Lieferumfang gehört der DIABLO-Treiber, der sich auf BIOS-Ebene installiert und alle Ausgaben auf "PRT:" abfängt. Eine Liste der unterstützten Escape-Sequenzen findet sich im Original-Handbuch, so daß wir aus Platzgründen an dieser Stelle auf einen Abdruck verzichten.

Anmerkungen zu AUX:

Die erweiterten seriellen Schnittstellen des TT (und Mega STE) haben natürlich auch Softwareprobleme aufgeworfen: sämtliche Programme gehen davon aus, daß BIOS-Kanal 1 mit der seriellen Schnittstelle verbunden ist. Daneben war es bislang selbstverständlich, daß man die vom BIOS-Kanal 1 benutzte Hardware mit "Rsconf()" konfigurieren kann. Atari hat das Problem auf verschiedenen Ebenen gelöst. Zunächst einmal gibt es nun mehr als die bisherigen sechs BIOS-Kanäle. Je nach verwendeter Hardware kennt beispielsweise das TT-BIOS die folgenden neuen Geräte (die "erweiterten Gerätenummern"):

Kanal	Belegung
6	ST-kompatible serielle Schnittstelle ("Modem 1")
7	SCC Kanal B ("Modem 2")
8	TTMFP-Schnittstelle ("Seriell 1")
9	SCC Kanal A ("Seriell 2")

Nur die Belegung von Kanal 6 liegt als ST-kompatible serielle Schnittstelle fest. Alle anderen sind von der verwendeten Hardware abhängig (wie viele es sind, kann man mit "Bconmap()" abfragen).

Kanal 1 ist nunmehr ein Platzhalter für "die" serielle Schnittstelle. Mittels "Bconmap()" kann man ihm irgendeine der erweiterten Gerätenummern zuordnen.

Dabei wird übrigens auch dafür gesorgt, daß bei der Umschaltung von Gerätenummer 1 die vier dazugehörigen BIOS-Systemvektoren angepaßt werden.

Das "Seriell"-Modul des erweiterten Kontrollfelds sorgt für eine entsprechende Initialisierung. Programme, die von den erweiterten Gerätenummern wissen, können natürlich auch über die BIOS-Zeichenausgaberoutinen auf die neuen Schnittstellen zugreifen.

Warum "Bconmap()" nun eine XBIOS- und nicht eine BIOS-Funktion ist? Legitimerweise gehen praktisch alle existierenden Programme davon aus, daß man mit "Rsconf()" spezielle Eigenschaften von BIOS-Kanal 1 verändern kann.

Das XBIOS sorgt daher nicht nur für die "Umlenkung" von Kanal 1 auf die passenden Routinen, sondern auch für die Umschaltung auf eine entsprechende "Rsconf()"-Routine. Auch für die Anpassung der "Iorec()"-Funktion wird gesorgt.

"Bconmap" verfügt dazu über eine BCONMAP-Struktur, die entsprechende Funktionszeiger für die einzelnen Geräte enthält:

```
typedef struct
{
     WORD      (*Bconstat) ();      /* Funktionszeiger */
     LONG      (*Bconin) ();
     LONG      (*Bcostat) ();
     void      (*Bconout) ();
     ULONG     (*Rsconf) ();
     IOREC     *iorec;              /* Zeiger auf IOREC-Struktur */
} MAPTAB;
```

```
typedef struct
{
     MAPTAB    *maptab;       /* Zeiger auf Funktionstabelle */
     WORD      maptabsize;    /* Anzahl der Einträge */
} BCONMAP;
```

"maptabsize" gibt an, wie viele erweiterte Kanäle bereits installiert sind. Wenn man 5 addiert, erhält man die größte erlaubte Nummer für die BIOS-Zeichenfunktionen und natürlich für "Bconmap()".

"maptab" zeigt auf ein Feld von "maptabsize" MAPTAB-Strukturen. Jede dieser Strukturen enthält Zeiger auf die entsprechenden BIOS-Zeichenfunktionen, eine "Rsconf()"-Funktion und den IOREC des betreffenden Geräts.

Die neue Vielfalt bei den seriellen Schnittstellen hat natürlich einige wichtige Konsequenzen:

- Viele der mit "Rsconf()" möglichen Einstellungen sind eng mit der Hardware des MFP-Bausteins verbunden. Daher muß man sich damit abfinden, daß viele Einstellungen nicht mehr unbedingt bei jeder Schnittstelle möglich sind.

- Direkte Folge: Einstellungen der seriellen Hardware sollten im allgemeinen spezialisierten Konfigurationsprogrammen wie dem Kontrollfeld vorbehalten bleiben und aus den eigentlichen Anwendungsprogrammen entfernt werden.

Blockorientierte Funktionen

Einleitung

Als blockorientiert bezeichnet man gemeinhin Geräte, die Daten ausschließlich in größeren Blöcken (meist Sektoren genannt) übertragen können. BIOS kann 32 derartige Geräte verwalten. Dazu stehen folgende Funktionen zu Verfügung:

Drvmap(): Liste der verfügbaren Geräte abfragen

Getbpb(): Informationen über das Gerät abfragen

Mediach(): Medienwechselstatus erfragen

Rwabs(): Sektoren lesen oder schreiben

Drvmap(): Wie viele Geräte gibt es?

Zur Buchführung über die vorhandenen Geräte benutzt BIOS die Systemvariable "_drvbits". Jedes Bit in "_drvbits" entspricht einem verfügbaren Gerät. Wer eigene BIOS-Treiber installiert, muß also das passende Bit in "_drvbits" setzen, um das Gerät anzumelden. Wer den Wert nur abfragen will, sollte statt dessen die BIOS-Funktion "Drvmap()" zu Rate ziehen (wenn man die Wahl hat, sollte man *immer* eine Betriebssystemfunktion der Benutzung einer Systemvariablen vorziehen).

Ein gesetztes Bit bedeutet natürlich noch längst nicht, daß auch Daten gelesen oder geschrieben werden können. Die Drive-Bits geben nur darüber Auskunft, für welche Gerätenummern ein Treiber installiert ist. Man stelle sich vor, es wäre anders: dann müßte Bit 0 in dem Moment gelöscht werden, in dem die Diskette aus Laufwerk A: entfernt wird (das gleiche gilt natürlich auch für jedes andere wechselbare Medium, zum Beispiel für eine Wechselplatte).

Und noch ein wichtiger Hinweis: BIOS-Geräte und GEMDOS-Laufwerke müssen *nicht* das gleiche sein. Das erkennt man schon daran, daß BIOS zur Zeit doppelt so viele Geräte wie GEMDOS erlaubt. Bei Benutzung von "Meta-DOS" kann es außerdem vorkommen, daß GEMDOS Geräte nicht über BIOS, sondern über einen von "Meta-DOS" installierten Treiber ansteuert. Die Liste der für GEMDOS verfügbaren Geräte sollte man daher besser mit "Dsetdrv()" abfragen!

Getbpb(): Informationen über ein Gerät ermitteln

"Getbpb()" liefert eine Geräteinformationsstruktur folgender Form:

```
typedef struct
{
     WORD  recsiz;   /* Bytes pro Sektor */
     WORD  clsiz;    /* Sektoren pro Cluster (Einheit) */
     WORD  clsizb;   /* Bytes pro Cluster */
     WORD  rdlen;    /* Länge d. Wurzelverzeichnisses in Sektoren*/
     WORD  fsiz;     /* Länge des File Allocation Table (FAT) */
     WORD  fatrec;   /* Startsektor der zweiten FAT */
     WORD  datrec;   /* Sektornummer des ersten freien Clusters */
     WORD  numcl;    /* Gesamtzahl der Cluster auf dem Medium */
     WORD  bflags;   /* Bitvektor, zur Zeit nur Bit 0 belegt:
                        0 (12-Bit-FAT), 1 (16-Bit-FAT) */
} BPB;
```

Eigentlich würde man erwarten, daß sich BIOS an dieser Stelle nur für die Anzahl der Sektoren und ihre Größe in Bytes interessiert. Wie man aber sieht, sind schon alle für das GEMDOS-Dateisystem interessanten Informationen eingetragen. Wie paßt das zur sonst so einleuchtenden Aufteilung in "höhere" und "niedrigere" Betriebssystemschichten?

Die hier zurückgelieferten Informationen werden bei praktisch allen Medien aus dem Bootsektor (Sektor 0) ermittelt, der eben auch schon Informationen über die logischen Strukturen enthält. GEMDOS selbst braucht sich damit nicht in die Niederungen von Bootsektoren und ähnlichem Kram herabzubewegen. Weitere Informationen zur BPB-Struktur finden Sie konsequenterweise in der Einführung zum GEMDOS-Dateisystem.

Bedeutet dies, daß BIOS nur Geräte mit GEMDOS-Dateisystem bedienen kann? Keinesfalls! Hauptsache ist nur, daß im BPB alle Felder mit sinnvollen Werten besetzt sind. Die Gesamtzahl der Sektoren kann man daraus dann selbst berechnen und fortan per "Rwabs()" auf das Gerät zugreifen:

```
/* Anzahl der verfügbaren Sektoren eines BIOS-Geräts feststellen.
   Return-Wert ist entweder die Anzahl oder im Fehlerfall 0 */

LONG SecCount (WORD device)
{
     BPB *B = Getbpb (device);
     if (!B)   /* Fehler? */
          return 0L;
     else
          return B->datrec + (LONG)B->numcl * B->clsiz;
}
```

Mediach(): Medienwechselstatus erfragen

Das BIOS kommt auch mit wechselbaren Medien zurecht. Dazu gibt es einerseits definierte Fehlermeldungen, die ein Treiber zurückliefern darf. Andererseits gibt es auch die Funktion "Mediach()", mit der man sich explizit über den Status des Mediums informieren kann.

Da aus technischen Gründen nicht bei allen Gerätetypen eine sichere Auskunft möglich ist, gibt es auch für "kann sein" einen definierten Return-Wert.

Vorsicht mit defekten Diskettenlaufwerken oder nicht einwandfrei funktionierenden Wechselplattentreibern: GEMDOS ist absolut darauf angewiesen, daß die Medienwechselerkennung funktioniert. Anderenfalls ist es sehr wahrscheinlich, daß früher oder später

Verwaltungsinformationen (FAT, Verzeichnisse) auf das falsche Medium geschrieben werden!

Rwabs(): Sektoren lesen und schreiben

Über diese Funktion werden alle Lese- und Schreibzugriffe abgewickelt. Für zum AHDI 3.0 kompatible Festplattentreiber wurde die Funktionsdefinition in Hinsicht auf Retries (Wiederholungen im Fehlerfall) und auf den direkten Plattenzugriff erweitert.

Installation eigener Treiber

Die Installation eigener BIOS-Treiber ist sehr einfach, da es für die drei letztgenannten Funktionen Systemvektoren gibt. Ein neuer Treiber muß also nur einen entsprechenden Eintrag in der Systemvariable "_drvbits" vornehmen und sich in die Vektoren "hdv_rw", "hdv_mediach" und "hdv_bpb" einklinken. Da man sich die Vektoren ja unter Umständen mit anderen Treibern teilt, sollten dabei folgende Regeln beachtet werden:

1. Es sollte das XBRA-Verfahren für vektorverbiegende Programme benutzt werden, damit andere Treiber korrekt weiterfunktionieren (und sich gegebenenfalls wieder entfernen können).

2. Alle Funktionen erhalten die BIOS-Laufwerksnummer als Parameter, und jeder Treiber sollte höllisch genau darauf aufpassen, daß er sich nicht im falschen Moment angesprochen fühlt.

Treiber für Diskettenlaufwerke

Einleitung

Nach Systemstart kennt das BIOS zunächst nur Funktionen für die Diskettenlaufwerke. Während des Bootvorgangs wird versucht, beide Laufwerke anzusprechen. Das Ergebnis – die Anzahl der angeschlossenen Laufwerke – kann man in der Systemvariable "_nflops" finden (intern wird dazu über "hdv_init" und "hdv_boot" gesprungen). *Wenn* Laufwerke gefunden wurden, trägt BIOS die Drive-Bits für Gerät 0 und Gerät 1 ein – ungeachtet der tatsächlichen Zahl von Diskettenlaufwerken. Ist nur ein Laufwerk angeschlossen, dann wird das zweite Gerät per Software "emuliert" (wer kennt die Meldung "Bitte Diskette B: in Laufwerk A einlegen" nicht?).

Für seine Lese- und Schreibzugriffe bedient sich der BIOS-Diskettentreiber der XBIOS-Funktionen "Floprd()", "Flopwr()" und gegebenenfalls "Flopver()" (und zwar dann, wenn die Systemvariable "_fverify" es so will).

Der Bootsektor

Wie man weiß, gibt es eine große Zahl von möglichen Diskettenformaten. Grund sind nicht nur die unterschiedlichen technischen Voraussetzungen (einseitig oder doppelseitig, 40 oder 80 Spuren, 5 1/4 oder 3 1/2 Zoll, normale oder hohe Schreibdichte), sondern auch die vielfältigen Versuche der ST-Besitzer, ihren Diskettenlaufwerken auch noch das letzte Byte zu entlocken (was vor dem Siegeszug der Festplatten auch wirlich ein wichtiges Thema war).

Zum Glück erweist sich das BIOS in dieser Hinsicht als sehr flexibel. Nach jedem Medienwechsel inspiziert es den Bootsektor der eingelegten Diskette. Dieser sollte folgende Daten enthalten:

Offset	Name	Belegung
$00	BRA.S	CPU-Sprungbefehl zur Bootroutine
$02	FILLER	6 Füllbytes
$08	SERIAL	3 Bytes Seriennummer
$0B	BPS	Bytes pro Sektor
$0D	SPC	Sektoren pro Cluster
$0E	RES	Reservierte Sektoren am Anfang des Mediums
$10	NFATS	Anzahl der File Allocations Tables
$11	NDIRS	Anzahl der Einträge im Wurzelverzeichnis
$13	NSECTS	Gesamtzahl der Sektoren
$15	MEDIA	"Media"-Byte
$16	SPF	FAT-Größe in Sektoren
$18	SPT	Track-Größe in Sektoren
$1A	NSIDES	Seiten pro Medium
$1C	NHID	Versteckte Sektoren

Da Sinn und Zweck des GEMDOS-Dateisystems ist, MS-DOS-Disketten lesen und beschreiben zu können, liegen alle 16 Bit großen Werte im Intel-Format (also High- und Low-Byte vertauscht!) vor!

Die Bedeutung der Felder im einzelnen:

BRA.S Dieses Feld ist nur bei auto-bootenden Disketten interessant und enthält dann einen Branch-Befehl zur eigentlichen Bootroutine.

FILLER	Sechs Bytes, die beliebig vergeben und für eigene Zwecke benutzt werden dürfen.
SERIAL	Eine 24 Bit große Seriennummer. Diese Seriennummer wird bei einem "unsicheren" Medienwechselstatus begutachtet und sollte nach Möglichkeit für jede benutzte Diskette verschieden sein (siehe auch "Protobt()").
BPS	Sektorgröße in Bytes.
SPC	Sektoren pro GEMDOS-Cluster.
RES	Anzahl der reservierten Sektoren (meist 1, weil der Bootsektor nur einen Sektor "verbraucht").
NFATS	Anzahl der File Allocation Tables (unter GEMDOS immer 2).
NDIRS	Anzahl der Einträge im Wurzelverzeichnis (im Gegensatz zu Unterverzeichnissen ist die Größe bei einem Wurzelverzeichnis ein fester Wert).
NSECTS	Gesamtzahl der Sektoren (inklusive der reservierten, also des Bootsektors).
MEDIA	"Media"-Byte. Dieses Byte wird vom BIOS nicht ausgewertet, da sich alle interessanten Informationen besser auf andere Weise ermitteln lassen. Anders unter MS-DOS (dazu gleich mehr).
SPF	FAT-Größe in Sektoren.
SPT	Anzahl der Sektoren pro Spur.
NSIDES	Anzahl der Seiten (bei Disketten normalerweise 1 oder 2, wer hätte etwas anderes erwartet?).
NHID	Wird vom BIOS ignoriert.

Das XBIOS bietet übrigens zum Erzeugen "frischer" Bootsektoren die Funktion "Protobt()" an.

Bootsektoren bei MS-DOS-Disketten

Leider ist es mit der MS-DOS-Kompatibilität nicht ganz so einfach, wie man es sich wünschen würde.

Das hat verschiedene Ursachen. Zuerst einmal hat ein MS-DOS-Bootsektor eben nur *fast* das gleiche Format:

Offset	Belegung
$00	CPU-Sprungbefehl zur Bootroutine
$03	Acht Zeichen langer Diskettenname

Der Sprungbefehl ist also ein Byte länger und natürlich in der Maschinensprache der Intel-Chips gehalten. Leider erwarten die meisten PC-Kompatiblen an dieser Stelle tatsächlich eine Intel-Befehlssequenz und weigern sich ansonsten beharrlich, die Diskette zu verarbeiten. Daher sollte man die ersten drei Bytes stets mit den Zahlen $E9 $00 $4E füllen. Folge: MS-DOS-kompatible Disketten sind auf dem Atari nicht bootbar.

Nächstes Problem: da, wo das Atari-BIOS eine zufällige Seriennummer erwartet, steht bei MS-DOS-Disketten ein "Diskettenname" (etwa: "MS-DOS 3.2"). Die Folge: wenn man mehrere gleich formatierte Disketten abwechselnd benutzt, funktioniert die Medienwechselerkennung nicht so, wie man es sich wünschen würde.

Kommen wir zum "Media"-Byte. Auch dieses Byte entstammt den Ursprüngen von MS-DOS und dient dort dazu, die verschiedenen Diskettentypen auseinanderzuhalten. Daß dabei längst nicht die gleiche Vielfalt wie über die anderen Felder des Bootsektors zu ereichen ist, dürfte klar sein. Dennoch bestehen viele MS-DOS-Versionen darauf, daß dieses Feld einen "vernünftigen" Wert hat (diese Liste erhebt weder Anspruch auf Vollständigkeit noch auf absolute Korrektheit):

Wert	Bedeutung
$F8	Festplatte oder einseitige Diskette mit 80 Spuren zu 9 Sektoren (360K)
$F9	Doppelseitige Diskette, 80 Spuren zu 9 oder mehr Sektoren (High-Density)
$FC	Einseitige Diskette, 40 Spuren zu 9 Sektoren
$FD	Doppelseitige Diskette, 40 Spuren zu 9 Sektoren
$FE	Einseitige Diskette, 40 Spuren zu 8 Sektoren
$FF	Doppelseitige Diskette, 40 Spuren zu 8 Sektoren

Wer meint, nun wäre MS-DOS zufrieden, der irrt. Zusätzlich muß man noch die ersten beiden (unbenutzten) Einträge beider FATs mit einer Kopie der Media-Bytes und folgenden $FFs füllen (wozu auch immer das gut sein soll). Fest steht, daß das Desktop ab GEM 1.4 auf

- den Intel-Sprungbefehl am Anfang des Bootsektors und
- die Kopien des Media-Bytes am Beginn der FATs

achtet und solche Disketten erfahrungsgemäß auf PCs erfolgreich gelesen und beschrieben werden können.

Schlußbemerkung zu diesem Thema: das BIOS des Atari hat normalerweise gar keine Probleme, auf PCs formatierte Disketten zu "verdauen". Im Zweifelsfall ist also die sicherste Lösung, die Diskette dort zu formatieren, wo sie später gelesen werden soll.

Bootbare Disketten

Wozu gibt es eigentlich bootbare Disketten? Wäre es nicht viel einfacher, das zu startende Programm in den AUTO-Ordner zu kopieren?

Die ersten ausgelieferten STs verfügten nur über ein 64 KByte großes Betriebssystem, das gerade einmal zum Laden des TOS von Diskette reichte. Diese ROMs hatten die Versionsnummer 0.00 und wurden von Atari allen Ernstes "Das Boot" genannt. Neben einer netten Grafikdemo enthielten sie die BIOS-Funktionen "Rwabs()" und "Getbpb()" und die XBIOS-Funktionen "Ssbrk()" und "Floprd()" – gerade eben genug, um Sektoren von einer Diskette zu lesen.

Ohne GEMDOS-Funktionen kann man aber nicht auf Dateien zugreifen, und so muß die dazu nötige Software "gebootet" werden.

Andere Anwendungen für bootbare Disketten:

- Fremde Betriebssysteme wie Minix, die, sind sie einmal geladen, die gesamte Systemverwaltung sowieso selbst übernehmen.

- Videospiele, die ebensowenig Betriebssystemroutinen brauchen, aber möglichst unkompliziert geladen werden sollen.

- Kleine Hilfsprogramme, die möglichst früh während der Systeminitialisierung gestartet werden sollen (bei Anwendern ohne Festplatte sehr beliebt).

Das BIOS wertet den Bootsektor während der Systeminitialisierung aus (und springt dazu durch den Vektor "hdv_boot"). Dazu wird er in einen Puffer geladen und dort eine Prüfsumme über die 256 16-Bit-Werte berechnet. Ist das Ergebnis $1234, dann wird zum ersten Byte im Sektor gesprungen. Die Adresse des Puffers ist nicht dokumentiert, und so muß die im Bootsektor enthaltene Routine positionsunabhängig programmiert sein. Für die Routine im Bootsektor gibt es nun zwei Möglichkeiten, wie sie fortfahren kann. Einerseits darf sie die Kontrolle per "rts" an das BIOS zurückgeben, das dann mit dem normalen Startvorgang fortfährt (dann darf sie allerdings keine Register verändern). Andererseits kann sie die Kon-

trolle auch ganz selbst übernehmen und dann beispielsweise ein völlig anderes Betriebssystem (oder ein Spiel) laden.

Ausführbare Bootsektoren sind übrigens nicht nur für Utilities und Betriebssysteme, sondern auch für *Virenprogramme* interessant.

Daher sollte man niemals mit "fremden" Disketten im Laufwerk booten!

Diskettenwechsel

Eine Beschreibung des Floppytreibers im BIOS wäre natürlich unvollständig, wenn das Thema "Diskettenwechsel" fehlen würde. Im Gegensatz zu anderen Systemen (wie dem Macintosh) bietet die Hardware beim Atari keine direkte Methode, Diskettenwechsel zu registrieren. Das BIOS überwacht statt dessen im Vertical Blank das Schreibschutzsignal, das bei einem Diskettenwechsel kurz unterbrochen wird. Das hat mehrere Folgen:

- Bei schreibgeschützten Disketten meldet das BIOS immer "vielleicht gewechselt". Ab TOS 1.04 wurde wenigstens eine Plausibilitätsprüfung eingeführt, die die Anzahl der gemeldeten Wechsel reduziert (dazu merkt sich das BIOS, wann der letzte Wechsel erfolgt ist, und geht dann davon aus, daß innerhalb eines bestimmten Zeitraums ganz bestimmt nicht noch ein Diskettenwechsel möglich ist).

- Für unsichere Fälle benötigt das BIOS die Seriennummer im Bootsektor. Wer mehrere, gleichartig formatierte Disketten mit gleicher Seriennummer abwechselnd benutzt, muß also mit Fehlern rechnen.

- Weitere Probleme sind vorprogrammiert, wenn man eine Diskette abwechselnd in zwei Rechnern benutzt. Szenario: Diskette wird aus Rechner A entnommen. Auf Rechner B wird eine neue Datei auf die Diskette geschrieben.

  Anschließend wird die Diskette wieder in das Laufwerk von Rechner A eingelegt. Das BIOS hat die Unterbrechung der Schreibschutz-Leitung entdeckt und meldet einen möglichen Medienwechsel. Daher wird die Seriennummer im Bootsektor überprüft – und für gleich befunden! Wenn jetzt Rechner A einen Schreibzugriff macht, werden mit einiger Wahrscheinlichkeit die von Rechner B veränderten Verwaltungsinformationen wieder mit den alten, vom GEMDOS gepufferten Werten überschrieben.

  Abhilfe: das BIOS zwingen, die Diskette als gewechselt anzusehen (wie zum Beispiel durch Drücken der <Esc>-Taste im Desktop). Die dazu notwendige Funktion ist in der Einführung zum GEMDOS beschrieben.

Formatieren von Disketten

Folgende Schritte sind notwendig, um eine "frische" Diskette zu erzeugen:

1. Alle Spuren beider Seiten mit "Flopfmt()" formatieren. Wenn in den ersten Sektoren Fehler auftreten, dann ist die Diskette unbrauchbar (dieser Platz wird für die Verwaltungsinformationen benötigt).

   Andere Defektstellen können theoretisch später in der FAT eingetragen werden. Bei den heutigen Diskettenpreisen lohnt sich das aber wegen der zu erwartenden verminderten Datensicherheit wohl nicht.

2. Der von Bootsektor, FATs und Wurzelverzeichnis belegte Platz muß gelöscht werden. Normalerweise sollte es reichen, dazu die ersten zwei Spuren mit "Flopwr()" zu löschen (also mit Nullen zu beschreiben).

3. Mit der BIOS-Funktion "Protobt()" wird ein geeigneter Bootsektor erzeugt. Für MS-DOS-kompatible Disketten müssen eventuell noch der Anfang des Bootsektors (im Speicher) und die Anfänge der beiden FATs (auf der Diskette) modifiziert werden.

4. Der Bootsektor wird mittels "Flopwr()" in Sektor 1 auf Spur 0, Seite 9 geschrieben.

Softwareunterstützung für "High-Density"

Für die Erkennung "High-Density"-fähiger Floppyhardware und entsprechender Betriebssystemroutinen kann der "_FDC"-Cookie benutzt werden. Ihm kann man entnehmen, was die höchste vom System unterstützte Schreibdichte ist.

Zur Zeit sind (zusätzlich zur Standardhardware) nur Werte für "High-Density" (1) und "Extra-High-Density" (2) dokumentiert.

Unterschiede ergeben sich dadurch eigentlich nur für die XBIOS-Funktionen "Floprd()", "Flopwr()", "Flopfmt()", "Flopver()" und "Protobt()". Indirekt sind damit natürlich auch die BIOS-Routinen "Rwabs()", "Getbpb()" und "Mediach()" betroffen.

Für die vier "Flop...()"-Aufrufe ergeben sich nur in Hinsicht auf die maximal erlaubte Sektorzahl Unterschiede (bei höheren Sektorzahlen wird entsprechend das richtige Format gewählt). "Protobt()" wird um neue Opcodes für Bootsektoren entsprechender Diskettentypen erweitert.

Treiber für Festplatten

Einleitung

Prinzipiell könnte das BIOS Festplatten wie ganz normale Diskettenlaufwerke behandeln. Aus verschiedenen Gründen geht es allerdings anders vor:

- Zur Markteinführung des ST war der Platz in den 192 KByte großen ROMs ausgesprochen wertvoll. Da es 1985 sowieso noch keine Festplatten für den ST zu kaufen gab, hat man dem BIOS gerade eben das Nötigste verpaßt, um den Treiber wenigstens von einer Platte booten zu können (es dauerte allerdings bis 1987, bis die Atari-Festplattensoftware dies auch ausnutzen konnte).

- Auf Festplatten paßt erheblich mehr als auf eine Diskette. Schon die ersten ST-Festplatten mit 20 MB Speicherkapazität überschritten die damals für GEMDOS maximale Mediengröße. Daß die Plattengrößen weiterwachsen würden, war leicht vorauszusehen.

Daher enthalten TOS-Versionen vor 2.00 keinen eigenen Festplattentreiber, sondern nur eine Routine zum Lesen und Ausführen des Rootsektors (Sektor 0) der Platte. TOS-Versionen ab 2.00 können über die XBIOS-Funktionen "DMAread()" und "DMAwrite()" auf Festplatten zugreifen, die fertige Einbindung als Treiber gibt es aber auch hier nicht. Da diese Routinen allerdings nicht geschwindigkeitsoptimiert sind, greifen die allermeisten Festplattentreiber (auch die von Atari) direkt auf die Hardware zu.

Die GEMDOS-spezifischen Kapazitätsgrenzen werden folgendermaßen umgangen: der verfügbare Speicherplatz des *physikalischen* Mediums wird auf mehrere *logische* Laufwerke verteilt. Diese logischen Laufwerke werden gemeinhin Partitionen genannt und haben eine für GEMDOS akzeptable Maximalgröße. Einige Vorteile dieses *Partitionierung* genannten Verfahrens:

- Kleinere Medien bedeuten für das Dateisystem weniger Arbeit und mehr Geschwindigkeit (wegen des geringeren Verwaltungsaufwands).

- Mehr Übersicht – speziell in Verbindung mit Programmen, die die Stärken eines hierarchischen Dateisystems schlecht ausnutzen (gemeint ist das alte GEM-Desktop, das nur Laufwerks-, aber keine Ordnersymbole auf dem Desktophintergrund ablegen kann).

- Nicht jede der Partionen muß auch tatsächlich ein GEMDOS-Dateisystem enthalten. Damit ist es möglich, auf einer Festplatte mit verschiedenen Betriebssystemen (Minix,

Macintosh-Emulation) zu arbeiten. Und mit Meta-DOS wäre es sogar denkbar, fremde Dateisysteme für GEMDOS zugänglich zu machen.

- Prinzipbedingt führen "Unfälle" im Dateisystem meist nur zum Dateiverlust auf einer Partition.

- Viele Festplattentreiber bieten die Möglichkeit, einzelne Partitionen gezielt gegen Schreibzugriffe zu schützen, was beim Testen neuer Software gegen unliebsame Überraschungen schützen kann.

Die Einteilung in mehrere Partitionen ist übrigens keine Erfindung von Atari. Auch unter MS-DOS und unter Unix (wenn auch dort aus anderen Gründen) gibt es Partitionen.

Bootvorgang

Das BIOS untersucht nacheinander die Rootsektoren der angeschlossenen Festplatten, und zwar beginnend mit der niedrigsten Gerätenummer (0). Beim TT werden erst die acht möglichen SCSI-Geräte und dann erst die ACSI-Geräte begutachtet.

Je nach TOS-Version werden dabei unterschiedliche Strategien benutzt:

- Bis einschließlich TOS 1.02 wird pro Gerät nur ein Leseversuch gestartet.

- In TOS 1.04 wird bei Fehlern ein zweiter Versuch gemacht – vorausgesetzt der Fehler war kein Time-Out (zu deutsch: Gerät nicht vorhanden).

- TOS 1.06 und TOS 1.62 benutzen beim Kaltstart großzügige Warteschleifen, so daß mit vielen Platten das gleichzeitige Einschalten von Rechner und Platte möglich ist.

- Ab TOS 2.05 (bzw. TOS 3.01) wird beim Kaltstart vor dem ersten Lesezugriff eine großzügige Pause von 90 Sekunden eingelegt. Damit soll den im Mega STE bzw. TT eingebauten Platten Gelegenheit gegeben werden, sich zu initialisieren (wie sollte man bei diesen Geräten auch Platte und Rechner getrennt einschalten?).

Kann ein Sektor gelesen werden, und hat die Prüfsumme (analog zum Booten von Disketten) den Wert $1234, dann wird zum ersten Byte des gelesenen Sektors gesprungen. Offizielle Unterlagen zu den Parametern dieser Routine gibt es leider nicht. Eine Analyse des vom AHDI ("Atari Hard Disk Interface") 4.0 benutzten Rootsektors ergab folgende Registerbelegung:

D7 Bits 5..7 enthalten die Nummer des ACSI-Geräts, das gerade getestet wird.

D4 Bits 0..2 enthalten die Nummer des SCSI-Geräts, das gerade getestet wird.

D3 Falls D3 die Long-Konstante "DMAr" enthält, gilt die Information aus D4 (und man darf die XBIOS-Funktion "DMAread()" aufrufen), ansonsten die aus D7.

Der im Rootsektor enthaltene Programmcode kann nun den eigentlichen Treiber nachladen, die BIOS-Vektoren und "_drvbits" passend initialisieren und dann ins BIOS zurückkehren. Alte Versionen des AHDI (vor 3.0) haben die weitere Systeminitialisierung (Laden der Programme im AUTO-Ordner, Starten von GEM und Desktop) allerdings selbst übernommen.

Wer es noch immer nicht gemerkt hat: den Festplattentreiber in den AUTO-Ordner einer Festplatte zu kopieren ist weder sinnvoll noch klug! Um die Programme im AUTO-Ordner überhaupt laden zu können, muß sowieso schon ein Festplattentreiber initialisiert sein! Normalerweise übernimmt der Rootsektor die Aufgabe, die Treiberdatei (beim AHDI: "SHDRIVER.SYS") im Wurzelverzeichnis der Bootpartition zu finden und zu laden.

Format des Rootsektors

Die Bezeichnung *Bootsektor* bezieht sich immer auf den ersten Sektor eines logischen Laufwerks. Der *Rootsektor* bezeichnet den physikalisch ersten Sektor des Mediums. Da bei einer Diskette kein Unterschied zwischen logischem und physikalischem Laufwerk gemacht wird, liegt der Bootsektor einer Diskette in Sektor 0.

Anders bei Festplatten, die in mehrere logische Laufwerke unterteilt sind: *jede* Partition einer Festplatte hat einen Bootsektor (BIOS-Sektornummer 0), die gesamte Festplatte aber nur *einen* Rootsektor.

Wie wir bereits gesehen haben, macht das BIOS keinerlei Annahmen darüber, was in einem Rootsektor steht: wenn die Prüfsumme $1234 ist, wird er ausgeführt, sonst nicht. Prinzipiell bleibt es damit den verschiedenen Festplattentreibern überlassen, ob und wie sie die Unterteilung auf verschiedene Partitionen vornehmen.

Praktikabel wäre das allerdings nicht:

- Gelegentlich möchte man neue oder andere Treiber installieren. Dann wäre es mehr als unbequem, müßte man die gesamte Platte neu einrichten.

- Alternative Betriebssysteme müssen direkt auf den Rootsektor zugreifen, um die für sie vorgesehenen Partitionen finden zu können.

– Wechselplattenbesitzer möchten ihr Medium auch auf Rechnern mit anderen Festplattentreibern ansprechen können.

Zum Glück hat Atari das vom eigenen Festplattentreiber unterstützte Format ausführlich dokumentiert, und so spricht alles dafür, dazu kompatibel zu bleiben:

Offset	Name	Belegung
$1C2	hd_siz	Gesamtzahl der 512-Byte-Sektoren als LONG-Wert
$1C6	p0_flg	Informationen über Partition 0
$1C7	p0_id	
$1CA	p0_st	
$1CE	p0_siz	
$1D2	p1_flg	Informationen über Partition 1
$1D3	p1_id	
$1D6	p1_st	
$1DA	p1_siz	
$1DE	p2_flg	Informationen über Partition 2
$1DF	p2_id	
$1E2	p2_st	
$1E6	p2_siz	
$1EA	p3_flg	Informationen über Partition 3
$1EB	p3_id	
$1EE	p3_st	
$1F2	p3_siz	
$1F6	bsl_st	Anfang der "Bad Sector List" (LONG-Wert)
$1FA	bsl_cnt	Länge der "Bad Sector List" (LONG-Wert)
$1FE		Platzhalter zur Errechnung der Prüfsumme

Atari unterscheidet zwischen *Standardpartitionen* und *Erweiterungspartitionen* ("extended partitions").

Standardpartitionen enthalten normalerweise ein "logisches" Laufwerk, während Erweiterungspartitionen dazu dienen, mehr als die vier im Rootsektor vorgesehenen Partitionen auf der Festplatte unterzubringen. Von den vier Partitionseinträgen darf nur einer der letzten drei für eine Erweiterungspartition benutzt werden. Für jede Partition steht eine zwölf Bytes große Informationsstruktur zu Verfügung (das Fragezeichen steht jeweils für die Partitionsnummer):

p?_flg (ein Byte)
Dieses Feld enthält Flags, die über einige Eigenschaften der Partition Auskunft geben:

Bit 0: Partition existiert (1) oder existiert nicht (0). Anhand dieses Bits kann man also feststellen, ob die betreffende Informationsstruktur überhaupt sinnvolle Daten enthält.

Bit 1..6: reserviert (auf Null setzen)

Bit 7: Partition ist bootbar (1) oder nicht bootbar (0). Der Atari-Festplattentreiber bootet von der ersten als bootbar gekennzeichneten Partition.

p?_id (drei Bytes)
Hier müssen drei Zeichen stehen, die den Typ der betreffenden Partition festlegen (Partitionskennung):

"GEM": Standard-GEMDOS-Partition (kleiner als 16 Megabyte)

"BGM": Große GEMDOS-Partition ("big GEMDOS partition")

"XGM": Erweiterungspartition

Andere Einträge werden von zum AHDI 3.0 kompatiblen Treibern ignoriert und können zur Kennzeichnung von Partitionen für andere Betriebssysteme benutzt werden.

p?_st (LONG)
Nummer des ersten Sektors der Partition, gerechnet ab Sektor 0.

p?_siz (LONG)
Größe der Partition in 512-Byte-Sektoren (die AHDI-Spezifikation sieht also Festplatten mit größeren Sektoren nicht vor).

Partitionstypen

Eine Standard-GEMDOS-Partition (Partitionskennung "GEM") enthält als ersten Sektor einen Bootsektor, wie man ihn schon von Disketten her kennt.

Da eine Festplattenpartition und eine Diskette aber eben doch nicht ganz vergleichbar sind, sind einige Anmerkungen zu den einzelnen Feldern notwendig:

BRA.S Wie schon erwähnt, ist der Programmcode zum Booten bei Festplatten im *Rootsektor* abgelegt. Normalerweise ist also der *Bootsektor* einer Partition für andere Zwecke frei. Viele Festplattentreiber nutzen den freien Platz, um hier

den restlichen zum Nachladen der Treiberdatei notwendigen Programmcode abzulegen.

BPS Sektorgröße in Bytes (bei Standardpartitionen immer 512 Bytes)

SPT bei Festplatten unbenutzt

NSIDES bei Festplatten unbenutzt

Frage: Wie erkennt der Festplattentreiber, ob das Medium eine 12-Bit-FAT oder eine 16-Bit-FAT hat? Gar nicht! Auf dem Atari haben Festplattenpartitionen immer nur 16-Bit-FATs (mit zwölf Bits käme man auch nicht weit). Anders bei MS-DOS: dort sind sowohl 12-Bit-FATs als auch 32-Bit-FATs möglich.

Da ältere GEMDOS-Versionen nur 32766 Cluster pro Laufwerk verwalten können und die Sektorgröße auf 512 Bytes beschränkt ist, können solche Standardpartitionen nie größer als 16 MByte werden.

Große GEMDOS-Partitionen (Partitionskennung "BGM") wurden mit AHDI 3.0 eingeführt, um die eben genannte Größenbeschränkung zu umgehen. Die maximale Kapazität einer Partition berechnet sich als das Produkt von maximaler Clusterzahl, der Anzahl von Sektoren pro Cluster und der Größe der Sektoren. Für die ersten beiden Werte zeigt sich GEMDOS leider überhaupt nicht kompromißbereit, und so mußte Atari auf größere Sektoren ausweichen.

Da man sich allerdings meistens nicht aussuchen kann, wie groß die Sektoren auf der Festplatte sein sollen (fast immer sind es 512 Bytes), täuscht der Festplattentreiber dem BIOS einfach die größeren Sektoren vor. Dazu faßt er einfach immer mehrere physikalische Sektoren zu einem logischen Sektor zusammen.

Bei BGM-Partitionen ist der Bootsektor also wie folgt geändert:

BPS Sektorgröße in Bytes (512, 1024, 2048, 4096 oder 8192 Bytes)

Bei "großen" Sektoren wird der "Rest" des Bootsektors (also alles bis auf die ersten 512 Bytes) mit Nullen aufgefüllt und hat weiter keine Bedeutung.

Damit ist es allerdings noch nicht getan: auch GEMDOS muß darüber informiert werden, wie groß ein Sektor maximal werden kann – schließlich hat es dafür eigene Pufferlisten, die entsprechend modifiziert sein wollen (mehr dazu in der Einleitung zum GEMDOS unter "GEMDOS-Puffer"). Eigene Versuche ergaben übrigens, daß GEMDOS 8192 Bytes große Sektoren gar nicht gern mag (ausprobieren!).

Für die veränderte Partitionskennung ("BGM" anstelle von "GEM") gibt es eigentlich keinen echten Bedarf: alle nötigen Informationen lassen sich schließlich aus dem Bootsektor herauslesen. Genauso verfährt auch AHDI 3.0: für ihn ist es völlig egal, ob die Partitionskennung "GEM" oder "BGM" ist.

Doch was würde passieren, ließe man einen "alten" Treiber auf die Festplatte los? Richtig: er würde wahrscheinlich die angegebene Sektorgröße ignorieren und einen ungenießbaren Datensalat fabrizieren.

Daher sollte man die Partitionskennungen wie folgt interpretieren:

"GEM": für alle Festplattentreiber nutzbar

"BGM": Festplattentreiber muß mit größeren logischen Sektoren zurechtkommen (also kompatibel zu AHDI 3.0 sein)

Eine weitere Einschränkung älterer Festplattentreiber war, daß sie nur maximal vier Partitionen unterstützen konnten. Kombiniert mit der Größenbeschränkung der einzelnen Partitionen war damit schon bei etwa 64 MByte großen Festplatten Schluß. Das war auch 1987 schon eine Einschränkung.

An dieser Stelle kommen die *Erweiterungspartitionen* ins Spiel. Bevor ich zur Struktur dieses Partitionstyps komme, kann ich mir eine Bemerkung nicht verkneifen: Selbst nach dem Schreiben eines AHDI-3-kompatiblen Festplattenpartitionierers ("SCSI-Tool") verstehe ich noch immer nicht, warum das Format gerade so aussieht, wie es aussieht.

Der Grundgedanke läuft darauf hinaus, daß alle zusätzlichen Partitionen in Form einer verketteten Liste organisiert werden. Damit erlaubt das Format prinzipiell beliebig viele Partitionen.

Eine Erweiterungspartition ist eine Partition, die aus mehreren *Teilpartitionen* bestehen kann. Jede dieser Teilpartitionen besteht aus einem *Hilfs-Rootsektor* und einer *Standardpartition* (wie sie oben beschrieben ist). Das Format eines Hilfs-Rootsektors ähnelt stark dem eines normalen Rootsektors. Einschränkungen sind:

- Die Felder "hd_siz", "bsl_st" und "bsl_cnt" sind nicht benutzt.

- Maximal zwei der Partitionsinformationsstrukturen dürfen benutzt sein (und müssen direkt aufeinander folgen). Die anderen beiden Einträge müssen mit Nullen gefüllt sein.

Die erste der beiden benutzten Informationsstrukturen muß für eine Standardpartition stehen.

Dabei gelten für einige der Felder leicht modifizierte Bedeutungen:

p?_flg: Nur Bit 0 ist benutzt (und muß gesetzt sein), bootbare Partitionen können nur im "echten" Rootsektor installiert werden. Alle weiteren Bits sind reserviert.

p?_id: Hier darf nicht "XGM" stehen.

p?_st: Startsektor der Partition relativ zu diesem Hilfs-Rootsektor.

Der folgende Eintrag darf leer sein (dann ist das Ende der verketteten Liste erreicht). Anderenfalls muß er einen Verweis auf den nächsten Hilfs-Rootsektor enthalten. Die einzelnen Felder sehen dann so aus:

p?_flg: Nur Bit 0 ist benutzt (und muß gesetzt sein). Alle weiteren Bits sind reserviert.

p?_id: Muß "XGM" enthalten.

p?_st: Startsektor der nächsten Teilpartition *relativ zum ersten Sektor der Erweiterungspartition* (!).

p?_siz: Größe der nächsten Teilpartition (normalerweise Partitionsgröße plus ein Sektor für den Hilfs-Rootsektor).

Tip: Wer AHDI-3-kompatible Software schreiben will, sollte eine Weile mit dem Festplattenpartitionierer "HDX" experimentieren und sich die erzeugten Rootsektoren genau ansehen!

Bad sector list

Festplatten tendieren schon wegen ihrer Größe dazu, die eine oder andere defekte Stelle aufzuweisen. Das gilt sogar für fabrikneue Platten.

Harddisks nach dem ST506-Standard werden von außen durch Angabe von physikalischen Positionen auf der Platte gesteuert – also Zylinder, Kopf, Spur etc. (ganz ähnlich wie bei einem Diskettenlaufwerk). Die Hersteller solcher Platten liefern in der Regel eine Liste der Defektstellen mit. In den Atari-Festplatten SH 204, SH 205, Megafile 20, Megafile 30 und Megafile 60 ist das ST506-Laufwerk allerdings indirekt über ein spezielles Controllerboard angeschlossen, so daß man sich nicht um die Geometrie der Platte zu kümmern braucht und direkt mit fortlaufend numerierten Sektoren arbeiten kann.

Der von Atari eingesetzte Controller erlaubt es prinzipiell, defekte Sektoren "auszumappen". Das heißt, daß der Controller intern eine Liste der defekten Stellen kennt und bei der Numerierung der Sektornummern selbsttätig ausfiltert.

Damit könnte man die Verwaltung der defekten Stellen eigentlich auf die Hardware abwälzen. Ob dies auch tatsächlich funktioniert, wird wohl vorläufig ungeklärt bleiben, da die Atari-Festplattensoftware von dieser Möglichkeit sowieso keinen Gebrauch macht und mittlerweile SCSI-Platten nicht nur im Mega STE und im TT weite Verbreitung gefunden haben.

SCSI-Platten haben es nämlich beim Defektmanagement sehr viel einfacher. Der Hersteller testet die Platte und sorgt auf Controller-Ebene für das Ausmappen der Sektoren. De facto erhält der Kunde also eine hundertprozentig fehlerfreie Platte.

Auch wenn neue Defekte auftreten, steht der SCSI-Controller hilfreich zur Seite. Die neuen Fehlerstellen werden einfach in die Defektliste aufgenommen – und schon sind sie "verschwunden". Manche Plattentypen haben sogar eigens dafür vorgesehene Reservesektoren und -spuren; einige gehen sogar so weit, das Aus- und Um-Mappen bei Bedarf im laufenden Betrieb vorzunehmen.

Leider nutzt Ataris "HDX" diese Möglichkeit nicht aus, obwohl das bei den in der Megafile 44 und den im Mega STE und TT integrierten Festplatten möglich wäre. Anders steht es bei Festplattentools, die von Fremdherstellern geliefert werden (zum Beispiel das schon oben erwähnte "SCSI-Tool" von Hard & Soft).

"HDX" geht einen anderen Weg: in der "Bad Sector List" – einem speziell dafür vorgesehenem Bereich auf der Platte – werden getrennte Listen für "Hersteller"-Defekte und neu aufgetretene Defekte geführt. Beim Umpartitionieren der Platten werden die defekten Sektoren dann auf Ebene der FAT (also auf Dateisystem-Ebene) unschädlich gemacht.

Wer ein Festplatten-Utility schreiben will, das diese "Bad Sector List" unterstützt, sollte sich beim Atari-Entwickler-Support die "AHDI Release Notes" besorgen. Wem es nur nur darauf ankommt, die Platte so zu partitionieren, daß sie mit "HDX" weiterbearbeitet werden kann: als Länge der "Bad Sector List" den Wert 1 eintragen, den betreffenden Sektor vollständig mit Nullen füllen, und in das vierte Byte den Wert $A5 eintragen (eine leere "Bad sector list" wird von "HDX" nicht akzeptiert)!

Andere Plattenformate

Das hier angegebene AHDI-3-Format ist erst 1989 von Atari eingeführt worden. Vorher gab es weder Erweiterungspartitionen noch größere Sektoren.

Daneben wurden auch noch weitere Felder im Rootsektor benutzt, um Informationen über die Plattengeometrie (Zylinder, Köpfe, Spuren) bereitzustellen. Diese Felder sind nicht offiziell dokumentiert und zu allem Überfluß ohnehin zu nichts zu gebrauchen:

- Durch die Fehlerkorrekturfähigkeiten moderner SCSI-Platten ist es sowieso nicht mehr möglich, von logischen Sektornummern auf spezielle Positionen auf dem Medium zu schließen.

- Viele SCSI-Platten benutzen eine variable Zahl von Sektoren pro Spur, um die äußeren Spuren der Platte besser nutzen zu können.

- Die beschriebenen Felder belegen wertvollen Platz im Rootsektor, der unter Umständen für Programmcode benötigt wird.

Daher sollte man sich auf keinen Fall darauf verlassen, an diesen Stellen des Rootsektors sinnvolle Daten vorzufinden.

Beim alten Atari-Format war die maximale Plattengröße auf 64 MByte festgelegt. Kein Wunder, daß viele Hersteller andere Formate ersonnen haben, um mehr Partitionen auf einer Platte unterbringen zu können.

Das heute von Atari benutzte Format hat allerdings zwei wichtige Vorteile:

- Es ist vollständig zum alten Format kompatibel – selbst dann, wenn man versehentlich einen alten Treiber startet.

- Es werden keine neuen Einträge im Rootsektor "verbraucht".

Der Rootsektor von MS-DOS-Festplatten sieht deutlich anders aus. Dies könnte einem ziemlich egal sein, gäbe es nicht Wechselplatten. AHDI enthält tatsächlich Code, der die Benutzung von unter MS-DOS partitionierten Syquest-Wechselmedien erlaubt.

Das allerdings mit folgenden Einschränkungen:

- Dieses "Feature" ist nicht offiziell dokumentiert – meist ein Zeichen dafür, daß es noch ungelöste Probleme gibt.

- Genau diese Probleme könnten damit zu tun haben, daß MS-DOS-Wechselplatten oft Cluster zu vier Sektoren verwenden – was GEMDOS laut Dokumentation gar nicht mag.

Die PUNINFO-Struktur

Zum AHDI 3.0 kompatible Festplattentreiber installieren eine Struktur, die viel Wissenswertes über den Treiber und die Platte enthält:

```
typedef struct
{
     WORD      puns;            /* Anzahl der Geräte */
     BYTE      pun[16];         /* diverse Flags */
     LONG      part_start[16];  /* Partitionsanfänge */
     LONG      P_cookie;        /* muß "AHDI" sein */
     LONG      *P_cookptr;      /* zeigt auf vorheriges Element */
     UWORD     P_version;       /* 0x0300 oder größer */
     UWORD     P_max_sector;    /* maximale Sektorgröße */
     LONG      reserved[16];
} PUN_INFO;
```

Zunächst fällt auf, daß die Felder innerhalb der Struktur nur Platz für 16 Einträge haben (obwohl das BIOS maximal 32 Geräte unterstützt). Dadurch kann man AHDI-3-kompatibel nur maximal 16 Gerätenummern verwalten.

"puns" gibt an, für wie viele Geräte sich der Festplattentreiber zuständig fühlt. In der "pun"-Tabelle befinden sich nähere Informationen zu den BIOS-Geräten 0 bis 15:

Bit 0..2: Gerätenummer der Festplatte.

Bit 3: Bei gesetztem Bit handelt es sich um ein Gerät an der SCSI- Schnittstelle (sonst: ACSI). Dieses Bit ist zwar noch nicht offiziell dokumentiert, kann aber wohl ohne Vorbehalt benutzt werden.

Bit 4..6: Reserviert.

Bit 7: Wenn dieses Bit gesetzt ist, dann wird das BIOS-Gerät nicht vom Plattentreiber verwaltet (gilt beispielsweise für die beiden Diskettenlaufwerke auf den Gerätenummern 0 und 1).

"part_start" enthält für jede (unterstützte) Gerätenummer die Nummer des Startsektors auf der betreffenden Festplatte. "P_cookie" ist ein Magic-Wert, mit dem man sich vergewissern sollte, ob man tatsächlich eine gültige PUN_INFO-Struktur vor sich hat.

Gleiches gilt für "P_cookptr" (der auf das vorherige Strukturelement zeigen sollte) und die Versionsnummer in "P_version".

Näheres zu "P_max_sector" finden Sie in der GEMDOS-Einleitung (Thema: GEMDOS-Puffer). Einige Felder dieser Struktur sind erst dann ausgefüllt, wenn ein "Getbpb()"-Aufruf für das Laufwerk stattgefunden hat.

Einen Zeiger auf diese Struktur erhält man über die Systemvariable "pun_ptr":

```
/* Zeiger auf PUN_INFO-Struktur ermitteln.
   Liefert entweder den Zeiger oder im Fehlerfall 0 zurück */

PUN_INFO *getpun (void)
{
     PUN_INFO *P;
     LONG oldstack;

     oldstack = Super (0L);
     P = *((PUN_INFO **)(0x516L));
     Super ((void *)oldstack);

     if (P)  /* überhaupt gesetzt? */
          if (P->P_cookie == 0x41484449L) /* Cookie gesetzt? */
               if (P->P_cookptr == &(P->P_cookie))
                    if (P->P_version >= 0x300)
                         return P;
     return 0L;
}
```

Unterstützung von Wechselplatten

Wechselplatten machen den Festplattentreibern auf vielfältige Art und Weise das Leben schwerer:

1. Die Medien können gewechselt werden.

2. Nicht jedes Medium muß die gleiche Anzahl von Partitionen besitzen.

3. Unter Umständen will man später ein Medium mit größeren Sektoren als vorher anmelden.

Problem 1 läßt sich relativ leicht lösen, da das BIOS sowieso schon den Begriff "Mediachange" kennt und die Erkennung eines SCSI-Medienwechsels vergleichsweise einfach ist.

Problem 2 hat zwei Seiten. Einerseits kann es passieren, daß das neu eingelegte Medium *weniger* Partitionen hat als das beim Booten eingelegte. Was passiert nun mit der freigewordenen Laufwerksnummer?

Nichts, genausowenig wie bei einem Diskettenlaufwerk ohne Diskette: "Getbpb()" liefert, ebenso wie "Mediach()" und "Rwabs()", auf Anfrage eine Fehlermeldung (0).

Der umgekehrte Fall ist schwieriger: was passiert, wenn das neu eingelegte Medium mehr Partitionen hat als das vorherige? Atari ist auf eine Verlegenheitslösung ausgewichen: man meldet dem Treiber bei der Installation, wie viele Laufwerkskennungen er bei einer Wechselplatte freihalten soll (wer hat eine bessere Lösung?). Die zusätzlichen Laufwerkskennungen sind dann in den "_drvbits" angemeldet, können aber erst nach Einlegen eines passenden Mediums angesprochen werden (ähnlich wie bei nicht eingelegten Disketten).

Dazu – und das ist weniger elegant – muß man an der Treiberdatei herumpatchen (das Format ist im Anhang beschrieben). Andere Festplattentools (wie zum Beispiel das mittlerweile vielzitierte "SCSI-Tool") machen es dem Anwender mittels Dialogbox und Mausbedienung einfacher.

Bei Problem 3 ist es ähnlich: da GEMDOS die maximale Sektorgröße für seine Pufferlisten kennen muß, ist eine nachträgliche Änderung schwierig. Einfacher ist es, dem Treiber von vornherein die maximal nötige Sektorgröße mitzuteilen. Auch dieser Wert kann beim AHDI in der Treiberdatei eingetragen werden (und wie immer geht es bei "SCSI-Tool" komfortabler). Weitere Informationen zu diesem Thema bei der Erörterung der GEMDOS-Pufferlisten in der GEMDOS-Einführung.

Welche Partition bekommt welche Gerätenummer?

Oft ist es nicht ganz einfach zu verstehen, wann ein Treiber die Partitionen welcher Platte unter welchen Laufwerksnamen anbietet: AHDI ist relativ clever und sucht selbsttätig auch auf mehreren angeschlossenen Platten nach anzusteuernden Partitionen.

Dazu sucht er, immer beginnend bei Gerätenummer 0, nach Festplatten (beim TT werden zuerst die SCSI- und dann die ACSI-Geräte untersucht). Die Suche bricht ab, wenn eine Gerätenummer nicht belegt ist.

Das heißt: Festplatten müssen, immer mit 0 anfangend, aufsteigende Gerätenummern aufweisen, Lücken sind nicht erlaubt. Damit ist es möglich, daß der Treiber zwar korrekt von einer unter ACSI-Nummer 1 laufenden Platte gebootet wird, dann aber keine Partitionen findet (weil die Suche mit Gerät 0 startet und auf dieser Nummer keine Platte reagiert).

Andere Festplattentreiber sind da flexibler. Oft sind nicht nur Lücken, sondern auch Vertauschungen in der Gerätereihenfolge erlaubt. Damit kann man beispielsweise Partitionen auf Gerät 1 niedrigere BIOS-Gerätenummern zuordnen als denen auf Gerät 0.

Meta-DOS

"Meta-DOS" erweitert XBIOS um ein Treiberkonzept für blockorientierte Geräte. Dafür hat Atari die Funktionsnummern 48 bis 63 reserviert.

Meta-DOS erlaubt die Installation von Treibern für 26 blockorientierte Geräte. Ihre Funktionen werden über neue XBIOS-Aufrufe (insbesondere zum Öffnen und Schließen der Geräte und zum Lesen und Schreiben von Blöcken) bereitgestellt.

Die Kommunikation von GEMDOS wird durch spezielle *logische* Gerätetreiber hergestellt (mehr dazu in der Einführung zum GEMDOS).

Meta-DOS ist eine Betriebssystemerweiterung, die sich noch in Entwicklung befindet. Zur Zeit sind folgende Treiber verfügbar:

- ein physikalischer Treiber für die CD-ROM-Laufwerke CDAR 504/505

- logische Treiber für die beiden in Verbindung mit CD-ROMs gebräuchlichen Formate ("High Sierra" und ISO).

In der Funktionsreferenz finden Sie die Dokumentation zu "Metainit()". Weitere Informationen finden Sie in den Meta-DOS-Entwicklerunterlagen.

Speicherverwaltung

Bei der Speicherverwaltung hat das BIOS wenig zu tun. Es hat lediglich die Aufgabe, beim Systemstart den vorhandenen Arbeitsspeicher zu finden und zu initialisieren. Über die Funktion "Getmpb()" wird GEMDOS die Möglichkeit gegeben, eine erste TPA (mehr dazu im GEMDOS-Kapitel) anzulegen.

BIOS- und XBIOS-Bindings

BIOS und XBIOS nehmen ihre Parameter auf dem Stack entgegen. Dabei wird das letzte Argument aus der Parameterliste als erstes auf den Stack gelegt. Schließlich wird mit Hilfe der Anweisung "TRAP #13" der BIOS-Trap-Dispatcher bzw. mit "TRAP #14" der XBIOS-Dispatcher aktiviert.

Funktionsergebnisse liefern BIOS und XBIOS im Prozessorregister D0 zurück. Nur die Register D3-D7 und A3-A7 werden gerettet. Alle anderen können durch den Aufruf verändert

werden! Das Verhalten für illegale Funktionsnummern ist bis auf einige wenige Ausnahmen ("Blitmode()" und "Bconmap()") nicht definiert.

BIOS und XBIOS sind bis zu einem gewissen Maß re-entrant.Damit sind auch Aufrufe aus Interrupts oder von innerhalb von BIOS oder XBIOS möglich. Dabei sind allerdings einige wichtige Einschränkungen zu beachten:

Der TRAP-Dispatcher benutzt einen festen Speicherbereich, um Registerinhalte, Statusregister und Rücksprungadressen zu retten. Dieser Bereich bietet laut Atari nur für maximal drei Rekursionsstufen Platz. Als Zeiger auf diesen Speicherbereich dient die Systemvariable "savptr".

Ein anderes Problem resultiert daraus, daß ein Interrupt just zu dem Zeitpunkt auftreten kann, in dem der Dispatcher Register speichert oder wiederherstellt. Eigentlich sollte der Dispatcher während dieses Vorgangs alle Interrupts sperren – doch tut er es leider nicht.

Für BIOS- oder XBIOS-Aufrufe aus einem Interrupt heraus muß man also so vorgehen:

```
; BIOS- oder XBIOS-Aufrufe aus dem Interrupt
; Frei nach: The Hitchhiker's Guide to the BIOS

savptr =  $4A2      ; Zeiger auf Registerpuffer
amount = 46         ; soviel Bytes werden verbraucht

im_interrupt:
     sub.l     #amount, savptr     ; Platz schaffen

; hier dürfen BIOS- bzw. XBIOS-Aufrufe erfolgen
     add.l     #amount, savptr     ; ...und wieder freigeben

; und raus aus dem Interrupt
     rte
```

Wichtige Einschränkung: *nur ein einziger* Interrupt-Handler darf gleichzeitig so verfahren!

Fehlermeldungen

0: E_OK ("OK (no error)")

Funktion erfolgreich ausgeführt (kein Fehler aufgetreten).

–1: ERROR ("Error")

Es ist ein Fehler aufgetreten, der nicht genauer spezifiziert werden kann.

–2: EDRVNR ("Drive not ready")

Angesprochenes Gerät ist nicht angeschlossen, nicht funktionsbereit oder reagiert nicht innerhalb der gesetzten Frist (Timeout).

–3: EUNCMD ("Unknown command")

Dem angesprochenen Peripheriegerät ist das gegebene Kommando unbekannt.

–4: E_CRC ("CRC error")

Beim Lesen eines Sektors ist ein Fehler aufgetreten (aufgetreten bei der CRC-Überprüfung).

–5: EBADRQ ("Bad request")

Das Peripheriegerät kann das Kommando nicht ausführen. Befehlsparameter und Kontext prüfen!

–6: E_SEEK ("Seek error")

Der angesprochene Track konnte vom Laufwerk nicht erreicht werden.

–7: EMEDIA ("Unknown media")

Leseversuch gescheitert, da das Medium keinen korrekten Bootsektor besitzt.

–8: ESECNF ("Sector not found")

Der betreffende Sektor wurde nicht gefunden.

–9: EPAPER ("Out of paper")

Drucker nicht betriebsbereit.

–10: EWRITF ("Write fault")

Fehler bei Schreiboperation aufgetreten.

–11: EREADF ("Read fault")

Fehler bei Leseoperation aufgetreten.

–12: EGENRL ("General error")

Allgemeiner Fehler (Kommentar im "Hitchhiker's guide to the BIOS": "Reserved for future catastrophes").

–13: EWRPRO ("Write on write-protected media")

Es wurde versucht, auf ein schreibgeschütztes Medium zu schreiben.

–14: E_CHNG ("Media change detected")

Seit der letzten Schreiboperation wurde das Medium gewechselt.

–15: EUNDEV ("Unknown device")

Das angesprochene Gerät ist dem Betriebssystem unbekannt.

–16: EBADSF ("Bad sectors on format")

Beim Formatiervorgang wurden defekte Sektoren entdeckt.

–17: EOTHER ("Insert other disk request)"

Eine andere Diskette muß eingelegt werden. Tritt nur auf, wenn Laufwerk B: angesprochen wird, ohne angeschlossen zu sein. In diesem Fall wird der Benutzer aufgefordert, "Diskette B:" in das erste Laufwerk einzulegen.

–18: EINSERT ("Insert disk")

Meta-DOS-Fehler: Medium einlegen!

–19: EDVNRSP ("Device not responding")

Meta-DOS-Fehler: Gerät antwortet nicht.

Der OS-Header

Der OS-Header ist eine Struktur, die viele wichtige Informationen über die TOS-Version sowie einige Zeiger auf andere Systemadressen enthält. Je nach Betriebssystemversion liegt er am Anfang des ROMs oder auch im RAM. Seine Adresse kann über die Systemvariable "_sysbase" ermittelt werden. Und so sieht er aus:

```
typedef struct _osheader
{
     UWORD      os_entry;       /* BRAnch-Instruktion zum RESET-
                                   Handler, Offset $00 */
     UWORD      os_version;     /* TOS-Versionsnummer, Offset $02*/
     void       *reseth;        /* Zeiger auf RESET-Handler,
                                   Offset $04 */
     struct     _osheader
                *os_beg;        /* Basisadresse Betriebssystem,
                                   Offset $08 */
     void       *os_end;        /* Erstes nicht vom OS benutztes
                                   Byte, Offset $0C */
     LONG       os_rsv1;        /* reserviert, Offset $10 */
     GEM_MUPB   *os_magic;      /* Zeiger auf "GEM memory usage
                                   parameter block", Offset $14 */
     LONG       os_date;        /* TOS-Herstellungsdatum im BCD-
                                   Format, etwa $02061986 für
                                   6. Februar 1986, Offset $18 */
     UWORD      os_conf;        /* verschiedene Konfigurationsbits,
                                   Offset $1C */
     UWORD      os_dosdate;     /* TOS-Herstellungsdatum im GEMDOS-
                                   Format, Offset $1E */

     /* die folgenden Felder erst ab TOS-Version 1.02 */
     char       **p_root;       /* Basisadresse des GEMDOS-Pools,
                                   Offset $20 */
     BYTE       **pkbshift;     /* Zeiger auf BIOS-interne Variable
                                   für den aktuellen Wert von
                                   "Kbshift()", Offset $24 */
     BASEPAGE   **p_run;        /* Adresse der Variablen, die einen
                                   Zeiger auf den aktuellen GEMDOS-
                                   Prozeß enthält, Offset $28 */
     char       *p_rsv2;        /* reserviert, Offset $2C */
} OSHEADER;
```

os_entry (Offset 0)
Diese beiden Bytes enthalten ein Sprungbefehl zum Anfang der Betriebssysteminitialisierung.

os_version (Offset 2)
Dies ist die TOS-Versionsnummer (TOS steht übrigens entgegen immer wieder verbreiteten, anders lautenden Gerüchten für "The Operating System"). Folgende Betriebssystemversionen waren zum Zeitpunkt der Drucklegung bekannt:

TOS 0.00 (os_version enthält $0000) war das BOOT-ROM, das mit den allerersten STs ausgeliefert wurde. Es war nur 64 KByte groß und diente in erster Linie zum Nachladen des vollständigen TOS von Diskette. Atari-interne Bezeichnung: "Das Boot".

TOS 1.00 (os_version: $0100) war das erste "fertige" TOS, das zunächst auf Diskette ("RAM-TOS") und ab Februar 1986 auf ROMs ausgeliefert wurde ("ROM-TOS").

TOS 1.02 (os_version: $0102) enthält gegenüber der Vorgängerversion Routinen zur Nutzung des Blitters und trägt daher den Namen "Blitter-TOS". Zuerst wurde es im Mega ST gesichtet, später dann aber standardmäßig auch in alle anderen Rechner eingebaut. Ebenso wie TOS 1.00 enthält es die GEMDOS-Version 0.13, die sich durch besonders große Unzuverlässigkeit auszeichnet.

Seit 1989 gibt es TOS 1.04 (os_version: $0104). Wegen des im Farbmodus mit Regenbogenfarben unterlegten Atari-Symbols im Desktop nennt es Atari "Rainbow-TOS". TOS 1.04 enthält ein verbessertes GEM (1.4) und ein halbwegs brauchbares GEMDOS (0.15) und ist die aktuellste in ST und Mega ST einsetzbare Version (wegen der Beschränkung auf 192 KByte ROM).

TOS 1.06 und TOS 1.62 (os_version: $0106 bzw. $0162) sind die in den Rechnern der STE-Serie benutzten TOS-Versionen. TOS 1.06 ist bis auf einige Anpassungen mit TOS 1.04 identisch. Version 1.62 enthält die verbesserte GEMDOS-Version 0.17.

TOS-Versionen mit einer 2 vor dem Dezimalpunkt sind für Rechner der Mega-STE-Linie reserviert. Die bislang erste und einzige Version ist TOS 2.05 (os_version: $0205).

Betriebssystemversionen für den TT/030 erkennt man an der 3 vor dem Dezimalpunkt.Die ersten ausgelieferten TTs enthielten TOS 3.01 (os_version: $0301). Seit Anfang 1991 wird statt dessen das um einige kleinere Fehler korrigierte TOS 3.05 ausgeliefert.

Originalton Atari zu TOS-Versionsnummern (Pexec-Cookbook): "Using the OS version number to determine the location of undocumented variables is not acceptable. You have been warned."

reseth (Offset 4)
Dieser LONG-Wert zeigt auf den Beginn des Reset-Handlers.

os_beg (Offset 8)
Dies ist ein Zeiger, der auf die Basisadresse des Betriebssystems zeigt. Dies kann, muß aber nicht die gleiche Adresse sein, auf die auch "_sysbase" zeigt.

os_end (Offset 12)
Zeigt auf das Ende des für BIOS, XBIOS und GEMDOS benötigte RAM.

os_magic (Offset 20)
Über diesen Zeiger erhält das BIOS Informationen über das Vorhandensein und die Speicherbedürfnisse von GEM.

"os_magic" zeigt auf einen "GEM memory usage parameter block" folgender Gestalt:

```
typedef struct
{
     LONG gm_magic;  /* muß 0x87654321 sein */
     void *gm_end;   /* Ende des von GEM benötigten
                        Speicherbereichs */
     void *gm_init;  /* Startadresse von GEM */
} GEM_MUPB;
```

Diese Werte werden vom BIOS bei der Systeminitialisierung ausgewertet. Wenn "gm_magic" den richtigen Wert hat, wird "gm_end" als "end_os" und "gm_init" als "exec_os" eingesetzt.

Anderenfalls wird für "end_os" der Wert aus "os_end" und für "exec_os" der Reset-Handler eingesetzt.

Zusammen mit der XBIOS-Funktion "Puntaes()" (löscht, falls im RAM, "gm_magic") und "_cmdload" ist es damit bei einer RAM-TOS-Version möglich, ohne GEM zu booten.

Dabei werden allerdings nur der von den GEM-eigenen Variablen, nicht aber der durch den Programmcode belegte Speicherplatz eingespart.

os_date (Offset 24)
TOS-Herstellungsdatum im BCD-Format. Einige typische Werte für deutschsprachige TOS-Versionen:

Version	os_date	Datum
1.00	$02061986	6. Februar 1986
1.02	$04221987	22. April 1987
1.04	$04061989	6. April 1989
1.06	$07291989	29. Juli 1989
2.05	$12051990	5. Dezember 1990
3.01	$08291990	29. August 1990
3.05	$12051990	5. Dezember 1990

os_conf (Offset 28)
Das unterste Bit enthält das NTSC/PAL-Flag (Bit gesetzt: PAL-Videosystem). In den restlichen Bits befindet sich eine Länderkennung, die dazu dienen kann, in kleineren Programmen automatisch die benutzte Sprache auszuwählen:

Land	Kürzel	Nummer
USA	USA	0
Bundesrepublik Deutschland	FRG	1
Frankreich	FRA	2
Großbritannien	UK	3
Spanien	SPA	4
Italien	ITA	5
Schweden	SWE	6
Schweiz (französisch)	SWF	7
Schweiz (deutsch)	SWG	8
Türkei	TUR	9
Finnland	FIN	10
Norwegen	NOR	11
Dänemark	DEN	12
Saudi-Arabien	SAU	13
Niederlande	HOL	14

Leider gibt es einen kleinen Haken: Um einen Fehler in sehr alten AHDI-Versionen zu umgehen, legt TOS ab Version 1.02 einen OS-Header im RAM an und kopiert dort "os_dosdate" in "os_conf". Vor dem Start der AUTO-Ordner-Programme wird dann wieder der "richtige" Wert eingetragen.

Nun ist es aber so, daß alte AHDI-Versionen die restliche Systeminitialisierung selbst übernehmen und daher nie wieder der richtige OSHEADER installiert wird. Abhilfe: Bei Zugriffen auf "os_conf" immer erst einmal indirekt über "os_beg" gehen, also zum Beispiel:

```
LONG savessp = Super (0L);
OSHEADER *O = *((OSHEADER **)(0x4f2L));
Super ((void *) savessp);

O = O->os_beg;    /* wegen eines Fehlers in einer alten AHDI-
                     Version */
```

So kommt man mit allen TOS- und AHDI-Versionen an den richtigen Wert.

os_dosdate (Offset 30)
Enthält die gleiche Information wie "os_date" – nur im GEMDOS-Format. Wird bei Rechnern ohne Hardware-Uhr als Startwert für die GEMDOS-Uhr benutzt.

p_root (Offset 32)
Zeigt auf die "Memory free list" des GEMDOS (in TOS 1.00 bei Adresse $56FA). Wird von "FOLDR100.PRG" benutzt, um den GEMDOS-Pool zu erweitern.

pkbshift (Offset 36)
Zeigt auf eine ein Byte große Variable, die den aktuellen Zustand der Umschalttasten (Shift, Control etc.) enthält (in TOS 1.00 bei Adresse $E1B). Dies ist der Wert, der von der BIOS-Funktion "Kbshift()" benutzt werden.

"pkbshift" sollte nur dann benutzt werden, wenn der Aufruf der BIOS-Funktion nicht möglich ist – beispielsweise in zeitkritischen Interrupt-Routinen.

p_run (Offset 40)
Zeigt auf einen GEMDOS-internen Zeiger, in dem die Adresse der Basepage des aktuell ausgeführten GEMDOS-Prozesses steht. Mehr dazu in der GEMDOS-Einleitung.

Systeminitialisierung

Für viele Projekte ist es interessant zu wissen, was genau bei der Initialisierung von TOS vor sich geht. Die Thematik wird allerdings dadurch verkompliziert, daß es erhebliche Unterschiede zwischen den einzelnen TOS- und Hardware-Versionen gibt und die aktuellste offizielle Dokumentation dazu ("Hitchhiker's Guide to the BIOS") noch aus dem Jahre 1985 stammt.

Im Anfang werden Stack-Pointer und Programmzähler mit den Werten aus den Speicherzellen 0 bis 7 geladen. Diese Adressen enthalten eine Kopie der ersten Bytes des ROMs. Der Wert des Stack-Pointers ist dabei eher zufällig, denn es handelt sich um die ersten vier Bytes der OSHEADER-Struktur (also os_entry und os_version).

Als nächstes werden mittels "move #$2700,sr" alle Interrupts gesperrt und eine "reset"-Instruktion ausgeführt. Sie dient dazu, alle angeschlossenen Peripheriebausteine zu initialisieren.

Wenn eine Diagnose-Cartridge eingelegt ist (dann enthält Adresse $00FA0000 den Wert $FA52235F), wird A6 mit einer Rücksprungadresse geladen und in den Code der Cartridge verzweigt (Sprung zu Adresse $00FA0004).

Nun folgt der Test auf "Warmstart". Wenn die einzelnen Validierungsregister ("memvalid", "memval2" etc.) die richtigen Werte enthalten, wird von einem "Warmstart" ausgegangen. Das heißt insbesondere, daß kein neuer Speichertest durchgeführt wird.

Als nächstes werden die im Reset-Vektor eingehängten Funktionen ausgeführt. Dazu wird "resvalid" auf den korrekten Wert hin überprüft, A6 mit einer Rücksprungadresse geladen und über "resvector" gesprungen. Wie man hier eigene Funktionen installieren sollte, wird im Anschluß an diesen Abschnitt beschrieben.

Es folgt die Initialisierung verschiedener Hardwarebausteine (PSG, Shifter, DMA-Sound). Wenn es sich nicht um einen Warmstart handelt, wird anschließend die Größe des verfügbaren RAMs festgestellt, der gefundene Speicher gelöscht, und die Validierungsregister ("memvalid" etc.) werden initialisiert. Weitere Details zum Speichertest finden Sie im Hardwareteil.

Nun wird der Beginn des Arbeitsspeichers gelöscht, die meisten Systemvariablen und der Cookie Jar initialisiert, der Bildspeicher am Ende des freien Arbeitsspeichers installiert, die Interrupt-Vektoren gesetzt und schließlich die BIOS-Routinen initialisiert.

Cartridge-Software mit gesetztem Bit 26 in CA_INIT wird initialisiert und gestartet (mehr zur Behandlung von Cartridges im Hardwareteil). Anschließend wird (je nach Monitortyp) der entsprechende Shifter-Modus gesetzt, der Bildschirm initialisiert und gegebenenfalls Cartridge-Software mit gesetztem Bit 24 in CA_INIT aufgerufen.

Der Interrupt-Level wird auf 3 gesetzt, daß heißt, alle Interrupts bis auf den Horizontal-Blank-Interrupt werden eingeschaltet. Es folgt der Aufruf von Cartridge-Software mit gesetztem Bit 25 in CA_INIT. GEMDOS wird initialisiert (das heißt: Standardkanäle werden gesetzt, interne Strukturen werden initialisiert, Speicherverwaltung wird übernommen etc.).

Cartridge-Software mit gesetztem Bit 27 in CA_INIT wird initialisiert und gestartet. Nun wird versucht, von der Floppy zu booten. Dazu wird durch "hdv_init" und "hdv_boot" gesprungen.

Anschließend folgt der Bootversuch von Festplatte, der je nach Hardware (ACSI und/oder SCSI) und TOS-Version recht verschieden aussehen kann (siehe unter "Treiber für Festplatten, Bootvorgang").

Was jetzt kommt, ist nicht offiziell dokumentiert und sollte, wenn überhaupt, nur vorsichtig in kleinen Utilities benutzt werden: das BIOS durchsucht den gesamten Speicher nach einem 512-Byte-Block (also $400, $600 etc.), der folgende Eigenschaften erfüllt:

- Erster LONG-Wert ist die Konstante $12123456.
- Im zweiten LONG steht ein Zeiger auf die betreffende Speicherseite.
- Die 16-Bit-Prüfsumme aller Worte in diesem 512-Byte-Block ist $5678.

Wenn alle diese Bedingungen erfüllt sind, dann führt das BIOS einen "jsr" zu Byte 8 der Speicherseite durch. Die Suche beginnt unmittelbar unter "phystop". Zu diesem Zeitpunkt ist allerdings durch die GEMDOS-Initialisierung bereits aller Speicher oberhalb von $800 gelöscht, so daß man ohne Tricks nur den Speicherbereich zwischen $600 und $7FF benutzen könnte.

Nun werden die Programme im AUTO-Ordner gestartet. Dabei ist das Wurzelverzeichnis von "_bootdev" das aktuelle Verzeichnis. Die einzelnen Programmdateien werden also mit "Fsfirst ("\AUTO*.PRG", ...)" gesucht.

Zum Schluß wird die Default-Shell gestartet: wenn "_cmdload" nicht 0 ist, wird versucht, "COMMAND.PRG" auszuführen. Anderenfalls wird mehr schlecht als recht ein Default-Environment zusammengezimmert und über "exec_os" GEM gestartet.

Falls dieser Schritt schiefgeht oder die Shell terminiert wird (normalerweise bei GEM nicht möglich), geht es zurück zum Reset.

Der Reset-Vektor

Was muß man tun, um eigene Funktionen in den Reset-Vektor einklinken zu können?

1. "resvalid" muß auf die Konstante $31415926 gesetzt werden, ansonsten beachtet das BIOS "resvector" gar nicht erst.
2. Der Rücksprung aus der Funktion erfolgt mit "jmp (A6)", denn zu diesem Zeitpunkt der Systeminialisierung ist noch kein Stack installiert (und damit "jsr" und "rts" nicht möglich).
3. Damit sich mehrere Programme installieren können, muß man sich nach Abarbeitung der Funktion sauber de-installieren.

Hier ein Beispiel dazu:

```
; Installation einer Funktion im Reset-Vektor
; nach: "Rainbow TOS Release Notes"

; Assembler: MAS68

RESMAGIC        equ  $31415926
_resvalid       equ  $426
_resvector      equ  $42a
_p_cookies      equ  $5a0

     .text

; eigene Funktion installieren

install:
     move.l     _resvalid,oldvalid
     move.l     #RESMAGIC,_resvalid
     move.l     _resvector,oldreset
     move.l     #newreset,_resvector
     rts

; dies ist die neue Funktion

     dc.b       "XBRACKJR"      ; XBRA-Struktur
oldreset:
     dc.l       0

newreset:

; dieser Code wird während des Resets ausgeführt

     move.l     oldreset,_resvector
     move.l     oldvalid,_resvalid
     jmp        (a6)

     .bss

oldvalid:
     .ds.l      1
```

Der Vertical Blank Handler

Einleitung

Der Vertical-Blank-Interrupt (VBI) wird von der Video-Hardware erzeugt und liegt auf Interruptebene 4 (Vektor bei Adresse $70).

In der Interrupt-Hierarchie nimmt er damit einen Platz über dem Horizontal-Blank (der beim ST normalerweise nicht benutzt wird) und unterhalb der MFP-Interrupts ein.

Der VBI tritt jeweils beim vertikalen Zeilenrücklauf des Elektronenstrahls auf und ist damit besonders für die Manipulation von Registern in der Video-Hardware geeignet (da man sonst mit Störungen auf dem Bildschirm rechnen müßte).

Daher tritt er – je nach verwendetem Video-System – zwischen 50 und 72 mal pro Sekunde auf.

Die folgende Beschreibung der vom BIOS installierten Vertical-Blank-Routine erhebt weder Anspruch auf Korrektheit noch auf Vollständigkeit. Speziell in neuen BIOS-Versionen können sich Unterschiede bei der Reihenfolge ergeben!

1. Zunächst wird der Zähler "_frclock" inkrementiert.

2. Falls "vblsem" kleiner oder gleich 0 ist, wird die Interruptroutine beendet.

3. Alle Register werden gerettet.

4. "_vbclock" wird inkrementiert.

5. Falls die aktuelle Auflösung "hoch" ist (bei ST und STE: Auflösung 2, beim TT: Auflösung 6) *und* nicht der entsprechende Bildschirm angeschlossen ist, wird die Auflösung auf "defshiftmd" gesetzt und durch den Vektor "swv_vec" gesprungen.

6. Die VT52-Cursor-Blink-Routine wird aufgerufen.

7. Wenn "colorptr" ungleich 0 ist, wird die dort eingetragene Farbpalette in die Farbregister der Video-Hardware übertragen und "colorptr" anschließend wieder gelöscht.

8. Wenn "screenpt" ungleich 0 ist, wird die dort eingetragene Adresse in "_v_bas_ad" und die dazugehörigen Hardwareregister übertragen und "screenpt" *nicht* wieder gelöscht.

9. Die "deferred" Vertical-Blank-Routinen werden nacheinander aufgerufen (mehr dazu im Anschluß).

10. Falls "prt_cnt" nicht 0 ist, wird durch "scr_dump" gesprungen (und dabei normalerweise eine Hardcopy ausgegeben).

11. Die geretteten Registerinhalte werden wiederhergestellt.

"Deferred" Vertical-Blank-Routinen

Mit den beiden Systemvariablen "nvbls" ($454) und "_vblqueue" ($456) hat man die Möglichkeit, eigene Funktionen in die System-VBI-Routine einzuklinken. Dazu durchsucht man die Tabelle, auf die "_vblqueue" zeigt, nach freien Einträgen (also nach Einträgen, die Nullzeiger enthalten).

Ist kein freier Platz da (Anzahl der Plätze steht in "nvbls"), reserviert man Platz für eine eigene Tabelle, kopiert die bestehende dorthin und fügt die eigene Routine hinten an. Anschließend muß man noch die beiden Systemvariablen aktualisieren.

Warnung: Dreisterweise überschreibt GEM bei der Initialisierung den Vektor für die erste Vertical-Blank-Routine durch den Zeiger für die Mausbewegungs-Routinen. Soll ein im AUTO-Ordner installierter Interrupt diesen Anschlag überleben, muß man bei der Suche nach einer freien Adresse die erste Adresse überspringen (der Klügere gibt nach...). Der Grund dafür liegt wahrscheinlich darin, daß vermieden werden sollte, daß bei der erneuten Initialisierung (nach Wechseln der Auflösung) eine zweite Vertical-Blank-Routine installiert wird.

Hier ein Beispiel:

```
     ; VBLank installieren

     move.w    nvbls,d0
     lsl.w     #2,d0          ; mal 4
     move.l    _vblqueue,a0
     moveq     #4,d1          ; einen Eintrag überspringen
search_vb_slot:
     tst.l     (a0,d1)        ; frei?
     beq       slot_found
     addq      #4,d1
     cmp.w     d0,d1          ; Ende erreicht?
     bne       search_vb_slot
```

```
    ; Fehler melden
    clr.w     d0
    rts

slot_found:
    move.l    #MyVB,0(a0,d1) ; VBlank-Handler eintragen

    ; und weiter...
```

Feste Adressen im System

Einleitung

Die untersten zwei Kilobytes des Arbeitsspeichers nehmen im Atari eine besondere Rolle ein: die Belegung der meisten Speicheradressen ist dokumentiert und darf unter Beachtung einiger Vorsichtsmaßnahmen auch verändert werden.

Im einzelnen handelt es sich um

- Vektoren für Systeminterrupts (ab Byte 8 aufwärts) und
- die eigentlichen Systemvariablen (ab $380 aufwärts).

Der Zugriff auf diesen Speicherbereich ist nur aus dem Supervisor-Modus heraus möglich.

Die Systemvariablen und -vektoren sind die absolut "unterste" Schicht des Betriebssystems. Anwendungsprogramme sollten schon in Hinsicht auf mögliche Multitasking-Erweiterungen des Betriebssystems (wie zum Beispiel "Multi-GEM" von Maxon) niemals auf diese Speicherbereiche zugreifen. Nur kleine Utilities und Programme, die unter die Rubrik "Systemsoftware" fallen, sollten sich hier zu schaffen machen.

Also:

- *Niemals* Systemvariablen verändern, die laut Atari nicht verändert werden dürfen.
- *Niemals* Systemvariablen benutzen, wenn statt dessen auch eine Betriebssystemfunktion in Frage käme (siehe zum Beispiel "Setexc()" und "Mfpint()").
- *Immer* sauber zwischen Anwendungsprogrammen und kleinen Utilities oder Systemsoftware trennen.

Liste der Systemvektoren

LONG $000 0 Reset: SSP

Man sollte es nicht glauben, aber diese und die folgenden Speicherzellen enthalten ROM – und zwar eine Spiegelung der ersten acht Bytes des ROMs.

Bei einem RESET (durch Tastendruck oder Einschalten des Rechners) wird der hier liegende Wert in den Supervisor-Stack-Pointer geladen (allerdings kein sinnvoller – der "richtige" Stack wird erst später vom BIOS installiert).

LONG $004 4 Reset: PC

Bei RESET wird der hier vorgefundene Wert in den Program-Counter geladen. Man findet hier also die Adresse, die bei einem RESET angesprungen wird.

LONG $008 8 Busfehler

Exception-Vektor 2: zwei Bömbchen.

Busfehler treten auf, wenn man versucht, auf Speicherbereiche zuzugreifen, auf die kein Zugriff erlaubt ist. Das können im Supervisor-Modus eigentlich nur nicht existierende Speicherbereiche sein. Im User-Modus kann es auch beim Zugriff auf Hardware-Register oder Bereiche unterhalb von $800 passieren.

Im Normalfall zeigt dieser Vektor auf die TOS-Routinen zur Anzeige von Bömbchen (in diesem Fall zwei an der Zahl).

LONG $00C 12 Adreßfehler

Exception-Vektor 3: drei Bömbchen.

68000 und 68010 können nur byteweise auf ungerade Adressen zugreifen. Diese Exception wird ausgelöst, wenn man dennoch einen Wort- oder Langwort-Zugriff versucht (auch Programmzähler und Stack-Pointer dürfen keine ungeraden Werte enthalten!).

LONG $010 16 Illegaler Befehl

Exception-Vektor 4: vier Bömbchen.

Es wurde versucht, einen illegalen Befehl auszuführen. Dieser Vektor wird von vielen Debuggern zur Verwendung für Break-Points geändert.

LONG $014 20 Division durch Null

Exception-Vektor 5: kein Bömbchen.

Bei einem DIV-Befehl wurde durch 0 geteilt.

Dieser Vektor zeigt im Normalfall auf einen "RTE"-Befehl (Return from exception); daher gibt es auch keine Bömbchen.

LONG $018 24 CHK-Befehl

Exception-Vektor 6

Es wurde eine Exception durch einen "CHK"-Befehl erzeugt.

LONG $01C 28 Befehl TRAPV

Exception-Vektor 7

Es wurde eine Exception durch einen "TRAPV"-Befehl erzeugt.

LONG $020 32 Privilegverletzung

Exception-Vektor 8

Es wurde versucht, einen Befehl auszuführen, der nur im Supervisor-Modus erlaubt ist.

Beim TT testet das BIOS, ob es sich bei dem betreffenden Befehl um einen "move sr,..." gehandelt hat: dieser Befehl ist im Gegensatz zum 68000er nur im Supervisor-Modus erlaubt. In einem solchen Fall setzt das BIOS statt dessen einen "move ccr,.." ein und versucht, das Programm weiterlaufen zu lassen.

LONG $024 36 Trace

Exception-Vektor 9

Ist das TRACE-Bit im Statusregister gesetzt, wird nach jeder Instruktion die hier angegebene Adresse angesprungen.

LONG $028 40 Line-A-Vektor

Exception-Vektor 10

Es wurde versucht, eine Instruktion auszuführen, die in den obersten vier Bits den Wert "$A" enthält. Zur Zeit wird dieser Vektor für die "Line-A-Routinen" benutzt (siehe unter VDI-Gerätetreiber).

LONG $02C 44 Line-F-Vektor

Exception-Vektor 11

Es wurde versucht, eine Instruktion auszuführen, die in den obersten Bits den Wert "$F" enthält.

Wird bis TOS 1.04 vom GEM benutzt und ist eigentlich für die Programmierung der Floating-Point-Unit gedacht.

LONG $030 48 Reserviert

– $05C 92

Exception-Vektoren 12 – 23

LONG $060 9 Spurious Interrupt

Exception-Vektor 24

Tritt auf, wenn ein Interrupt ausgelöst wurde, die Ursache dafür aber nicht feststellbar war.

LONG $064 100 Autovektor-Interrupt, Level 1

Unbenutzt.

LONG $068 104 Autovektor-Interrupt, Level 2

Zeigt auf den Handler für Horizontal-Blanks.

LONG $06C 108 Autovektor-Interrupt, Level 3

Unbenutzt.

LONG $070 112 Autovektor-Interrupt, Level 4

Zeigt auf den Handler für Vertical-Blanks.

LONG $074 116 Autovektor-Interrupt, Level 5

Unbenutzt.

LONG $078 120 Autovektor-Interrupt, Level 6

Zeigt auf Handler für MFP-Interrupts.

LONG $07C 124 Autovektor-Interrupt, Level 7

Unbenutzt.

LONG $080 128 TRAP #0

Exception-Vektor 32

Vektor für den Befehl "TRAP #0". Da vom Betriebssystem nicht benutzt, gibt es Bömbchen.

LONG $084 132 TRAP #1

Exception-Vektor 33

Vektor für den Befehl "TRAP #1". Zeigt auf den Dispatcher für die GEMDOS-Funktionen.

LONG $088 136 TRAP #2

Exception-Vektor 34

Vektor für den Befehl "TRAP #2". Wird von AES und VDI benutzt.

LONG $08C 140 TRAP #3

–$0B0 –176 TRAP #12

Exception-Vektoren 35 bis 44: Entsprechend viele Bömbchen, da sie zur Zeit vom Betriebssystem nicht benutzt werden.

LONG $0B4 180 TRAP #13

Exception-Vektor 45

Vektor für den Befehl "TRAP #13". Zeigt auf den Dispatcher für die BIOS-Funktionen.

LONG $0B8 184 TRAP #14

Exception-Vektor 46. Vektor für den Befehl "TRAP #14". Zeigt auf den Dispatcher für die XBIOS-Funktionen.

LONG $0BC 188 TRAP #15

Unbenutzt.

LONG $0C0 192 Reserviert
– $0FC – 252

Exception-Vektoren 48 – 63

LONG $100 256 BUSY Interrupt

ST-MFP-Interrupt 0. Wird durch parallele Schnittstelle ausgelöst. Normalerweise nicht benutzt.

LONG $104 260 DCD Interrupt

ST-MFP-Interrupt 1. Wird durch die serielle Schnittstelle ("Carrier detect") ausgelöst. Normalerweise unbenutzt.

LONG $108 264 CTS Interrupt

ST-MFP-Interrupt 2. Wird durch die serielle Schnittstelle ("Clear to send") ausgelöst. Normalerweise unbelegt.

LONG $10C 268 GPU Done

ST-MFP-Interrupt 3. Kann vom Blitter benutzt werden, um den Abschluß einer Operation anzuzeigen. Normalerweise nicht benutzt.

LONG $110 272 Baudratengenerator

ST-MFP-Interrupt 4. Normalerweise nicht benutzt.

LONG $114 276 200 Hz System Timer

ST-MFP-Interrupt 5. Zeigt auf den Systemtimer-Interrupt und darf auf keinen Fall verändert werden (wird für Timing-Schleifen im TOS benötigt!).

LONG $118 280 IKBD/MIDI

ST-MFP-Interrupt 6. Zeigt auf den Handler für IKBD- und MIDI-Interrupts.

LONG $11C 284 FDC/ACSI

ST-MFP-Interrupt 7. Normalerweise unbelegt.

LONG $120 288 Display Enable Signal

ST-MFP-Interrupt 8. Normalerweise gesperrt.

LONG $124 292 RS232 Sendefehler

ST-MFP-Interrupt 9. Wird bei Übertragungsfehlern beim Senden von Daten über die serielle Schnittstelle ausgelöst.

LONG $128 296 RS232 Sendepuffer leer

ST-MFP-Interrupt 10. Wird ausgelöst, wenn der Sendevorgang eines einzelnen Bytes abgeschlossen worden ist.

LONG $12C 300 RS232 Empfangsfehler

ST-MFP-Interrupt 11. Tritt bei Empfangsfehlern auf.

LONG $130 304 RS232 Empfangspuffer voll

ST-MFP-Interrupt 12. Ein komplettes Zeichen ist von der seriellen Schnittstelle empfangen worden.

LONG $134 308 –

ST-MFP-Interrupt 13. Unbenutzt.

LONG $138 312 Ring Indicator

ST-MFP-Interrupt 14. Wird ausgelöst, wenn die serielle Schnittstelle einen ankommenden Anruf bemerkt (z. B. bei Verwendung von Modems). Da dieser Interrupt so leicht auszulösen ist (entweder mit einem speziellen Schalter an der RS232-Schnittstelle oder oft – bei angeschlossenem Modem – durch Aus- und Einschalten des Modems), wird er gerne in Debuggern als Break-Signal benutzt.

LONG $13C 316 Monochrom Monitor Detect

ST-MFP-Interrupt 15. Unbenutzt.

LONG $140 320 TT-MFP-Interrupts

– $17C 380

16 Interruptvektoren für den zweiten MFP im TT.

LONG $180 384 TT-SCC-Interrupts

– $1BC 444

Platz für die Interruptvektoren des TT-SCC-Bausteins.

Post-Mortem-Informationen

Bei einem Systemabsturz versucht das BIOS, noch Informationen über die Art des Problems auszugeben und das System wieder halbwegs in Ordnung zu bringen.

Zu den Maßnahmen gehören:

- Je nach Exceptionnummer wird in der Bildschirmmitte eine Reihe von Bombensymbolen ausgegeben. Dies natürlich per direktem Bildschirmzugriff, VDI-Funktionen können in diesem Moment natürlich *nicht* aufgerufen werden.

- Im Speicherbereich von $380 bis $3FF werden die aktuelle Registerbelegung und Teile des Stacks gerettet.

- Der vom BIOS-Dispatcher benötigte "savptr" wird zurückgesetzt.

- Das Programm wird mit "Pterm (–1)" beendet.

Und noch ein paar Anmerkungen:

- Jede dieser Aktionen kann natürlich in Unglücksfällen zu weiteren Exceptions führen. Resultat sind dann die verhaßten "Bombengirlanden".

- Der BIOS-Dispatcher ignoriert Exception Nummer fünf (Division durch 0). So bleiben unter Umständen einige Programmfehler unbemerkt.

- Der 68030 ist nicht hundertprozentig zum 68000 kompatibel. So gibt es beispielsweise einen Befehl, der auf dem 68000 im User-Modus erlaubt ist, auf dem 68030 allerdings mit einer "privilege violation" quittiert wird. Das TT-BIOS kompensiert dies durch einen speziellen Exception-Handler, der versucht, den Befehl sinnvoll zu emulieren.

Und hier die vom Exception-Handler benutzten Systemvariablen:

LONG $380 896 proc_lives

Wenn in dieser Adresse die magische Zahl ("magic number") $12345678 steht, sind die in den folgenden Adressen vorliegenden Informationen gültig.

LONG $384 900 proc_dregs

– $3a0 928

In diesen Adressen werden im Falle einer Exception, die Bömbchen hervorruft, folgende Register gerettet:

D0, D1, D2, D3, D4, D5, D6, D7

LONG $3A4 932 proc_aregs

– $3C0 960

In diesen Adressen werden im Falle einer Exception, die Bömbchen hervorruft, folgende Register gerettet:

A0, A1, A2, A3, A4, A5, A6, A7' (Supervisor-Stack-Pointer)

BYTE $3C4 964 proc_enum

Nummer der aufgetretenen Exception.

LONG $3C8 968 proc_usp

Geretter Wert des User-Stack-Pointers.

WORD $3CC 972 proc_stk

Die obersten sechzehn Worte des Stacks beim Auftreten der Exception.

Liste der Systemvariablen

LONG $400 1024 etv_timer

Logischer GEMDOS-Vektor 256 (siehe Einleitung zum GEMDOS). Sollte immer nur mittels "Setexc()" gesetzt werden.

LONG $404 1028 etv_critic

Logischer GEMDOS-Vektor 257 (siehe Einleitung zum GEMDOS). Sollte immer nur mittels "Setexc()" gesetzt werden.

LONG $408 1032 etv_term

Logischer GEMDOS-Vektor 258 (siehe Einleitung zum GEMDOS). Sollte immer nur mittels "Setexc()" gesetzt werden.

LONG $40C 1036 etv_xtra

Reserviert für die logischen GEMDOS-Vektoren 259 bis 263 (zur Zeit nicht benutzt).

LONG $420 1056 memvalid

Sollte die magische Zahl $752019F3 enthalten.

Siehe auch "memval2", "memval3" und "memcntrl".

BYTE $424 1060 memcntrl

Enthält die untersten vier Bits des Speicherkontroll-Registers ($FFFF8001).

LONG $426 1062 resvalid

Wenn diese Adresse bei RESET den Wert $31415926 enthält, wird durch "resvector" gesprungen.

LONG $42A 1066 resvector

Wird bei der Systeminitialisierung benutzt. Enthält den Vektor für RESET, falls "resvalid" den korrekten Wert beinhaltet. Zum Zeitpunkt des Aufrufs sind die Hardwareregister noch nicht gesetzt, und auch der Stack-Pointer ist noch nicht initialisiert.

LONG $42E 1070 phystop

Zeiger auf das erste Byte über dem physikalischen Ende des ST-kompatiblen RAM-Bereichs (zum Beispiel $00010000 bei 1MB RAM).

LONG $432 1074 _membot

Unteres Ende des unter GEMDOS freien ST-kompatiblen Speichers (also Anfang der "ursprünglichen" TPA). Wird von der BIOS-Funktion "Getmpb()" verwendet.

LONG $436 1078 _memtop

Analog zu "_membot" das Ende des freien ST-kompatiblen Speicherbereichs.

LONG $43A 1082 memval2

Sollte die Magic Number $237698AA enthalten. Haben sowohl "memval2" als auch "memvalid" (und ab TOS 1.02 "memval3") den geforderten Wert, wird beim nächsten RESET nur ein Warmstart durchgeführt.

WORD $43E 1086 flock

Wenn hier ein Wert ungleich 0 steht, dann darf *nicht* auf den DMA-Chip zugegriffen werden. DMA-Gerätetreiber *müssen* also zunächst abfragen, ob der DMA-Chip blockiert worden ist, und "flock" dann, wenn sie mit der Arbeit beginnen, selbst setzen.

WORD $440 1088 seekrate

Seekrate für die beiden Floppies. Erlaubte Werte:

Wert	Seekrate
0	6ms
1	12ms
2	2ms
3	3ms

Diese Systemvariable wird gleich nach dem Systemstart vom BIOS (durch den Aufruf der in "hdv_init" eingetragenen Routine) ausgelesen und danach ignoriert.

Zum Ändern der tatsächlich benutzten Seekrate muß man die XBIOS-Funktion "Floprate()" verwenden.

WORD $442 1090 _timr_ms

Zeit in Millisekunden, die normalerweise zwischen zwei Aufrufen des System-Timers vergeht (liegt daher bei 20; wg. 50 Hz). Dieser Wert wird auch von der BIOS-Funktion "Tickcal()" zurückgeliefert.

WORD $444 1092 _fverify

Legt fest, ob das BIOS beim Schreiben auf Diskette per "Rwabs()" einen Verify durchgeführen soll (0 bedeutet: kein Verify). Im Normalfall ist Verify eingeschaltet.

WORD $446 1094 _bootdev

Enthält normalerweise die Nummer des Laufwerks, von dem gebootet worden ist.

Das BIOS benutzt diese Variable, um den Standardzugriffspfad für die Environmentvariable "PATH" zu ermitteln. In allen bekannten TOS-Versionen greift es dazu allerdings auf das höherwertige Byte zu, so daß zumindest an dieser Stelle immer "A:\" herauskommt. Harddisktreiber von Fremdherstellern beheben dieses Problem meist.

Mit dem Atari-Festplattentreiber fällt es nicht weiter auf, da die AES-Accessories immer auf Laufwerk C: suchen, falls dieses existiert (und von anderen Partitionen kann man mit AHDI sowieso nicht booten).

Daneben wird diese Variable auch noch beim Booten benutzt, um das Bootlaufwerk zu wählen (das klappt im allgemeinen aber nur dann, wenn kein Festplattentreiber gebootet wird).

Wenn man also "_bootdev" auf 1 setzt, keine autobootende Festplatte angeschlossen hat und einen Reset auslöst, wird von Laufwerk B: gebootet.

WORD $448 1096 palmode

Legt laut Atari die Fernsehnorm fest.

Die Werte:

0: NTSC-Modus (60 Hz)

sonst: PAL-Modus (50 Hz)

Tatsächlich wird diese Systemvariable nicht berücksichtigt. Eine Änderung der Bildwiederholfrequenz ist nur über die entsprechenden Hardwareregister möglich.

BYTE $44A 1098 defshiftmd

Standard-Farbgrafik-Auflösung. Schaltet der Computer auf Farbbetrieb um (nach RESET durch Wechseln der Stecker oder Einschalten), wird in die angegebene Auflösung geschaltet.

BYTE $44C 1100 sshiftmd

Kopie des Modus-Registers des Shifters ($FFFF8260). Die Werte:

Wert	Bildgröße
0:	320 * 200 (vier Planes)
1:	640 * 200 (zwei Planes)
2:	640 * 400 (ein Plane)
4:	640 * 480 (vier Planes, nur beim TT)
6:	1280 * 960 (ein Plane, nur beim TT)
7:	320 * 480 (acht Planes, nur beim TT)

Alle anderen Werte sind für künftige Erweiterungen reserviert.

LONG $44E 1102 _v_bas_ad

Zeiger auf den Anfang des Bildspeichers, der beim ST auf einer 256-Byte-Grenze beginnen muß (Sorry, kein Raum für großartige Tricks für Fine-Scrolling). Beim STE und TT ist es eine 2- bzw. 8-Byte-Grenze.

Dies ist der Wert, der normalerweise von "Logbase()" zurückgeliefert wird.

WORD $452 1106 vblsem

1: Vertical-Blank-Handler aktiviert.

WORD $454 1108 nvbls

Anzahl der Einträge, auf die "_vblqueue" zeigt. Identisch mit der Maximalzahl von gleichzeitig installierbaren Vertical-Blank-Routinen. Standardwert ist 8.

LONG $456 1110 _vblqueue

Zeiger auf Zeigertabelle für Vertical-Blank-Prozesse.

LONG $45A 1114 colorptr

Zeiger auf eine Farbpalette, die beim nächsten Vertical Blank in die ST-Hardware-Farbregister geladen wird (ab Adresse $FFFF8240). Damit wird ein unschönes Zucken auf dem Bildschirm vermieden. Steht in "colorptr" eine Null, passiert gar nichts. Nach der Übertragung der Farbwerte wird "colorptr" gelöscht.

Besser ist es, die entsprechenden VDI-Funktionen zu nutzen, da schon beim "Atari TT" die Farbpalette mehr als 16 Einträge umfassen kann. Bei anderen Grafikkarten wird diese Variable ganz ignoriert.

LONG $45E 1118 screenpt

Zeiger auf den Anfang des Bildspeichers. Wird beim nächsten Vertical Blank in die betreffenden Hardwareregister übertragen, anschließend aber nicht gelöscht (daher sollte man statt dessen immer mit "Setscreen()" arbeiten!).

LONG $462 1122 _vbclock

Anzahl der bereits erfolgten Vertical Blanks.

LONG $466 1126 _frclock

Wie "_vbclock", mit dem Unterschied, daß die Zählung nicht durch "vblsem" angehalten wird.

LONG $46A 1130 hdv_init

Vektor zu den Initialisierungsroutinen für die Diskettenlaufwerke. Wird vor dem Lesen der Bootsektoren ausgelesen und kann daher nur von reset-residenten Programmen oder ROM-Modulen verändert werden.

Zu den Aufgaben gehören:

- Initialisierung der Diskettenlaufwerke ("_nflops" wird entsprechend gesetzt)
- Übertragung von "seekrate" in die BIOS-internen Variablen

LONG $46E 1134 swv_vec

Zeiger auf die Routine, die auf das Anschließen eines Schwarzweiß- bzw. Farbmonitors reagiert (zeigt zu Beginn auf die normale RESET-Routine).

LONG $472 1138 hdv_bpb

Vektor zur Routine, die den BPB (BIOS Parameter Block) eines BIOS-Laufwerks ermittelt. Auf dem Stack (4(sp)) wird die BIOS-Gerätenummer übergeben.

In D0 liefert man entweder einen Zeiger auf den BPB oder im Fehlerfall 0 zurück.

LONG $476 1142 hdv_rw

Vektor zur Routine zum Lesen und Schreiben von Blöcken auf BIOS-Laufwerken. Auf dem Stack werden die gleichen Parameter wie bei "Rwabs()" übergeben (beginnend mit 4(sp): rwflag).

In D0 liefert man den Return-Wert für "Rwabs()" zurück (beispielweise 0, wenn alles geklappt hat).

LONG $47A 1146 hdv_boot

Vektor zur Routine zum Laden des Bootsektors. Diese Routine wird vom BIOS benutzt, um festzustellen, ob ein Bootsektor vorhanden und ob er ausführbar ist.

LONG $47E 1150 hdv_mediach

Vektor zur Routine zur Bestimmung des Medienwechsel-Status eines BIOS-Laufwerks. Auf dem Stack (4(sp)) wird die BIOS-Gerätenummer übergeben.

In D0 liefert man den Return-Wert für "Mediach()" zurück.

WORD $482 1154 _cmdload

Wenn dieses Register nicht 0 ist, wird versucht, anstelle von GEM das Programm "COMMAND.PRG" zu starten (kann durch ein Programm in einem ausführbaren Bootsektor gesetzt werden).

BYTE $484 1156 conterm

Attributbits für BIOS-Gerät "CON:":

Bit	Bedeutung
0	Tastenklick (Keyclick) ein/aus
1	Tastenwiederholung ein/aus

Bit	Bedeutung
2	Glocke (Ping!) bei Ausgabe von CTRL-G
3	Bei "Bconin()" aktuellen Wert von "Kbshift()" in den Bits 24..31 zurückgegeben

LONG $486 1158 trp14ret

Offiziell nicht dokumentiert und wohl auch unbenutzt.

LONG $48A 1162 criticret

Offiziell nicht dokumentiert und wohl auch unbenutzt.

MD $48E 1166 themd

MD-Struktur des GEMDOS.

Diese wird ein einziges Mal bei der Initialisierung des Systems gesetzt und darf nicht verändert werden (und das wird sie durch Benutzung der BIOS-Funktion "Getmpb()"!).

LONG $49E 1182 ____md

Offiziell nicht dokumentiert und wohl auch unbenutzt.

LONG $4A2 186 savptr

Zeiger auf Register-Zwischenspeicher von BIOS und XBIOS. Mehr dazu bei der Dokumentation des BIOS- bzw. XBIOS-Bindings.

WORD $4A6 1190 _nflops

Anzahl der angeschlossenen Diskettenlaufwerke (0, 1 oder 2).

LONG $4A8 1192 con_state

Interner Zeiger für Bildschirmausgaberoutinen; offiziell nicht dokumentiert.

WORD $4AC 1196 sav_row

Interner Puffer zur Zwischenspeicherung der Cursor-Position; offiziell nicht dokumentiert.

LONG $4AE 1198 sav_context

Sollte eigentlich ein Zeiger auf den Speicherbereich sein, in den bei Exceptions die Register und Teile des Stacks gerettet werden. Tatsache aber ist, daß er vom TOS nicht benutzt wird und daß man daher direkt auf die Variablen bei Adresse $380 zugreifen muß.

LONG $4B2 1202 _bufl

Zwei Zeiger auf GEMDOS-Pufferlisten. Mehr dazu in der Einführung zum GEMDOS.

LONG $4BA 1210 _hz_200

Bisherige Anzahl der 200-Hz-Interrupts.

LONG $4BE 1214 the_env

Zeiger auf die Standard-Environment-Strings (unbenutzt!).

LONG $4C2 1218 _drvbits

Bit-Tabelle über die angemeldeten BIOS-Laufwerke (Bit Null für Laufwerk "A:" etc.). Hier ist also Platz für 32 Einträge! Diese Variable wird im TOS 1.00 beim Reset nicht gelöscht. Eigene Treiber sollten daher bei einem Reset die selbst eingetragenen Bits löschen. Ansonsten kann es passieren, daß Laufwerkskennungen "verschwinden" (viele Treiber installieren sich immer auf der ersten "freien" Gerätenummer).

LONG $4C6 1222 _dskbufp

Zeiger auf einen 1024 Bytes großen Puffer zum Lesen und Schreiben auf Disketten oder Festplatten (z. B. beim Bootversuch). Wird auch vom VDI verwendet.

LONG $4CA 1228 _autopath

Zeiger auf Zugriffspfad für AUTO-Ordner (unbenutzt und nicht offiziell dokumentiert).

LONG $4CE 1232 _vbl_list

Ursprüngliche Liste der Vertical-Blank-Routinen (immer nur über "_vblqueue" zugreifen!).

WORD $4EE 1262 prt_cnt

Zähler für die ALT-HELP-Tastendrücke:

Wert	Bedeutung
−1:	normaler Status
0:	Hardcopy beginnen
>0:	Hardcopy abbrechen und auf −1 zurücksetzen

Diese Variable muß man auch vor einem Aufruf von "Prtblk()" setzen!

WORD $4F0 1264 _prtabt

Flag für Abbruch des Druckvorgangs (unbenutzt).

LONG $4F2 1266 _sysbase

Zeiger auf eine Struktur folgender Form:

```
typedef struct _osheader
{
     UWORD     os_entry;      /* BRAnch-Instruktion zum RESET-
                                 Handler, Offset $00 */
     UWORD     os_version;    /* TOS-Versionsnummer, Offset $02 */
     void      *reseth;       /* Zeiger auf RESET-Handler,
                                 Offset $04 */
     struct _osheader
               *os_beg;       /* Basisadresse des Betriebssystems,
                                 Offset $08 */
     void      *os_end;       /* Erstes nicht vom OS benutztes
                                 Byte, Offset $0C */
     LONG      os_rsv1;       /* reserviert, Offset $10 */
     GEM_MUPB  *os_magic;     /* Zeiger auf "GEM memory usage
                                 parameter block", Offset $14 */
     LONG      os_date;       /* TOS-Herstellungsdatum im BCD-
                                 Format, etwa $02061986 für
                                 6. Februar 1986, Offset $18 */
     UWORD     os_conf;       /* verschiedene Konfigurationsbits,
                                 Offset $1C */
     UWORD     os_dosdate;    /* TOS-Herstellungsdatum im GEMDOS-
                                 Format, Offset $1E */

     /* die folgenden Felder erst ab TOS-Version 1.02 */
```

```
    char      **p_root;      /* Basisadresse des GEMDOS-Pools,
                                Offset $20 */
    BYTE      **pkbshift;    /* Zeiger auf BIOS-interne Variable
                                für den aktuellen Wert von
                                "Kbshift()", Offset $24 */
    BASEPAGE  **p_run;       /* Adresse der Variablen, die einen
                                Zeiger auf den aktuellen GEMDOS-
                                Prozeß enthält,Offset $28 */
    char      *p_rsv2;       /* reserviert, Offset $2C */
} OSHEADER;
```

Zum "OS-Header" gibt es viel zu sagen und deshalb auch einen eigenen Abschnitt (direkt vor diesem!).

LONG $4F6 1270 _shell_p

Dieser Zeiger wird vom ROM nicht genutzt. Das heißt: Programme, die ihn selbst benutzen, müssen ihn im Falle eines Resets und natürlich auch bei Programmbeendigung löschen!

Normalerweise wird "_shell_p" von UNIX-ähnlichen Shells gesetzt und zeigt auf eine Routine, die eine Kommandozeile abarbeitet. Die Adresse der Zeichenkette wird auf dem Stack (4(sp)) übergeben, das Ergebnis der Operation erhält man in Register D0. Die meisten C-Libraries enthalten ein Binding für die Standardfunktion "system()", das in etwa so aussieht:

```
/* WORD system (const char *cmd)
   Führt das Kommando "cmd" über die in _shell_p installierte
   Shell aus.

   Wenn "cmd" ein Nullzeiger ist, wird festgestellt, ob eine Shell
   installiert ist. Ansonsten wird die Zeile übergeben und der
   Return-Wert der Shell zurückgeliefert.
*/
#define _SHELL_P ((LONG *)0x4f6L)

WORD system (const char *cmd)
{
    WORD cdecl (*do_sys)(const char *cmd);
    LONG oldssp;
    oldssp = Super (0L);
    do_sys = (void (*))*_SHELL_P;
    Super ((void *)oldssp);
```

```
    if (!cmd)
        return (do_sys != 0L);

    if (do_sys != 0L)
        return do_sys (cmd);

    else
        return -1;
}
```

Was in der Kommandozeile stehen kann, ist natürlich von der benutzten Shell abhängig. Bei allen in Frage kommenden Shells handelt es sich allerdings um UNIX-ähnliche Shells, so daß man davon ausgehen kann, daß UNIX-übliche Standardkommandos und auch eigene, zusätzlich installierte TOS-Programme gestartet werden können.

LONG $4FA 1274 end_os

Zeiger auf das erste nicht für TOS-interne Variablen benutzte Byte (also das erstes Byte des freien Speichers).

LONG $4FE 1278 exec_os

Zeiger auf das erste Byte des Textsegments des Shell-Programms.

Das Shell-Programm wird nach der vollständigen Initialisierung von GEMDOS und dem Abarbeiten des AUTO-Ordners mittels "Pexec()" gestartet (normalerweise AES und Desktop).

LONG $502 1282 scr_dump

Zeiger auf Hardcopyroutine (wird von der XBIOS-Funktion "Scrdmp()" benutzt).

LONG $506 1286 prv_lsto

Zeiger auf Routine zum Feststellen des Status des parallelen Ports (ebenfalls für Hardcopy-Routine).

LONG $50A 1290 prv_lst

Zeiger auf Routine zur Ausgabe auf dem parallelen Port (ebenfalls für Hardcopy-Routine). Das auszugebende Zeichen steht in 6(sp).

LONG $50E 1294 prv_auxo

Zeiger auf Routine zum Feststellen des Status der seriellen Schnittstelle (ebenfalls für Hardcopy-Funktion).

LONG $512 1298 prv_aux

Zeiger auf Routine zur Ausgabe auf dem seriellen Port (ebenfalls für Hardcopy-Routine). Das auszugebende Zeichen steht in 6(sp).

LONG $516 1302 pun_ptr

Zeigt bei erfolgreicher Installation eines AHDI-kompatiblen Festplattentreibers auf die folgende Datenstruktur. Näheres dazu im Abschnitt "Treiber für Festplatten".

```
typedef struct
{
     WORD      puns;              /* Anzahl der Geräte */
     BYTE      pun[16];           /* diverse Flags */
     LONG      part_start[16];    /* Partitionsanfänge */
     LONG      P_cookie;          /* muß "AHDI" sein */
     LONG      *P_cookptr;        /* zeigt auf das vorherige
                                     Element */
     UWORD     P_version;         /* 0x0300 oder größer */
     UWORD     P_max_sector;      /* maximale Sektorgröße */
     LONG      reserved[16];
} PUN_INFO;
```

LONG $51A 1306 memval3 (ab TOS 1.02)

Siehe auch memval und memval2. In diesem Fall ist der "magic value" $5555AAAA.

LONG $51E 1310 xconstat (ab TOS 1.02)

Acht Vektoren für "Bconstat()"-Routinen (mehr dazu in der BIOS-Einführung unter "Zeichenorientierte Funktionen").

LONG $53E 1342 xconin (ab TOS 1.02)

Acht Vektoren für "Bconin()"-Routinen (mehr dazu in der BIOS-Einführung unter "Zeichenorientierte Funktionen").

LONG $55E 1374 xcostat (ab TOS 1.02)

Acht Vektoren für "Bcostat()"-Routinen (mehr dazu in der BIOS-Einführung unter "Zeichen-orientierte Funktionen"). *Vorsicht*: für "Bcostat()" sind die Kanalnummern 3 (MIDI) und 4 (IKBD) vertauscht.

LONG $57E 1406 xconout (ab TOS 1.02)

Acht Vektoren für "Bconout()"-Routinen (mehr dazu in der BIOS-Einführung unter "Zeichen-orientierte Funktionen").

WORD $59E 1438 _longframe

Wenn dieses Flag nicht 0 ist, dann ist eine CPU mit langen Stackframes (also kein 68000er) installiert.

Dieser Wert ist speziell dann von Interesse, wenn man eine Routine in einen Exception-Vektor einklinken will und die zu untersuchenden Werte auf dem Stack übergeben werden. Wenn "_longframe" 0 ist, findet man dann die Parameter bei Offset 6, ansonsten bei Offset 8.

Man kann sich übrigens durchaus darauf verlassen, daß der hier stehende Wert korrekt ist. Hersteller von "Beschleunigerkarten" müssen sich sowieso darum kümmern, daß diese Systemvariable korrekt gesetzt ist – sonst würde ein großer Teil der Atari-Systemsoftware (wie zum Beispiel der Diablo-Emulator der SLM-Laserdrucker) nicht funktionieren.

LONG $5A0 1440 _p_cookies

Zeiger auf den Cookie Jar (deutsche Übersetzung: "Keksdose"). Dokumentation im Anschluß zu diesem Abschnitt.

LONG $5A4 1444 ramtop

Zeiger auf das Ende des Fast-RAMs im TT. Nicht offiziell dokumentiert!

LONG $5A8 1448 ramvalid

Magic-Wert, der anzeigt, ob "ramtop" einen sinnvollen Wert enthält (muß $1357BD13 sein). Nicht offiziell dokumentiert!

LONG $5AC 1452 bell_hook (ab TOS 1.06)

Zeiger auf Routine zur Ausgabe des "Ping"-Geräusches.

BIOS sorgt selbsttätig für die Abfrage des Flags in "con_term" und ruft diese Routine nur dann auf, wenn das Geräusch wirklich erklingen soll. Die Routine wird im Supervisor-Modus aufgerufen, per "rts" abgeschlossen und darf die Register D0-D2 und A0-A2 verändern. Auch BIOS-Aufrufe vom "Innern" der Routine aus sind erlaubt.

LONG $5B0 1456 kcl_hook (ab TOS 1.06)

Zeiger auf Routine zur Ausgabe des Tastenklick-Geräuschs. BIOS sorgt selbsttätig für die Abfrage des Flags in "con_term" und ruft diese Routine nur dann auf, wenn das Geräusch wirklich erklingen soll. Die Routine wird im Supervisor-Modus aufgerufen, per "rts" abgeschlossen, darf die Register D0-D2 und A0-A2 verändern und sollte nicht allzu viel Zeit verbrauchen.

Der Cookie Jar

Einleitung

Beim "Cookie Jar" ("Keksdose") handelt es sich im Grunde genommen um eine Generalisierung der Systemvariablen. Die wichtigsten Unterschiede:

- Der Cookie Jar ist eine Tabelle, von der nur die Basisadresse bekannt ist. Damit kann sie bei Bedarf verlängert und im Speicher umhergeschoben werden.

- Jeder Eintrag im Cookie Jar hat eine (hoffentlich) eindeutige Kennung. Damit hat man die Möglichkeit, auch selbst Einträge für eigene Zwecke vorzunehmen.

"Cookie" ist ein im Atari-Slang gebräuchlicher Terminus und steht für meist 32 Bits große Codenummern. Meistens interpretiert man sie als vier ASCII-Zeichen, die eine Abkürzung über den jeweiligen Verwendungszweck enthalten. Zu jedem Cookie gehört ein 32 Bits großer Eintrag, der je nach Cookie verschiedene Bedeutungen haben kann. Meist handelt es sich um Zeiger auf weitere Strukturen oder um Versionsnummern. Der "Cookie Jar" ist nichts anderes als ein Array von Cookies und ihren Werten.

Einige Anwendungsbeispiele

"MACCEL3", der Mausbeschleuniger von Atari (gehört zum Lieferumfang von Mega STE und TT), trägt in den Cookie Jar einen Zeiger auf eine programminterne Struktur ein, mit deren Hilfe man im laufenden Betrieb seine Parameter verstellen kann (MACCEL3 benutzt aller-

dings einen Cookie, der nicht aus ASCII-Zeichen besteht – wohl deshalb, weil sein Vorgänger MACCEL2 vor der eigentlichen Definition des Cookie Jar entstanden ist). In diesem Fall wird der Cookie also von einem residenten Utility installiert, das über den Cookie Datenstrukturen zur Konfiguration nach "außen" führt. Diese Schnittstelle kann dann von einem Accessory oder einem Kontrollfeld-Modul genutzt werden.

Das BIOS trägt ab TOS 1.06 die sogenannten System-Cookies ein, mit Hilfe derer man detaillierte Informationen über die installierte Hardware erfragen kann.

Aufbau des Cookie Jar

Der Zeiger bei Adresse $5A0 ("_p_cookies") ist entweder 0 (dann ist noch kein Cookie Jar installiert), oder er zeigt auf eine Tabelle von Paaren von 32-Bit-Werten.

Im ersten steht jeweils die Identifikation in Form von vier ASCII-Zeichen. Bei der Definition eigener Kennungen sollte man folgendes beachten:

- Mit "_" beginnende Kennungen sind für Atari reserviert.
- Die vier Buchstaben sollten im engsten Sinne "druckbar" sein (ASCII-Codes zwischen 32 und 126, keine nationalen Sonderzeichen!).
- Die vier Buchstaben sollten eine Abkürzung ergeben, aus der man auf das zugehörige Programm schließen kann. Varianten der Wörter "Cookie", "Vector" u. ä. gehören nicht dazu!

Das Ende der Liste wird durch einen "Nullcookie" angezeigt (also $00000000), der als Wert die maximale Anzahl von Einträgen im Cookie Jar enthält.

Abfrage von Cookies

Es ist leicht, Werte existierender Cookies abzufragen – eine Beispielfunktion in ANSI-C:

```
/* WORD GetCookie (LONG cookie, LONG *value)
  "cookie" im Cookie Jar suchen.

  Bei Erfolg wird der Wert in "value" abgelegt und als
  Funktionswert TRUE zurückgeliefert. Anderenfalls erhält
  man FALSE. */
```

```
WORD GetCookie (LONG cookie, LONG *value)
{
     LONG oldstack;
     LONG *cookiejar;

     /* Zeiger auf Cookiejar holen */

     oldstack = Super (0L);
     cookiejar = *((LONG **)0x5a0L);
     Super ((void *)oldstack);

     /* Cookiejar überhaupt vorhanden? */
     if (cookiejar == 0L)
          return FALSE;
     do
     {
          /* Cookie gefunden? */
          if (cookiejar[0] == cookie)
          {
               /* nur eintragen, wenn "value" kein Nullpointer */
              if (value)
                   *value = cookiejar[1];

               return TRUE;
          }
          else
          {
               /* nächsten Cookie nehmen */

               cookiejar = &(cookiejar[2]);
          }
     } while (cookiejar[-2]); /* Nullcookie? */
     return FALSE;
}
```

Installation eigener Cookies

Wie trägt man nun "seinen" Cookie in die Liste ein? Dazu stellt man zunächst fest, wie lang die Liste eigentlich ist (also vom Beginn der Liste an nach dem "Nullcookie" suchen, dessen Wert ja die Länge der Liste anzeigt). Da eben dieser Nullcookie auch das Ende der Liste

anzeigt, muß man nur dort seinen eigenen Cookie hineinkopieren und den Nullcookie um einen Eintrag zum Ende hin zu verschieben.

War die Liste allerdings schon voll, dann bleibt nichts anderes übrig, als Platz für eine neue (längere) Liste zu allozieren, die alten Einträge dorthin zu kopieren und "_p_cookies" umzubiegen. Wenn man schon dabei ist, sollte man vielleicht gleich mehr Platz als für einen zusätzlichen Cookie schaffen.

Dies geht logischerweise nur in Programmen, die sich nach ihrem Aufruf mittels "Ptermres()" resident im Speicher verankern. Andere Programme oder Acessories können und dürfen keine Cookies installieren! Statt dessen sollte das AES-Message-Passing oder die Environment-Variablen benutzt werden.

Bleibt noch die Frage: wie kann man einen Cookie wieder entfernen? Leider reicht es nicht, einfach den bestehenden Eintrag zu "löschen" (denn was soll schon ein "leerer" Eintrag sein?). Also kommt man nicht umhin, alle folgenden Cookies innerhalb des Cookie Jar um einen Eintrag nach vorne zu kopieren.

Das ist zwar nicht sonderlich bequem, aber so oft wird man Cookies ja nicht entfernen...

Der Cookie Jar vor TOS 1.06

Vor TOS 1.06 wurde der Cookie Jar noch nicht automatisch installiert. Dann hat "_p_cookies" den Wert 0.

Programme können allerdings problemlos einen leeren Cookie Jar anlegen und "_p_cookies" entsprechend setzen. Das System (und alle anschließend geladenen Programme) verhalten sich dann genauso, als wäre der Cookie Jar schon immer dagewesen.

Nur eine Einschränkung ist zu berücksichtigen: bei einem Reset wird "_p_cookies" nicht automatisch gelöscht. Ein Programm, das einen neuen Cookie Jar anlegt, muß sich also auch darum kümmern, daß "_p_cookies" bei einem Reset wieder gelöscht wird!

Dazu ein Beispielprogramm, das einen leeren Cookie Jar anlegt:

```
; cookiejr.s ;
; Assembler: MAS68
; Installiert einen 40 Einträge langen
; Cookie Jar, sofern noch nicht vorhanden
; beseitigt ihn ggfs. bei einem Reset
```

```
RESMAGIC        equ  $31415926
_resvalid       equ  $426
_resvector      equ  $42a
_p_cookies      equ  $5a0

     .text

     pea        Install
     move.w     #38,-(sp)       ; Supexec
     trap       #14             ; XBIOS
     addq.l     #6,sp

     tst.w      d0              ; installiert?
     beq        NotInstalled

     ; Installationsmeldung ausgeben

     pea        Success
     move.w     #9,-(sp)        ; Cconws
     trap       #1              ; GEMDOS

     ; und resident halten

     move.l     4(sp),a0        ; Basepage
     move.l     #$100,d0        ; 256 Bytes Basepage
     add.l      $C(a0),d0       ; p_lten
     add.l      $14(a0),d0      ; p_dlen
     add.l      $1C(a0),d0      ; p_blen
     clr.w      -(sp)
     move.l     d0,-(sp)
     move.w     #49,-(sp)       ; Ptermres
     trap       #1              ; GEMDOS
NotInstalled:
     pea        Failure
     move.w     #9,-(sp)        ; Cconws
     trap       #1              ; GEMDOS
     clr.w      -(sp)           ; richtig terminieren
     trap       #1              ; GEMDOS Pterm

Install:
     clr.w      d0
```

```
     tst.l     _p_cookies     ; schon installiert?
     beq       InstallIt
     rts                      ; Schluß!

InstallIt:
     moveq     #1,d0
     move.l    #NewCookies,_p_cookies
     move.l    _resvalid,OldValid
     move.l    #RESMAGIC,_resvalid
     move.l    _resvector,OldReset
     move.l    #NewReset,_resvector
     rts

     dc.b      "XBRACKJR"     ; XBRA-Struktur
OldReset:
     dc.l      0
NewReset:
     clr.l     _p_cookies     ; wieder löschen
     move.l    OldReset,_resvector
     move.l    OldValid,_resvalid
     jmp       (a6)

     .data

Success:
     .dc.b     "cookiejr: Cookiejar for 40 cookies
                 installed.",13,10,0
Failure:
     .dc.b     "cookiejr: Cookiejar already in use.",13,10,0
NewCookies:
     .dc.l     0,40   ; 40 Stück
     .dcb.l    78,0   ; 78 mal 0

     .bss

OldValid:
     .ds.l     1


     .end
```

Vom BIOS belegte Cookies

Ab TOS 1.06 wird der Cookie Jar automatisch bei der Systeminitialisierung installiert und mit einigen Informationen über den benutzen Rechner versehen.

Zur Zeit sind folgende Cookies dokumentiert:

_CPU – Prozessortyp

Hat die Werte 0, 10, 20, 30 oder 40 für Rechner mit 68000, 68010, 68020, 68030 oder 68040.

_FPU – FPU-Typ

Dieser Cookie beschreibt, auf welche Art Hardware und Betriebssystem die Arbeit mit Fließkommazahlen unterstützen. Dabei sind verschiedene Schnittstellen denkbar:

- In ST- und STE-Modellen kann der 68881 als Peripheriebaustein installiert werden (beim ST und STE über eine Steckkarte wie die SFP 004 von Atari; beim Mega STE über einen Steckplatz auf der Hauptplatine).

- Bei Systemen mit 68020 oder 68030 kann eine eventuell vorhandene FPU über die Line-F-Instruktionen angesprochen werden.

- Beim 68040 sind die meisten Befehle des 68882 direkt in der CPU eingebaut. Einige wenige Ausnahmen werden per Software emuliert. Beim Zugriff über Line-F besteht vollständige Kompatibilität.

- Bei Rechnern mit TOS 1.06 oder neueren TOS-Versionen ist es denkbar, nachträglich einen Line-F-Treiber (für Softwareemulation oder für den Zugriff auf einen als Peripheriebaustein betriebenen 68881) zu betreiben.

Das obere Wort beschreibt den Typ des benutzten Floating-Point-Koprozessors:

Bit 0:	falls gesetzt: SFP 004 oder kompatible FPU-Karte (68881 als Peripheriebaustein)
Bit 1..2:	68881 oder 68882 als Koprozessor, im Detail: 0: weder – noch 1: 68881 oder 68882, Typ unbekannt 2: 68881 3: 68882
Bit 3:	falls gesetzt: 68040

68881 und 68882 sind von der Softwareseite her praktisch vollständig kompatibel. Daher macht sich das BIOS des TT auch keine Mühe, den genauen Typ zu erkennen. Software, die die genaue Typenbezeichnung braucht, kann diese Information selbst zu ermitteln versuchen und den Cookie entsprechend korrigieren.

Das untere Wort ist für Informationen über Softwareunterstützung via Line-F-Trap reserviert und ist zur Zeit noch nicht benutzt. Laut Atari bedeutet ein Wert ungleich 0, daß Line-F-Unterstützung vorhanden ist.

Für den Anwendungsprogrammierer sind eigentlich nur zwei Fragen interessant:

1. Hat der Rechner eine als Peripheriebaustein installierte FPU? Hierfür sollte man bei vorhandenem FPU-Cookie Bit 0 des oberen Werts testen. Ist der Cookie nicht vorhanden (zum Beispiel in Versionen vor TOS 1.06), muß man direkt die Hardwareadressen überprüfen (Löschen des LONG-Werts an Adresse $FFFFFA46 führt zum Bus-Error).

2. Können die Line-F-Instruktionen für Floating-Point-Berechnungen benutzt werden? Hierzu reicht es, den FPU-Cookie zu testen. Wenn das obere Wort einen Wert größer 1 oder das untere Wort einen Wert ungleich 0 hat, sind Line-F-Instruktionen erlaubt.

_FRB – Fast-RAM-Buffer

Da das "Fast RAM" des TT für normale ACSI-DMA-Transfers nicht benutzt werden kann, legt das BIOS auf solchen Rechnern einen 64 KByte großen Puffer im ST-RAM an, dessen Adresse man hier vorfindet. Gerätetreiber für die ACSI-Schnittstelle dürfen diesen Puffer als temporären Zwischenspeicher für Transfers in das Fast-RAM nutzen. Der Zugriff wird über die Systemvariable "flock" koordiniert.

Wenn dieser Cookie nicht da ist, verfügt die Maschine entweder über kein Fast-RAM oder keine ACSI-Schnittstelle.

_MCH – Maschinentyp

Beschreibt den benutzten Rechnertyp. Das obere Wort bezeichnet die Rechnerfamilie:

0: ST (520ST, 1040ST und Mega ST und ähnliche)
1: STE (1040 STE, Mega STE)
2: TT

Das untere Wort dient für feinere Unterscheidungen und ist zur Zeit nur beim Mega STE (0x0010) ungleich 0.

_SND – Soundhardware

Bittabelle, die die vorhandenen Soundmöglichkeiten beschreibt.

Bit 0: GI/Yamaha Sound Chip wie in bisher allen bekannten Rechnern
Bit 1: Stereo-DMA-Sound (wie beim STE und dem TT)

_SWI – DIP-Switches

Werte der "Konfigurationsschalter" (Dip-Schalter), falls vorhanden.

_VDO – Videohardware

Beschreibt die verfügbare Videohardware. Das obere Wort wird für die grobe Klassifizierung benutzt. Das untere Wort ist für feinere Unterscheidungen reserviert.

0: ST
1: STE
2: TT

Weitere von Atari benutzte Cookies

Neuere Systemsoftware bzw. Patchprogramme benutzen ebenfalls den Cookie Jar:

Cookie **Programm**

_FDC Dieser Cookie kann von Treibersoftware für Floppycontroller höherer Schreibdichte installiert werden. Das oberste Byte gibt über die Art der höchsten Schreibdichte im System Auskunft:

Wert	Bedeutung
0	normales Floppyinterface (zum Beispiel 720K bei doppelseitigen Disketten)
1	"High-Density" (HD, 1,44 MByte bei 3,5-Zoll-Disketten)
2	"Extra High Density" (ED, 2,88 MByte)

Damit ist natürlich noch nicht garantiert, daß tatsächlich ein solches Laufwerk angeschlossen und eine passende Diskette eingelegt ist!

Alle weiteren Werte sind für künftige Erweiterungen reserviert. Die restlichen drei Bytes beschreiben den Ursprung der Controller-Hardware:

Wert	Bedeutung
$000000	keine Information verfügbar
$415443	("ATC") Die Hardware ist entweder Teil des Systems oder ist nachträglich auf solche Art und Weise in das System integriert worden, daß sie sich genauso wie ein Teil der Originalhardware einer Atari-Maschine verhält.
$445031	("DP1") Die Hardware entstammt einem Bausatz der Firma "DreamPark Development". Alle anderen mit "DP" beginnenden Kennungen sind ebenfalls für diesen Hersteller reserviert.
andere	Hardwareentwickler können sich vom Atari-Entwickler-Support eigene Kennungen zuteilen lassen.

_FLK Wenn dieser Cookie existiert, dann verfügt das installierte Gemdos über File-Locking-Erweiterungen (Wert des Cookies ist die Versionsnummer der Erweiterung).

_INF STEFIX. Patchprogramm für Fehler im Desktop von TOS 1.06.

_NET Flag für GEMDOS-Netzwerkerweiterungen. Der Inhalt des Cookies ist ein Zeiger auf zwei LONG-Werte. Der erste enthält eine Herstellerkennung für das Netzwerk, der zweite die vom Hersteller vergebene Versionsnummer.

_OOL POOLFIX3. Patchprogramm für Fehler in GEMDOS-Version 0.15.

_SLM Diablo-Treiber für SLM-Laserdrucker (Wert zeigt auf nicht dokumentierte Struktur).

Außerdem benutzt auch der Maus-Beschleuniger "MACCEL3" den Cookie Jar.

Der Cookie selbst besteht jedoch nicht aus druckbaren ASCII-Zeichen.

Das XBRA-Verfahren für vektorverbiegende Programme

Speicherresidente Programme, die Vektoren verbiegen, haben stets zwei Probleme:

- Sie können den Vektor nur schwer zurücksetzen, da sich ja unter Umständen mittlerweile ein anderes Programm in den gleichen Vektor gehängt haben könnte.
- Sie können aus dem gleichen Grund nur schwer überprüfen, ob sie bereits installiert sind.

Das hier angegebene XBRA-Verfahren ("eXtended BRAner") wurde erstmals 1988 im "Atari ST Profibuch" vorgeschlagen und geht auf eine Idee von Moshe Braner zurück, die nur in einem Punkt etwas erweitert wurde. Jedes vektorverbiegende Programm plaziert direkt vor seiner eigenen Einsprungadresse (also genau vor der Adresse, auf die der Vektor gesetzt wurde) folgende Struktur:

```
typedef struct
{
     char xb_magic[4];    /* "XBRA" = 0x58425241 */
     char xb_id[4];       /* vier Buchstaben lange Kennung wie beim
                             Cookie Jar */
     LONG xb_oldvec;      /* ursprünglicher Wert des Vektors */
} XBRA;
```

Die Konstante "xb_magic" wurde hinzugefügt, um eine hundertprozentige Erkennung der XBRA-Struktur zu ermöglichen. Mit diesen zusätzlichen Informationen können Programme leicht feststellen, ob sie schon installiert sind, und sich leicht aus der Vektorkette ausklinken.

Ohne Übertreibung kann man sagen, daß sich das XBRA-Verfahren vollständig durchgesetzt hat. Mittlerweile wird es gemeinhin als schlechter Programmierstil angesehen, wenn man es nicht benutzt. Selbst Atari Amerika verbiegt in eigenen Utilities und Patchprogrammen die Vektoren mittlerweile XBRA-kompatibel.

Es ist erstrebenswert, daß es nicht zu Doppeltbelegungen von XBRA-Kennungen kommt. Daher führt der Autor die sogenannte "XBRA-Liste", in der offiziell gemeldete Kennungen eingetragen werden und die in regelmäßigen Abständen in Fachzeitschriften veröffentlicht wird. Anfragen bitte (vollständig mit Autor und Bezugsquelle für das betreffende Programm und einer Aufstellung der benutzten Systemadressen) an:

Julian Reschke
Redaktion ST-Magazin
Markt & Technik Verlag AG
Hans-Pinsel-Straße 2

8013 Haar bei München

BIOS-Referenz

Bconin (BIOS 2)

Liest ein Zeichen von einem Eingabegerät ein, sobald es verfügbar ist. Daher sollte man diese Funktion nach Möglichkeit zusammen mit "Bconstat()" verwenden!

Deklaration in C:

```
LONG Bconin (WORD dev);
```

Aufruf in Assembler:

```
move.w    dev,-(sp)  ; Offset 2
move.w    #2,-(sp)   ; Offset 0
trap      #13
addq.l    #4,sp
```

Parameter:

dev:	Nummer des betreffenden Eingabegeräts:	
	PRT (0):	parallele Schnittstelle
	AUX (1):	serielle Schnittstelle
	CON (2):	Konsole (VT-52)
	MIDI (3):	MIDI
Bconin():	Bits 0..7:	das eingelesene Zeichen

Bemerkungen

Für das Gerät "CON" enthalten die Bits 16..23 den Scancode der betreffenden Taste. Ist zusätzlich das entsprechende Bit in der Systemvariable conterm gesetzt, dann findet man in den Bits 24..31 den aktuellen Wert von "Kbshift()".

Bconout (BIOS 3)

Gibt ein Zeichen auf dem angegebenem Gerät aus (und kehrt erst dann zurück, wenn das Zeichen tatsächlich ausgegeben worden ist – also Vorsicht bei Benutzung des Druckers u.ä.).

Deklaration in C:

```
void Bconout (WORD dev, WORD c);
```

Aufruf in Assembler:

```
move.w    c,-(sp)    ; Offset 4
move.w    dev,-(sp)  ; Offset 2
move.w    #3,-(sp)   ; Offset 0
trap      #13
addq.l    #6,sp
```

Parameter:

dev: Nummer des betreffenden Ausgabegeräts:

PRT (0):	parallele Schnittstelle
AUX (1):	serielle Schnittstelle
CON (2):	Konsole (VT-52)
MIDI (3):	MIDI-Schnittstelle
IKBD (4):	Tastatur-Prozessor
RAWCON (5):	Konsole (ohne Terminalemulation)

c: Auszugebendes Zeichen

Bemerkungen

Man beachte, daß aufgrund eines Fehlers die mit "Setprt()" gemachten Einstellungen ignoriert werden.

Die Zeichenausgabe über "RAWCON" ist übrigens ein ganzes Stück schneller als die über "CON", da die VT52-Sequenzen nicht ausgewertet werden müssen.

Für den Drucker wird im Fehlerfall 0 zurückgeliefert.

Bconstat (BIOS 1)

Stellt den Status eines Eingabegerätes fest.

Deklaration in C:

```
WORD Bconstat (WORD dev);
```

Aufruf in Assembler:

```
move.w    dev,-(sp)  ; Offset 2
move.w    #1,-(sp)   ; Offset 0
trap      #13
addq.l    #4,sp
```

Parameter:

dev:	Nummer des betreffenden Eingabegeräts:	
	AUX (1):	serielle Schnittstelle
	CON (2):	Konsole (VT-52)
	MIDI (3):	MIDI-Port
Bconstat():	0:	kein Zeichen verfügbar
	−1:	mindestens ein Zeichen kann eingelesen werden

Bemerkungen
Die Geräte "AUX" und "MIDI" werden per Interrupt betrieben, mehr dazu unter "Iorec()".

Bcostat (BIOS 8)

Liefert den Ausgabestatus eines Ausgabegeräts.

Deklaration in C:

```
LONG Bcostat (WORD dev);
```

Aufruf in Assembler:

```
move.w    dev,-(sp) ; Offset 2
move.w    #8,-(sp)  ; Offset 0
trap      #13
addq.l    #4,sp
```

Parameter:

dev:	Nummer des betreffenden Ausgabegeräts:	
	PRT (0):	parallele Schnittstelle
	AUX (1):	serielle Schnittstelle
	CON (2):	Konsole (VT-52)
	IKBD (3):	Tastatur-Prozessor
	MIDI (4):	MIDI-Schnittstelle
	RAWCON (5):	Konsole (ohne Terminalemulation)
Bcostat():	0:	Zeichen kann noch nicht gesendet werden
	−1:	Zeichen kann gesendet werden

Bemerkungen
Man beachte die Vertauschung von "MIDI" und "IKBD" gegenüber den anderen BIOS-Funktionen! Laut Atari wird dies aus Kompatibilitätsgründen auch so bleiben.

Drvmap (BIOS 10)

Liefert einen Bitvektor der dem BIOS bekannten logischen Laufwerke.

Deklaration in C:

```
LONG Drvmap (void);
```

Aufruf in Assembler:

```
move.w    #$A,-(sp) ; Offset 0
trap      #13
addq.l    #2,sp
```

Parameter:

Drvmap(): Bitvektor mit den vorhandenen Laufwerken (A: Bit 0, B: Bit 1...).
Es sind also *32* Geräte möglich!

Bemerkungen
"Drvmap()" liefert den Inhalt der Systemvariable "_drvbits" zurück.

Normalerweise ist es interessanter, welche Laufwerke GEMDOS kennt. Um das festzustellen, sollte man besser die GEMDOS-Funktion "Dsetdrv()" einsetzen.

Getbpb (BIOS 7)

Liefert Adressen auf den BIOS-Parameter-Block des betreffenden Geräts (springt dazu durch den Vektor "hdv_bpb" ($472)):

```
typedef struct
{
     WORD  recsiz;  /* Bytes pro Sektor */
     WORD  clsiz;   /* Sektoren pro Cluster (Einheit) */
     WORD  clsizb;  /* Bytes pro Cluster */
     WORD  rdlen;   /* Länge des Wurzelverzeichnis in Sektoren */
     WORD  fsiz;    /* Länge des File Allocation Table (FAT) */
     WORD  fatrec;  /* Startsektor der zweiten FAT */
     WORD  datrec;  /* Sektornummer des ersten freien Clusters */
     WORD  numcl;   /* Gesamtzahl der Cluster auf dem Medium */
     WORD  bflags;  /* Bitvektor, zur Zeit nur Bit 0 belegt:
                       0 (12-Bit-FAT), 1 (16-Bit-FAT) */
} BPB;
```

Deklaration in C:

```
BPB *Getbpb (WORD dev);
```

Aufruf in Assembler:

```
move.w    dev,-(sp) ; Offset 2
move.w    #7,-(sp)  ; Offset 0
trap      #13
addq.l    #4,sp
```

Parameter:

dev: Gerätenummer (A: 0, B: 1, C: 2...)

Getbpb(): Zeiger auf den BPB des Geräts oder im Fehlerfall 0 (zum Beispiel, wenn die Informationen im Bootsektor den Verdacht nahelegen, daß die Diskette nicht korrekt formatiert ist)

Bemerkungen

"Getbpb()" setzt den Mediachange-Status im BIOS zurück. Weitere Hinweise dazu finden Sie in der Einführung zum GEMDOS.

Getmpb (BIOS 0)

Diese Funktion dient zur Initialisierung der Speicherverwaltung. Sie wird von GEMDOS ein einziges Mal beim Starten des Systems aufgerufen, um die Ursprungs-TPA zu erzeugen (mehr dazu in der GEMDOS-Einführung), und darf anschließend nicht noch einmal aufgerufen werden.

"Getmpb()" füllt eine Speicherparameter-Struktur (Memory Parameter Block), die folgende Gestalt hat:

```
typedef struct
{
     MD *mp_mfl;      /* Zeiger auf "Memory Free List" */
     MD *mp_mal;      /* Zeiger auf "Memory Allocated List" */
     MD *mp_rover;    /* roving pointer; wird nur intern benötigt */
}MPB;
```

Die MD-Struktur hat das folgende Format:

```
typedef struct md
{
     struct md *m_link;  /* Zeiger auf nächsten MD */
     LONG      m_start;  /* Anfangsadresse des Blocks */
     LONG      m_length; /* Länge des Blocks */
     BASEPAGE  *m_own;   /* Zeiger auf Prozeß-Beschreibungsstruktur
                            */
} MD;
```

Jede MD enthält also einen Zeiger auf eine weitere Struktur (verkettete Liste) sowie die Anfangsadresse und die Länge des entsprechenden Speicherabschnitts.

"m_own" zeigt auf die BASEPAGE-Struktur des Prozesses, dem der Speicherblock gehört.

Deklaration in C:

```
void Getmpb (MPB *p_mpb);
```

Aufruf in Assembler:

```
pea       p_mpb     ; Offset 2
move.w    #0,-(sp)  ; Offset 0
trap      #13
addq.l    #6,sp
```

Parameter:

p_mpb: zeigt auf eine vom Programm zur Verfügung gestellte MPB-Struktur

Kbshift (BIOS 11)

Liefert oder verändert den momentanen Tastaturstatus. Die Bitbelegung sieht folgendermaßen aus:

Bit	Belegung
Bit 0:	Shift-Taste rechts
Bit 1:	Shift-Taste links
Bit 2:	Control-Taste
Bit 3:	Alternate-Taste
Bit 4:	CapsLock gesetzt
Bit 5:	Maustaste rechts (ClrHome)
Bit 6:	Maustaste links (Insert)
Bit 7:	reserviert (0)

Deklaration in C:

```
LONG Kbshift (WORD mode);
```

Aufruf in Assembler:

```
move.w    mode,-(sp)      ; Offset 2
move.w    #$B,-(sp)       ; Offset 0
trap      #13
addq.l    #4,sp
```

Parameter:

mode:	entweder negativ (dann wird nur der aktuelle Status zurückgeliefert) oder >=0 (dann legt mode den neuen Status fest)
Kbshift():	aktueller (bzw. bisheriger) Tastaturstatus

Bemerkungen

"Kbshift()" fragt lediglich eine BIOS-interne Systemvariable ab. Deren Adresse kann man bei Bedarf (also zum Beispiel in Interrupt-Routinen) mit Hilfe von "_sysbase" berechnen.

Mediach (BIOS 9)

Stellt fest, ob ein Medienwechsel stattgefunden hat. Für Disketten sind folgende Hinweise zu beachten: die Erkennung eines Diskettenwechsels funktioniert nur dann korrekt, wenn die Diskette *nicht* schreibgeschützt ist. Im Zweifelsfall ist es wichtig, daß Disketten unterschiedliche Seriennummern tragen, was nicht bei allen Formatierprogrammen und auch nicht bei unter MS-DOS formatierten Disketten gewährleistet ist.

Doch nicht nur Disketten können gewechselt werden: auch bei Wechselplatten oder CD-ROMs (oder anderen "neuartigen" Geräten) kann es zu einem Medienwechsel kommen.

Daher: Niemals davon ausgehen, daß ein Gerät nicht wechselbar ist!

Deklaration in C:

```
LONG Mediach (WORD dev);
```

Aufruf in Assembler:

```
move.w    dev,-(sp) ; Offset 2
move.w    #9,-(sp)  ; Offset 0
trap      #13
addq.l    #4,sp
```

Parameter:

dev:	Nummer des Laufwerks (A: 0, B: 1, C: 2...)	
Mediach():	0:	Medium wurde mit Sicherheit nicht gewechselt
	1:	Medium wurde möglicherweise gewechselt
	2:	Medium wurde mit Sicherheit gewechselt
	(endlich einmal dreiwertige Logik!)	

Bemerkungen
Normalerweise wird diese Funktion vom GEMDOS aufgerufen. Dabei sind folgende Fälle möglich:

Return-Wert 0: Alles ist in Ordnung, und GEMDOS arbeitet normal weiter.

Return-Wert 1: GEMDOS macht einen Lesezugriff, um einen definitiven Status zu erfragen.

Return-Wert 2: GEMDOS erkennt den Medienwechsel, vergißt alle Informationen über das Laufwerk (siehe in der GEMDOS-Einführung zum Thema "Medienwechsel") und ruft "Getbpb()" auf. Dadurch wird dem BIOS signalisiert, daß der Medienwechsel verarbeitet ist und der Status zurückgesetzt werden kann.

Rwabs (BIOS 4)

Liest und schreibt logische Sektoren von Disketten oder anderen Geräten, die über die Harddisk-Vektoren (ab $46a, "hdv_init") installiert sind. Dazu gehören natürlich nicht nur Festplatten, sondern auch RAM-Disks und Cache-Programme. Diese BIOS-Routine springt dazu direkt durch den Vektor "hdv_rw" ($476).

Beim Betrieb von Festplatten kann man normalerweise nur auf die einzelnen Partitionen zugreifen. Um Festplatten partitionieren zu können, muß man allerdings direkten Zugriff auf alle Sektoren der Platte haben.

AHDI-3.0-kompatible Festplattentreiber kennen daher eine Erweiterung, die den direkten (physikalischen) Zugriff aus die Festplatten zuläßt.

Auch der Parameter "lrecno" ist eine Erweiterung, die es nur mit AHDI-3.0-kompatiblen Festplattentreibern gibt. Damit ist es möglich, auf mehr als 65535 Sektoren der Platte zuzugreifen (was bei Festplatten mit mehr als 32 MByte Größe natürlich sehr wichtig ist).

Wie stellt man fest, ob ein solcher Festplattentreiber installiert und für welche Geräte er zuständig ist?

Ganz einfach: Man inspiziert die Systemvariable "pun_ptr", die auf eine Struktur mit allen notwendigen Informationen zeigen sollte.

Deklaration in C:

```
LONG Rwabs (WORD rwflag, void *buf, WORD count, WORD recno,
            WORD dev, LONG lrecno);
```

Aufruf in Assembler:

```
move.l    lrecno,-(sp)    ; Offset 14
move.w    dev,-(sp)       ; Offset 12
move.w    recno,-(sp)     ; Offset 10
move.w    count,-(sp)     ; Offset 8
pea       buf             ; Offset 4
move.w    rwflag,-(sp)    ; Offset 2
move.w    #4,-(sp)        ; Offset 0
trap      #13
lea       $12(sp),sp
```

Parameter:

dev:	Im Normalmodus: Nummer des logischen Laufwerks (A: 0, B: 1, C: 2...). Im "physikalischen" Modus: 2 plus Gerätenummer (also 2..9 für ACSI-Platten, 10..17 für SCSI-Platten).
recno:	Nummer des ersten Sektors (bei –1 wird statt dessen "lrecno" benutzt)
lrecno:	dieser Parameter wird nur benutzt, wenn "recno" –1 ist und man einen AHDI-3.0-kompatiblen Festplattentreiber benutzt (siehe unter "pun_ptr")
count:	Anzahl der zu übertragenden Sektoren
buf:	Anfangsadresse des Puffers (darf auf ungerader Adresse liegen, was allerdings die Geschwindigkeit verringert!)
rwflag:	Bitvektor, der die Art der Operation festlegt: Bit 0: lesen (0) oder schreiben (1) Bit 1: Medienwechsel beachten (0) oder ignorieren (1) Bit 2: Im Fehlerfall "Retry" versuchen (0) oder nicht (1) (nur bei zu AHDI 3.0 kompatiblen Festplattentreibern) Bit 3: Normalmodus (0) oder "physikalischer" Modus (1) (nur bei zu AHDI 3.0 kompatiblen Festplattentreibern)
Rwabs():	0 (OK) oder eine BIOS-Fehlermeldung

Setexc (BIOS 5)

Setzt und liest die Inhalte von Exception-Vektoren.

Die Vektoren 2 bis 255 sind durch die Exceptions der CPU belegt. Ihre Vektoren liegen an den Adressen $8 bis einschließlich $3FF.

Die acht weiteren Vektoren 256 bis 263 sind für GEMDOS reserviert und liegen *zur Zeit* an den Adressen $400 bis $41F. Das muß natürlich nicht unbedingt so bleiben. Mehr zu den GEMDOS-Vektoren finden Sie im entsprechenden Abschnitt der GEMDOS-Einführung.

Deklaration in C:

```
LONG Setexc (WORD vecnum, void (*vec)());
```

Aufruf in Assembler:

```
pea       vec             ; Offset 4
move.w    vecnum,-(sp)    ; Offset 2
move.w    #5,-(sp)        ; Offset 0
trap      #13
addq.l    #8,sp
```

Parameter:

vecnum:	Nummer des zu setzenden Exception-Vektors (ist identisch mit der zu setzenden Adresse, dividiert durch 4)
vec:	Neue Adresse (oder –1, wenn diese nicht verändert werden soll)
Setexc():	Bisheriger (bzw. aktueller) Wert des Vektors

Tickcal (BIOS 6)

Liefert den Inhalt von "_timr_ms", also die Anzahl von Millisekunden zwischen zwei Aufrufen des Systemtimers.

Deklaration in C:

```
LONG Tickcal (void);
```

Aufruf in Assembler:

```
move.w    #6,-(sp)   ; Offset 0
trap      #13
addq.l    #2,sp
```

Parameter:

Tickcal(): Anzahl der Millisekunden

XBIOS-Referenz

Bconmap (XBIOS 44) – nicht auf allen TOS-Versionen

Mit dieser Funktion kann dem BIOS-Kanal 1 (serielle Schnittstelle) eine der erweiterten Kanalnummern zugeordnet werden. Die bisher zugeordnete Kanalnummer wird zurückgeliefert.

Als Sonderfall kann ein Zeiger auf die BCONMAP-Struktur abgefragt werden, über die man die maximal erlaubte BIOS-Gerätenummer abfragen und neue Gerätetreiber installieren kann (siehe Einleitung zum XBIOS).

"Bconmap()" beeinflußt einerseits die BIOS-Vektortabelle in den Systemvariablen (beginnend bei "xconstat", Adresse $51E). Andererseits wird auch das Verhalten von "Rsconf()" und "Iorec()" entsprechend modifiziert.

Deklaration in C:

```
LONG Bconmap (WORD devno);
```

Aufruf in Assembler:

```
move.w    devno,-(sp)     ; Offset 2
move.w    #$2C,-(sp)      ; Offset 0
trap      #14
addq.l    #4,sp
```

Parameter:

devno:	–2:	Zeiger auf BCONMAP-Struktur zurückliefern
	–1:	Nur aktuelle Kanalnummer abfragen
	0:	Test, ob "Bconmap()" vorhanden ist
	>=6:	Neue Kanalnummer
	bei ungültigen Angaben (kleiner –3 oder zwischen 0 und 5) wird 0 zurückgeliefert	
Bconmap():	für devno = –2:	Zeiger auf BCONMAP-Struktur
	sonst:	bisher eingestellte Kanalnummer

Bemerkungen
Laut Atari-Systemprogrammierer Allan Pratt sollte man seine TOS-Version folgendermaßen auf das Vorhandensein von "Bconmap()" testen:

```
WORD has_bconmap (void)
{
     return (0L == Bconmap (0));
}
```

Bioskeys (XBIOS 24)

Stellt die ursprüngliche Tastaturbelegung (die man mit "Keytbl()" ändern kann) wieder her.

Deklaration in C:

```
void Bioskeys (void);
```

Aufruf in Assembler:

```
move.w    #$18,-(sp)      ; Offset 0
trap      #14
addq.l    #2,sp
```

Blitmode (XBIOS 64) – erst ab TOS 1.02

Diese Funktion dient zur Konfiguration des Blitter-Chips (ab TOS 1.02).

Deklaration in C:

```
WORD Blitmode (WORD mode);
```

Aufruf in Assembler:

```
move.w    mode,-(sp)      ; Offset 2
move.w    #$40,-(sp)      ; Offset 0
trap      #14
addq.l    #4,sp
```

Parameter:

mode: –1: Status der Blitterregister auslesen
sonst: Blitterbetrieb wird umgeschaltet.
In Bit 0 wird folgende Information übergeben:
0: Blitter aus (wird durch Bildschirmtreiber nicht benutzt)
1: Blitter an (wird durch Bildschirmtreiber benutzt)
Bit 15 muß 0 sein

Blitmode(): Bisheriger Blitterstatus:
Bit 0: Blitter wird nicht benutzt (0)
Blitter wird benutzt (1)
Bit 1: kein Blitter installiert (0)
Blitter-Chip vorhanden
Bit 15:0
Alle weiteren Bits sind reserviert

Bemerkungen

Obwohl in TOS 1.00 nicht vorhanden, darf man laut Atari "Blitmode()" ohne Versionsabfrage benutzen (ein Seiteneffekt im XBIOS-Dispatcher macht es möglich). Darauf sollte man sich aber besser nicht verlassen – vielleicht gibt es ja Programme, die den XBIOS-Trap "verbiegen" und nicht darauf achten, daß für Opcode 64 der gleiche Return-Wert wie beim ROM zurückgeliefert wird.

Cursconf (XBIOS 21)

Legt verschiedene Cursor-Attribute fest oder fragt die Blinkfrequenz des Cursors ab.

Deklaration in C:

```
WORD Cursconf (WORD function, WORD operand);
```

Aufruf in Assembler:

```
move.w    operand,-(sp)   ; Offset 4
move.w    function,-(sp)  ; Offset 2
move.w    #$15,-(sp)      ; Offset 0
trap      #14
addq.l    #6,sp
```

Parameter:

function:	CURS_HIDE (0):	Cursor ausschalten
	CURS_SHOW (1):	Cursor einschalten
	CURS_BLINK (2):	Cursorblinken einschalten
	CURS_NOBLINK (3):	Cursorblinken ausschalten
	CURS_SETRATE (4):	Blinkrate auf "operand" stellen
	CURS_GETRATE (5):	Blinkrate abfragen
operand:	nur bei "CURS_SETRATE" benutzt; gibt an, nach wie vielen Vertical Blanks der Cursor einmal invertiert wird	
Cursconf():	bei "CURS_GETRATE" aktuelle Blinkrate	

DMAread (XBIOS 42) – nicht auf allen TOS-Versionen

Liest eine Anzahl von Sektoren von einem ACSI- oder SCSI-Gerät. Der zu übertragende Speicherbereich muß für die jeweilige Hardware beschreibbar sein (mehr dazu im Hardwareteil).

In den allermeisten Fällen ist es sinnvoller, statt dessen die Funktion "Rwabs()" zu verwenden!

Deklaration in C:

```
LONG DMAread (LONG sector, WORD count, void *buffer, WORD devno);
```

Aufruf in Assembler:

```
move.w    devno,-(sp)     ; Offset 12
pea       buffer          ; Offset 8
move.w    count,-(sp)     ; Offset 6
move.l    sector,-(sp)    ; Offset 2
move.w    #$2A,-(sp)      ; Offset 0
trap      #1
lea       $E(sp),sp
```

Parameter:

sector:	erste Sektornummer
count:	Anzahl der Sektoren
buffer:	Anfangsadresse im Speicher
devno:	Gerätenummer (0..7: ACSI-Geräte 0..7, 8..15: SCSI-Geräte 0..7, darüber: reserviert für künftige Erweiterungen)

DMAwrite(): liefert einen BIOS-Fehlercode zurück (0: OK)

Bemerkungen

Geräte am SCSI-Bus werden von dieser Funktion per Handshake (also nicht per DMA) betrieben.

DMAwrite (XBIOS 43) – nicht auf allen TOS-Versionen

Schreibt eine Anzahl von Sektoren auf ein ACSI- oder SCSI-Gerät. Der zu übertragende Speicherbereich muß für die jeweilige Hardware lesbar sein (mehr dazu im Hardwareteil).

In den allermeisten Fällen ist es sinnvoller, statt dessen die Funktion "Rwabs()" zu verwenden!

Deklaration in C:

```
LONG DMAwrite (LONG sector, WORD count, void *buffer, WORD devno);
```

Aufruf in Assembler:

```
move.w    devno,-(sp)     ; Offset 12
pea       buffer          ; Offset 8
move.w    count,-(sp)     ; Offset 6
move.l    sector,-(sp)    ; Offset 2
move.w    #$2B,-(sp)      ; Offset 0
trap      #1
lea       $E(sp),sp
```

Parameter:

sector:	erste Sektornummer
count:	Anzahl der Sektoren
buffer:	Anfangsadresse im Speicher
devno:	Gerätenummer (0..7: ACSI-Geräte 0..7, 8..15: SCSI-Geräte 0..7, darüber: reserviert für künftige Erweiterungen)

DMAwrite(): liefert einen BIOS-Fehlercode zurück (0: OK)

Bemerkungen

Geräte am SCSI-Bus werden von dieser Funktion per Handshake (also nicht per DMA) betrieben.

Dosound (XBIOS 32)

Mit dieser Funktion kann man vollautomatisch eine Reihe von Werten in die Register des Soundchips schreiben lassen.

Deklaration in C:

```
void Dosound (const char *ptr);
```

Aufruf in Assembler:

```
pea       ptr             ; Offset 2
move.w    #$20,-(sp)      ; Offset 0
trap      #14
addq.l    #6,sp
```

Parameter:

ptr: Zeiger auf eine Bytefolge mit Soundbefehlen. Folgende Befehlscodes werden unterstützt:

$0x byte: Byte in Register x des Soundchips schreiben

$80 byte: Byte in temporäres Register laden

$81 byte1 byte2 byte3 (drei Byte-Argumente!)

- byte1: Nummer des Soundregisters, in das der Wert des temporären Registers übertragen werden soll
- byte2: wird jeweils zum temporären Register addiert
- byte3: Endwert des temporären Registers, bei dem die "Schleife" abgebrochen wird

$82-$FF byte: Anzahl der Jiffies (20 ms), die bis zum nächsten Kommando vergehen sollen (bei byte = 0 wird vollständig abgebrochen)

EgetPalette (XBIOS 85) – nur für TT-Videohardware

Liest einen zusammenhängenden Bereich aus den TT-Farbregistern aus.

Deklaration in C:

```
void EgetPalette (WORD colorNum, WORD count, WORD *palettePtr);
```

Aufruf in Assembler:

```
pea       palettePtr      ; Offset 6
move.w    count,-(sp)     ; Offset 4
move.w    colorNum,-(sp)  ; Offset 2
move.w    #$55,-(sp)      ; Offset 0
trap      #14
lea       $A(sp),sp
```

Parameter:

colorNum:	Nummer des ersten auszulesenden Farbregisters
count:	Anzahl der auszulesenden Farbregister
palettePtr:	Zeiger auf zu übertragende Farbtabelle (muß auf eine gerade Adresse zeigen)

EgetShift (XBIOS 81) – nur für TT-Videohardware

Fragt den Wert des Shiftmode-Registers im Video-Shifter des TT ab.

Deklaration in C:

```
WORD EgetShift (void);
```

Aufruf in Assembler:

```
move.w    #$51,-(sp)      ; Offset 0
trap      #14
addq.l    #2,sp
```

Parameter:

EgetShift(): Wert für das Modus-Register des TT-Shifters. Bitbelegung:

Bit 15:	Smear-Modus (siehe unter "EsetSmear()")
Bit 12:	Hyper Mono (siehe unter "EsetGray()")
Bit 10..8:	Modus, wie er auch von "Getrez()" zurückgeliefert wird
Bit 3..0:	Nummer der Farbregister-Bank (siehe unter "EsetBank()")

EsetBank (XBIOS 82) – nur für TT-Videohardware

Die 256 Farbregister des TT sind in 16 verschiedene Bänke unterteilt. Mit diesem Kommando kann man die aktive Bank auswählen.

Deklaration in C:

```
WORD EsetBank (WORD bankNum);
```

Aufruf in Assembler:

```
move.w    bankNum,-(sp)   ; Offset 2
move.w    #$52,-(sp)      ; Offset 0
trap      #14
addq.l    #4,sp
```

Parameter:

bankNum: Nummer der neuen aktiven Bank (0..15). Bei einem negativen Wert wird die Banknummer nicht geändert.
EsetBank(): bisherige Banknummer

EsetColor (XBIOS 83) – nur für TT-Videohardware

Setzt (sofort, nicht erst im nächsten Vertical Blank) eines der 256 Farbregister des TT-Shifters.

Deklaration in C:

```
WORD EsetColor (WORD colorNum, WORD color);
```

Aufruf in Assembler:

```
move.w    color,-(sp)     ; Offset 4
move.w    colorNum,-(sp)  ; Offset 2
move.w    #$53,-(sp)      ; Offset 0
trap      #14
addq.l    #6,sp
```

Parameter:

colorNum:	Nummer des zu verändernden Farbregisters (0..255). Bei negativen Werten wird die eingestellte Farbe nicht verändert.
color	neue Farbe
EsetColor():	bisheriger Wert

EsetGray (XBIOS 86) – nur für TT-Videohardware

Der TT-Videochip kennt auch einen Graustufen-Modus, in dem man anstelle von 4096 Farbtönen aus einer Graustufenpalette von 256 Tönen auswählen kann. In diesem Modus wird jeweils nur das untere Byte eines Eintrags in einem Farbregister benutzt.

Deklaration in C:

```
WORD EsetGray (WORD switch);
```

Aufruf in Assembler:

```
move.w    switch,-(sp)    ; Offset 2
move.w    #$56,-(sp)      ; Offset 0
trap      #14
addq.l    #4,sp
```

Parameter:

switch: ungleich 0: Graustufenmodus einschalten
EsetGray(): alter Wert

EsetPalette (XBIOS 84) – nur für TT-Videohardware

Setzt (sofort) einen zusammenhängenden Bereich von Farbregistern in der Farbpalette des TT-Shifters.

Deklaration in C:

```
void EsetPalette (WORD colorNum, WORD count, WORD *palettePtr);
```

Aufruf in Assembler:

```
pea       palettePtr     ; Offset 6
move.w    count,-(sp)    ; Offset 4
move.w    colorNum,-(sp) ; Offset 2
move.w    #$54,-(sp)     ; Offset 0
trap      #14
lea       $A(sp),sp
```

Parameter:

colorNum:	Nummer des ersten zu ändernden Farbregisters
count:	Anzahl der zu setzenden Farbregister
palettePtr:	Zeiger auf zu übertragende Farbtabelle (muß auf eine gerade Adresse zeigen)

EsetShift (XBIOS 80) – nur für TT-Videohardware

Setzt das Shiftmode-Register im Video-Shifter des TT.

Deklaration in C:

```
WORD EsetShift (WORD shftMode);
```

Aufruf in Assembler:

```
move.w    newmode,-(sp)   ; Offset 2
move.w    #$50,-(sp)      ; Offset 0
trap      #14
addq.l    #4,sp
```

Parameter:

shftMode: Neuer Wert für das Modus-Register des TT-Shifters. Bitbelegung:
- Bit 15: Smear-Modus (siehe unter "EsetSmear()")
- Bit 12: Hyper Mono (siehe unter "EsetGray()")
- Bit 10..8: Modus, wie er auch von "Getrez()" zurückgeliefert wird
- Bit 3..0: Nummer der Farbregister-Bank (siehe unter "EsetBank()")

EsetShift(): alter Wert des Registers

EsetSmear (XBIOS 87) – nur für TT-Videohardware

Mit dieser Funktion kann der Smear-Modus des TT-Videobausteins umgeschaltet werden. Im Smear-Modus wird anstelle der Hintergrundfarbe (Farbe 0) die jeweils zuletzt dargestellte Farbe gezeichnet.

Deklaration in C:

```
WORD EsetSmear (WORD switch);
```

Aufruf in Assembler:

```
move.w    switch,-(sp)    ; Offset 2
move.w    #$57,-(sp)      ; Offset 0
trap      #14
addq.l    #4,sp
```

Parameter:

switch:	0:	Smear-Modus aus
	>0:	Smear-Modus ein
	<0:	nur den bisherigen Wert abfragen

EsetSmear(): bisheriger Wert

Flopfmt (XBIOS 10)

Formatiert eine Spur auf einer Diskette.

Deklaration in C:

```
WORD Flopfmt (void *buf, LONG filler, WORD devno, WORD spt,
              WORD trackno, WORD sideno, WORD interlv, LONG magic,
              WORD virgin);
```

Aufruf in Assembler:

```
move.w    virgin,-(sp)    ; Offset 24
move.l    magic,-(sp)     ; Offset 20
move.w    interlv,-(sp)   ; Offset 18
move.w    sideno,-(sp)    ; Offset 16
move.w    trackno,-(sp)   ; Offset 14
move.w    spt,-(sp)       ; Offset 12
move.w    devno,-(sp)     ; Offset 10
move.l    #0,-(sp)        ; Offset 6
pea       buf             ; Offset 2
move.w    #$A,-(sp)       ; Offset 0
trap      #14
lea       $1A(sp),sp
```

Parameter:

buf: Zeiger auf einen Speicherbereich, in dem die Daten für die Spur temporär gespeichert werden können (bei neun Sektoren pro Spur mindestens 8 KByte, muß bei höheren Sektorzahlen entsprechend vergrößert werden)

filler: bei den "alten" TOS-Versionen unbenutzt (0).
Ab TOS 1.02 kommt "filler" eine neue Bedeutung zu. Übergibt man als "interlv" –1, dann wird "filler" als Zeiger auf eine Tabelle von Sektornummern benutzt (16-Bit-Words). Damit hat man also die Möglichkeit, die Reihenfolge der Sektoren auf der Spur frei zu wählen.

devno: 0: Laufwerk A:
1: Laufwerk B:

spt: Sektoren pro Spur (normalerweise 9). Wenn der "_FDC"-Cookie gesetzt ist, sind auch 18 Sektoren ("High-Density") bzw. 36 Sektoren ("Extra-High-Density") erlaubt.

Die Umschaltung zwischen den verschiedenen Schreibverfahren findet bei 13 (HD) bzw. 26 (ED) statt.

trackno: Nummer der Spur (0..79 oder ggfs. etwas darüber)

sideno: 0 oder 1 (nur bei doppelseitigen Disketten)

interlv: Bestimmt, wie viele physikalische Sektoren jeweils zwischen zwei logischen Sektoren liegen (normalerweise einer)

magic: $87654321, sonst wird nicht formatiert

virgin: Dieses Bitmuster wird beim Formatieren in jeden Sektor hineingeschrieben (normalerweise $e5e5, was jeweils abwechselnd gesetzte Bits bedeutet). Die oberen vier Bits dürfen auf keinen Fall gleichzeitig gesetzt sein, da dieses Bitmuster vom Disk-Controller als Kommando interpretiert und daher zu Chaos führen würde.

Flopfmt(): 0: alles o.k.

Bei Formatierfehlern wird eine durch 0 abgeschlossene Liste der fehlerhaften Sektoren in den Puffer geschrieben.

Bemerkungen

Auch das Desktop nutzt ab TOS 1.02 die Möglichkeit, eine Liste von Sektornummern zu übergeben. Dies erlaubt es, die Spuren untereinander so zu "spiralisieren", daß beim Spurwechsel möglichst wenig Zeit mit dem Warten auf den "nächsten" Sektor vertan wird.

Floprate (XBIOS 41) – erst ab TOS 1.04

Mit dieser TOS-Funktion kann man ab TOS 1.04 die Seekraten (Spurwechselzeiten) für beide Laufwerke setzen und abfragen. Auf älteren ROM-TOS-Versionen muß man die vorher undokumentierten Systemvariablen verwenden:

TOS-Version	Laufwerk A	Laufwerk B
1.00	$A08	$A0C
1.02	$A4E	$A52

Die Seekrate kann folgende Werte annehmen:

Wert	Seekrate
0:	6ms
1:	12ms
2:	2ms
3:	3ms

Deklaration in C:

```
WORD Floprate (WORD drive, WORD seekrate);
```

Aufruf in Assembler:

```
move.w    seekrate,-(sp) ; Offset 4
move.w    drive,-(sp)    ; Offset 2
move.w    #$29,-(sp)     ; Offset 0
trap      #14
addq.l    #6,sp
```

Parameter:

drive: Nummer des Laufwerks (0: A, 1: B)
seekrate: neue Seekrate oder –1, wenn die Seekrate nicht verändert werden soll
Floprate(): vorherige Seekrate

Bemerkungen
Auf der folgenden Seite sehen Sie eine Beispielroutine zum portablen Setzen der Seekrate.

```
/*   Portables Setzen der Floppy-Seekrate für alle TOS-Versionen
     (außer RAM-TOS 1.00).
      Parameter: genau wie bei XBIOS-Funktion "Floprate()" */

WORD SeekRate (WORD driv, WORD set)
{
     LONG stack;
     WORD version;
     OSHEADER *sys;

     /* Zeiger auf OS-Header holen */
     stack = Super (0L);
     sys = *((OSHEADER **)0x4f2);
     version = sys->os_version;
     Super ((void *)stack);
     /* bei neuem TOS einfach "Floprate()" aufrufen */
     if (version >= 0x104)
          return Floprate (driv, set);
     else
     {
          /* sonst Zeiger auf interne GEMDOS-Variablen
          berechnen */
          WORD *sk, merk;

          if (version == 0x102)
               sk = (WORD *)0xa4e;
          else
               sk = (WORD *)0xa08;

          /* Laufwerk B: 2 WORDS dahinter */

          if (driv) sk = &(sk[2]);

          merk = *sk;
          /* Wert nur bei ungleich -1 eintragen */
          if (set != -1) *sk = set;

          /* alten Wert immer zurückliefern */
          return merk;
     }
}
```

Floprd (XBIOS 8)

Liest einen oder mehrere physikalische Sektoren von einer Diskette (vgl. BIOS-Funktion "Rwabs()").

Deklaration in C:

```
WORD Floprd (void *buf, LONG filler, WORD devno, WORD sectno,
             WORD trackno, WORD sideno, WORD count);
```

Aufruf in Assembler:

```
move.w    count,-(sp)     ; Offset 18
move.w    sideno,-(sp)    ; Offset 16
move.w    trackno,-(sp)   ; Offset 14
move.w    sectno,-(sp)    ; Offset 12
move.w    devno,-(sp)     ; Offset 10
move.l    #0,-(sp)        ; Offset 6 (unbenutzt)
pea       buffer          ; Offset 2
move.w    #8,-(sp)        ; Offset 0
trap      #14
lea       $14(sp),sp
```

Parameter:

buffer:	Zeiger auf Speicherbereich für die eingelesenen Sektoren
filler:	unbenutzt
devno:	0 (Laufwerk A:), 1 (Laufwerk B:)
sectno:	Nummer des Startsektors (normalerweise zwischen 1 und 9)
trackno:	Nummer der Spur (normalerweise zwischen 0 und 79)
sideno:	Seite der Diskette (0 bei einseitigen, 0 oder 1 sonst)
count:	Anzahl der zu lesenden Sektoren (die alle auf der gleichen Spur liegen müssen)
Floprd():	0: mit Erfolg abgeschlossen

Flopver (XBIOS 19)

Liest eine Reihe von physikalischen Sektoren von der Diskette und vergleicht mit einem gegebenen Speicherbereich.

Deklaration in C:

```
WORD Flopver (void *buf, LONG filler, WORD devno, WORD sectno,
              WORD trackno, WORD sideno, WORD count)
```

Aufruf in Assembler:

```
move.w    count,-(sp)     ; Offset 18
move.w    sideno,-(sp)    ; Offset 16
move.w    trackno,-(sp)   ; Offset 14
move.w    sectno,-(sp)    ; Offset 12
move.w    devno,-(sp)     ; Offset 10
move.l    #0,-(sp)        ; Offset 6 (unbenutzt)
pea       buf             ; Offset 2
move.w    #$13,-(sp)      ; Offset 0
trap      #14
lea       $14(sp),sp
```

Parameter:

buf:	Adresse des zu vergleichenden Speicherbereiches
filler:	unbenutzt (kann auf 0 gesetzt werden)
devno:	Nummer des Diskettenlaufwerks (0: A, 1: B)
sectno:	Nummer des Startsektors (normalerweise 1..9)
trackno:	Nummer der Spur (normalerweise 0..79)
sideno:	Diskettenseite (0..1)
count:	Anzahl der zu vergleichenden Sektoren (der letzte Sektor muß auf der gleichen Spur liegen wie der Startsektor)
Flopver():	0: kein Fehler Ansonsten findet man bei "buf" eine (durch 0 abgeschlossene) Liste der fehlerhaften Sektoren.

Flopwr (XBIOS 9)

Schreibt einen oder mehrere physikalische Sektoren auf eine Diskette.

Deklaration in C:

```
WORD Flopwr (void *buf, LONG filler, WORD devno, WORD sectno,
             WORD trackno, WORD sideno, WORD count);
```

Aufruf in Assembler:

```
move.w    count,-(sp)     ; -> 24(sp)
move.w    sideno,-(sp)    ; Offset 16
move.w    trackno,-(sp)   ; Offset 14
move.w    sectno,-(sp)    ; Offset 12
move.w    devno,-(sp)     ; Offset 10
move.l    #0,-(sp)        ; Offset 6 (unbenutzt)
pea       buffer          ; Offset 2
move.w    #9,-(sp)        ; Offset 0
trap      #14
lea       $14(sp),sp
```

Parameter:

buffer:	Zeiger auf Speicherbereich mit den zu schreibenden Sektoren
filler:	unbenutzt
devno:	0 (Laufwerk A:), 1 (Laufwerk B:)
sectno:	Nummer des Startsektors (normalerweise zwischen 1 und 9)
trackno:	Nummer der Spur (normalerweise zwischen 0 und 79)
sideno:	Seite der Diskette (0 bei einseitigen, 0 oder 1 sonst)
count:	Anzahl der zu schreibenden Sektoren (die alle auf der gleichen Spur liegen müssen)
Flopwr():	0: mit Erfolg abgeschlossen

Getrez (XBIOS 4) – nur für ST/STE/TT-Videohardware

Liefert die aktuelle Bildschirmauflösung.

Deklaration in C:

```
WORD Getrez (void);
```

Aufruf in Assembler:

```
move.w    #4,-(sp)   ; Offset 0
trap      #14
addq.l    #2,sp
```

Parameter:

Getrez(): 0: 320 * 200 (vier Planes)
1: 640 * 200 (zwei Planes)
2: 640 * 400 (ein Plane)
4: 640 * 480 (vier Planes, nur beim TT)
6: 1280 * 960 (ein Plane, nur beim TT)
7: 320 * 480 (acht Planes, nur beim TT)
Alle anderen Werte sind für künftige Erweiterungen reserviert.

Bemerkungen
Sinnvoller ist es, statt dessen die Rückgabewerte der VDI-Funktion "v_opnvwk()" auszuwerten! Speziell dann, wenn eine Grafikkarte aktiv ist, hat der Rückgabewert von "Getrez()" de facto keinen Aussagewert!

Gettime (XBIOS 23)

Liest aus der Hardwareuhr das aktuelle Datum.

Deklaration in C:

```
LONG Gettime (void);
```

Aufruf in Assembler:

```
move.w    #$17,-(sp)      ; Offset 0
trap      #14
addq.l    #2,sp
```

Parameter:

Gettime(): Aktuelles Datum:

Bit 0..4:	Sekunden (mit 2 zu multiplizieren)
Bit 5..10:	Minuten
Bit 11..15:	Stunden
Bit 16..20:	Tag
Bit 21..24:	Monat
Bit 25..31:	Jahr (1980 addieren!)

Giaccess (XBIOS 28)

Setzt und liest Register im General-Instruments-Soundchip.

Deklaration in C:

```
char Giaccess (char data, WORD regno);
```

Aufruf in Assembler:

```
move.w    regno,-(sp)     ; Offset 4
move.w    data,-(sp)      ; Offset 2
move.w    #$1C,-(sp)      ; Offset 0
trap      #14
addq.l    #6,sp
```

Parameter:

data:	beim Schreibzugriff wird dieser 8-Bit-Wert in das entsprechende Register geschrieben
regno:	Nummer des GI-Registers (0..15). Beim Schreibzugriff wird zusätzlich Bit 7 gesetzt, so daß sich dabei die Werte 128..143 ergeben.
Giacess():	Wert des angegebenen GI-Registers

Ikbdws (XBIOS 25)

Gibt eine Zeichenkette an den IKBD-Chip aus.

Deklaration in C:

```
void Ikbdws (WORD cnt, const char *ptr);
```

Aufruf in Assembler:

```
pea       ptr             ; Offset 4
move.w    cnt,-(sp)       ; Offset 2
move.w    #$19,-(sp)      ; Offset 0
trap      #14
addq.l    #8,sp
```

Parameter:

cnt: Anzahl der auszugebenden Bytes minus 1
ptr: Zeiger auf die Bytefolge im Speicher

Initmous (XBIOS 0)

Initialisiert die Mauszeigerroutinen.

Deklaration in C:

```
void Initmous (WORD type, PARAM *param, void *(*vec)());
```

Aufruf in Assembler:

```
pea       vec             ; Offset 8
pea       param           ; Offset 4
move.w    type,-(sp)      ; Offset 2
move.w    #0,-(sp)        ; Offset 0
trap      #14
lea       12(sp),sp
```

Parameter:

type:
- 0: Maus ausschalten
- 1: Maus einschalten, relativer Modus
- 2: Maus einschalten, absoluter Modus
- 3: unbenutzt
- 4: Maus einschalten, Tastaturemulation

param: Zeiger auf eine Struktur folgenden Typs:

```
typedef struct
{
        BYTE topmode;
        BYTE buttons;
        BYTE xparam;
        BYTE yparam;
} PARAM;
```

wobei die Strukturelemente folgende Bedeutung haben:

topmode:
- 0: Y = 0 am unteren Rand
- 1: Y = 1 am oberen Rand

buttons: siehe IKBD-Beschreibung

xparam, yparam: Zusätzliche Parameter, abhängig vom ausgewählten Modus

Für den absoluten Modus muß sich direkt die folgende Struktur anschließen:

```
typedef struct
{
          WORD xmax;       /* Maximale X-Position */
          WORD ymax;       /* Maximale Y-Position */
          WORD xinitial;   /* Anfangsposition X */
          WORD yinitial;   /* Anfangsposition Y */
} EXTRA;
```

vec: Zeiger auf Maustreiberroutine

Iorec (XBIOS 14)

Liefert einen Zeiger auf eine gerätespezifische Informationsstruktur folgenden Formats:

```
typedef struct
{
     LONG ibuf;      /* Zeiger auf den Puffer */
     WORD ibufsiz;   /* Länge des Puffers */
     WORD ibufhd;    /* nächste Schreibposition */
     WORD ibuftl;    /* nächste Leseposition */
     WORD ibuflow;   /* "untere Wassermarke" */
     WORD ibufhi;    /* "obere Wassermarke" */
} IOREC;
```

Für die serielle Schnittstelle schließt sich direkt ein entsprechender Puffer für die Ausgabe an.

Die beiden letzten Zeiger werden nur für die serielle Schnittstelle im XON/XOFF- oder RTS/CTS-Betrieb genutzt. Fällt der "Zeichenpegel" im Puffer unter die "untere Wassermarke", wird das Sendegerät zum Senden weiterer Zeichen aufgefordert. Übersteigt der Pegel die "obere Wassermarke", wird das Sendegerät aufgefordert, keine weiteren Zeichen zu senden.

Deklaration in C:

```
IOREC *Iorec (WORD devno);
```

Aufruf in Assembler:

```
move.w    devno,-(sp)     ; Offset 2
move.w    #$E,-(sp)       ; Offset 0
trap      #14
addq.l    #4,sp
```

Parameter:

devno:	Gerätenummer:
	0: RS 232
	1: IKBD (Tastatur)
	2: MIDI
Iorec():	Zeiger auf IOREC-Struktur

Jdisint (XBIOS 26)

Sperrt den angegebenen Interrupt des MFP 68901.

Deklaration in C:

```
void Jdisint (WORD intno);
```

Aufruf in Assembler:

```
move.w    intno,-(sp)     ; Offset 2
move.w    #$1A,-(sp)      ; Offset 0
trap      #14
addq.l    #4,sp
```

Parameter:

intno: Nummer des zu sperrenden Interrupts (0..15)

Jenabint (XBIOS 27)

Schaltet den angegebenen Interrupt des MFP 68901 ein.

Deklaration in C:

```
void Jenabint (WORD intno);
```

Aufruf in Assembler:

```
move.w    intno,-(sp)     ; Offset 2
move.w    #$1B,-(sp)      ; Offset 0
trap      #14
addq.l    #4,sp
```

Parameter:

intno: Nummer des betreffenden Interrupts (0..15)

Kbdvbase (XBIOS 34)

Liefert einen Zeiger auf eine Struktur folgender Form:

```
typedef struct
{
     void (*midivec)();  /* Midi-Eingabe */
     void (*vkbderr)();  /* Tastatur-Fehler */
     void (*vmiderr)();  /* MIDI-Fehler */
     void (*statvec)();  /* Status von IKBD lesen */
     void (*mousevec)(); /* Mausabfrage */
     void (*clockvec)(); /* Uhrzeitabfrage */
     void (*joyvec)();   /* Joystickabfrage */
     void (*midisys)();  /* MIDI-Systemvektor */
     void (*ikbdsys)();  /* IKBD-Systemvektor */
     WORD drvstat;       /* IKBD-Treiberstatus */
} KBDVECS;
```

midivec: Routine, die vom MIDI-Port empfangene Bytes (in D0) in einen Puffer schreibt

vkbderr, vmiderr: werden beim Überlauf von Daten von der Tastatur oder vom MIDI-Port aufgerufen

statvec, mousevec, clockvec, joyvec: Zeiger auf die Routinen, die die entsprechenden Informationspakete von MIDI oder IKBD verarbeiten.
Die Adresse des "packets" wird in A0 und auf dem Stack übergeben.
Die Routinen sollten mit einem RTS abgeschlossen sein und nicht länger als 1 ms laufen.

midisys, ikbdsys: werden beim Eintreffen von Daten am entsprechenden Port aufgerufen. "midisys" springt indirekt durch "midivec"; "ikbdsys" verzweigt in eine der vier in Frage kommenden Unterroutinen (s.o.).

drvstat: Wenn diese Statusvariable ungleich 0 ist, dann verschickt der IKBD gerade ein "packet".

Deklaration in C:

```
KBDVECS *Kbdvbase (void);
```

Aufruf in Assembler:

```
move.w    #$22,-(sp)      ; Offset 0
trap      #14
addq.l    #2,sp
```

Parameter:

Kbdvbase(): Zeiger auf KBDVECS-Struktur

Bemerkungen

Bevor man entweder Daten in eine der Routinen "einspeisen" oder einen der Vektoren ändern kann, muß man sichergehen, daß momentan nicht gerade ein "packet" verschickt wird. Dann sollte man alle Interrupts sperren und anschließend nochmals testen, ob tatsächlich kein "packet" mehr unterwegs ist. Erst dann darf man weiterarbeiten!

Kbrate (XBIOS 35)

Setzt die Tastatur-Wiederholrate oder fragt sie ab.

Deklaration in C:

```
WORD Kbrate (WORD initial, WORD repeat);
```

Aufruf in Assembler:

```
move.w    repeat,-(sp)    ; Offset 4
move.w    initial,-(sp)   ; Offset 2
move.w    #$23,-(sp)      ; Offset 0
trap      #14
addq.l    #6,sp
```

Parameter:

initial:	Ansprechzeit bis zur ersten Wiederholung (initial/(20 ms)) oder –1 (keine Änderung)	
repeat:	Wiederholgeschwindigkeit (repeat/(20 ms)) oder –1 (keine Änderung)	
Kbrate():	Bit 0..7:	bisheriger Wert von "repeat"
	Bit 8..15:	bisheriger Wert von "initial"

Keytbl (XBIOS 16)

Setzt die Adressen der Tastencode-Umwandlungstabellen. Ein Code wird umgerechnet, indem der Scancode der betreffenden Taste als Index in die Tabelle (von ASCII-Zeichen) verwendet wird. Zurückgeliefert wird ein Zeiger auf eine Struktur folgender Form:

```
typedef struct
{
     char *unshift;   /* Tabelle für "normale" Tastendrücke */
     char *shift;     /* Tabelle für Tastendrücke mit "Shift" */
     char *capslock;  /* Tabelle für Tastendrücke bei "Capslock" */
} KEYTAB;
```

Deklaration in C:

```
KEYTAB *Keytbl (char *unshift, char *shift, char *capslock);
```

Aufruf in Assembler:

```
pea       capslock       ; Offset 10
pea       shift          ; Offset 6
pea       unshift        ; Offset 2
move.w    #$10,-(sp)     ; Offset 0
trap      #14
lea       $E(sp),sp
```

Parameter:

unshift:	Zeiger auf "unshift"-Tabelle (oder –1, wenn keine neue Adresse gesetzt werden soll)
shift:	Zeiger auf "shift"-Tabelle (oder –1, wenn keine neue Adresse gesetzt werden soll)
capslock:	Zeiger auf "capslock"-Tabelle (oder –1, wenn keine neue Adresse gesetzt werden soll)
Keytbl():	Zeiger auf KEYTAB-Struktur

Bemerkungen

Jede der Tabellen hat nur 128 Einträge, obwohl es einige Tastenkombinationen mit höherem Scancode gibt (siehe Tabelle).

Logbase (XBIOS 3) – nur für ST/STE/TT-Hardware

Liefert die Anfangsadresse des logischen Bildschirmspeichers (das ist derjenige, der bei allen Grafikoperationen angesprochen wird).

Deklaration in C:

```
void *Logbase (void);
```

Aufruf in Assembler:

```
move.w    #3,-(sp)   ; Offset 0
trap      #14
addq.l    #2,sp
```

Parameter:

Logbase(): Anfangsadresse des logischen Bildschirmspeichers

Metainit (XBIOS 48)

Mit dieser Funktion kann man Informationen über die aktuell installierte Meta-DOS-Version (und die von ihr installierten Geräte) erfragen. Da nicht jeder Meta-DOS benutzt und es deshalb nicht überall installiert ist, muß man folgendermaßen vorgehen:

1. METAINFO-Struktur vollständig löschen (also 16 Nullbytes).
2. Metainit() aufrufen
3. Testen, ob "version" noch immer einen Nullzeiger enthält (dann nämlich ist Meta-DOS nicht installiert)

```
typedef struct
{
     ULONG     drivemap; /* Tabelle mit Bits für die Meta-DOS-
                            Gerätetreiber "A".."Z" (Bit 0: "A") */
     char      *version; /* Zeichenkette mit Namen und
                            Versionsnummer von Meta-DOS */
     LONG      reserved[2];
} METAINFO;
```

Damit läßt sich schon mal feststellen, ob Meta-DOS installiert ist und welche physikalischen Gerätenummern belegt sind. Diese Geräte kann man dann mit weiteren, für Meta-DOS reservierten XBIOS-Aufrufen (Nummern 48 bis 63) ansteuern.

Deklaration in C:

```
void Metainit (METAINFO *buffer);
```

Aufruf in Assembler:

```
pea       buffer          ; Offset 2
move.w    #$30,-(sp)      ; Offset 0
trap      #14
addq.l    #6,sp
```

Parameter:

buffer: Zeiger auf METAINFO-Struktur, die von Meta-DOS ausgefüllt wird

Mfpint (XBIOS 13)

Setzt die Interrupt-Vektoren des MFP 68901. Dabei wird jeweils der alte Wert überschrieben.

Deklaration in C:

```
void Mfpint (WORD interno, void (*vector)());
```

Aufruf in Assembler:

```
pea       vector          ; Offset 4
move.w    interno,-(sp)   ; Offset 2
move.w    #$D,-(sp)       ; Offset 0
trap      #14
addq.l    #8,sp
```

Parameter:

interno:	Nummer des zu ändernden Vektors (0..15)
vector:	Neue Adresse für die betreffende Interrupt-Routine

Midiws (XBIOS 12)

Gibt eine Bytefolge auf dem MIDI-Port aus.

Deklaration in C:

```
void Midiws (WORD cnt, const char *ptr);
```

Aufruf in Assembler:

```
pea       ptr             ; Offset 4
move.w    cnt,-(sp)       ; Offset 2
move.w    #$C,-(sp)       ; Offset 0
trap      #14
addq.l    #8,sp
```

Parameter:

cnt: Anzahl der auszugebenden Bytes minus 1
ptr: Zeiger auf Bytefolge

NVMaccess (XBIOS 46) – nur auf dem TT

Die Echtzeituhr des TT verfügt über 50 Bytes nichtflüchtiges ("non volatile") RAM, das man daher sehr gut für Konfigurationszwecke benutzen kann. Die letzten beiden Bytes werden als Prüfsumme benutzt. "NVMaccess()" dient zur Verwaltung dieses RAMs.

Die Belegung des NVMs ("non volatile memory") wird von Atari festgelegt. Die Benutzung für nicht von Atari dokumentierte Zwecke kann und wird zu schwerwiegenden Problemen führen!

Bislang ist über die Verwendung des NVMs noch nicht entschieden. Im Bedarfsfall sollte man der zuständigen Atari-Vertretung Vorschläge unterbreiten.

Deklaration in C:

```
WORD NMVaccess (WORD op, WORD start, WORD count, BYTE *buffer);
```

Aufruf in Assembler:

```
pea       buffer          ; Offset 8
move.w    count,-(sp)     ; Offset 6
move.w    start,-(sp)     ; Offset 4
move.w    op,-(sp)        ; Offset 2
move.w    #$2E,-(sp)      ; Offset 0
lea       $C(sp),sp
```

Parameter:

op:	0:	NVM auslesen, dabei Prüfsumme überprüfen
	1:	NVM beschreiben, Prüfsumme überprüfen und neu setzen
	2:	NVM initialieren und Prüfsumme setzen
start:	Nummer des Startbytes	
count:	Anzahl der zu übertragenden Bytes	
NVMaccess():	OK (0):	alles in Ordnung
	EBADRQ (–5):	Argumente nicht im erlaubten Bereich
	EGENRL (–12):	Prüfsumme war nicht konsistent (beim Lesen werden die Bytes trotzdem übertragen)

Offgibit (XBIOS 29)

Löscht Bits im "PORT A"-Register des GI-Chips.

Deklaration in C:

```
void Offgibit (WORD bitno);
```

Aufruf in Assembler:

```
move.w    bitno,-(sp)     ; Offset 2
move.w    #$1D,-(sp)      ; Offset 0
trap      #14
addq.l    #4,sp
```

Parameter:

bitno: Mit dieser Bitmaske wird der aktuelle Zustand des Registers logisch undiert.

Ongibit (XBIOS 30)

Setzt Bits im "PORT A"-Register des GI-Chips.

Deklaration in C:

```
void Ongibit (WORD bitno);
```

Aufruf in Assembler:

```
move.w    bitno,-(sp)     ; Offset 2
move.w    #$1E,-(sp)      ; Offset 0
trap      #14
addq.l    #4,sp
```

Parameter:

bitno: Mit dieser Maske wird der aktuelle Zustand des Registers logisch oderiert.

Physbase (XBIOS 2) – nur für ST/STE/TT-Videohardware

Liefert die Anfangsadresse des physikalischen Bildschirmspeichers (das ist genau der Speicherbereich, den man auf dem Bildschirm sieht).

Deklaration in C:

```
void *Physbase (void);
```

Aufruf in Assembler:

```
move.w    #2,-(sp)   ; Offset 0
trap      #14
addq.l    #2,sp
```

Parameter:

Physbase(): Anfangsadresse des physikalischen Bildschirmspeichers.

Protobt (XBIOS 18)

Baut im Speicher einen Bootsektor auf.

Deklaration in C:

```
void Protobt (void *buf, LONG serialno, WORD disktype,
              WORD execflag);
```

Aufruf in Assembler:

```
move.w    execflag,-(sp) ; Offset 12
move.w    disktype,-(sp) ; Offset 10
move.l    serialno,-(sp) ; Offset 6
pea       buf            ; Offset 2
move.w    #$12,-(sp)     ; Offset 0
trap      #14
lea       $E(sp),sp
```

Parameter:

buf:	Zeiger auf einen 512 Bytes langen Puffer, der den neuen Bootsektor enthalten soll	
serialno:	–1:	gleiche Seriennummer wie beim letzten Mal
	>= $01000000:	zufällige Seriennummer berechnen
	sonst:	neue Seriennummer
disktype:	–1:	Typ nicht verändern
	0:	40 Spuren, einseitig (180 K)
	1:	40 Spuren, doppelseitig (360 K; MS-DOS-Standardformat)
	2:	80 Spuren, einseitig (360 K; Standard einseitig)
	3:	80 Spuren, doppelseitig (720 K; Standard doppelseitig)
	4:	High-Density (nur wenn im "_FDC"-Cookie der Wert für HD-Unterstützung eingetragen ist)
	5:	Extra-High-Density (nur wenn im "_FDC"-Cookie der Wert für ED-Unterstützung eingetragen ist)
execflag:	–1:	nicht ändern
	0:	Bootsektor nicht ausführbar
	1:	Bootsektor ausführbar (siehe Aufbau von Bootsektoren)

Bemerkungen
Damit die Diskette auch auf MS-DOS-Systemen lesbar ist, muß man manuell die hexadezimalen Werte $E9 $00 $4E in die ersten drei Bytes des Bootsektors eintragen.

"Protobt()" neigt zu Verschwendung: auf einer normalen 720K-Diskette wird Platz für fünf FAT-Sektoren angelegt, obwohl drei völlig reichen würden.

Prtblk (XBIOS 36)

Ähnelt der normalen Hardcopy-Funktion (von der aus sie auch aufgerufen wird).

Allerdings ist sie um einiges flexibler; sie erwartet einen Parameterblock folgender Form:

```
typedef struct
{
     LONG pb_scrptr;     /* Zeiger auf Bildschirmanfang */
     WORD pb_offset;     /* dazu zu addierender Offset */
     WORD pb_width;      /* Bildschirmbreite in Punkten */
     WORD pb_height;     /* Bildschirmhöhe in Punkten */
     WORD pb_left;       /* Linker Rand in Punkten */
     WORD pb_right;      /* Rechter Rand in Punkten */
     WORD pb_screz;      /* Auflösung */
     WORD pb_prrez;      /* Druckertyp Atari/Epson */
     LONG pb_colptr;     /* Zeiger auf Farbpalette
                            (zum Beispiel $FF8240) */
     WORD pb_prtype;     /* 0: Atari Matrix monochrom,
                            1: Atari Matrix farbig,
                            2: Atari Typenrad monochrom,
                            3: Epson Matrix monochrom */
     WORD pb_prport;     /* Schnittstelle Centronics/RS 232 */
     LONG pb_mask;       /* Zeiger auf Halbtonmaske */
} PBDEF;
```

Insbesondere muß man beachten, daß die Summe von "pb_width", "pb_left" und "pb_right" genau die tatsächliche Bildschirmbreite ergeben muß.

Ferner muß man vorher "_dumpflg" ($4EE) auf 1 setzen – denn sonst passiert überhaupt nichts. Der Druckvorgang kann mit "Alternate/Help" abgebrochen werden.

Nach der Rückkehr sollte man "_dumpflg" wieder auf –1 setzen.

Deklaration in C:

```
void Prtblk (PBDEF *defptr);
```

Aufruf in Assembler:

```
move.w    #1,_dumpflg
pea       defptr          ; Offset 2
move.w    #$24,-(sp)      ; Offset 0
trap      #14
addq.l    #6,sp
move.w    #-1,_dumpflg
```

Parameter:

defptr: Zeiger auf PBDEF-Struktur

Bemerkungen
Gibt über die Systemvektoren "prv_lsto", "prv_lst", "prv_auxo" bzw. "prv_aux" aus.

Die offizielle Dokumentation ("Hitchhiker's Guide to the BIOS") führt diese Funktion zwar auf, schweigt sich allerdings über alle Parameter aus.

Puntaes (XBIOS 39)

Das AES wird nur gestartet, wenn "os_magic" im OSHEADER auf die korrekte magische Zahl ($87654321) zeigt. "Puntaes()" setzt dieses Flag (sofern möglich) zurück und bootet dann das System neu.

Deklaration in C:

```
void Puntaes (void);
```

Aufruf in Assembler:

```
move.w    #$27,-(sp)      ; Offset 0
trap      #14
addq.l    #2,sp
```

Random (XBIOS 17)

Mit dieser Funktion kann man sich eine 24-Bit-Zufallszahl berechnen lassen. Der Anfangswert nach Systeminitialisierung sollte stets unterschiedlich sein, da mit dem Wert des vertikalen Zeilenzählers (im Videochip) initialisiert wird.

Gleichwohl handelt es sich nicht um einen echten Hardware-Zufallszahlengenerator (wie zum Beispiel bei Ataris 8-Bittern). Statt dessen wird nach dem Algorithmus

S = (S * 3.1415926...) + 1

verfahren. Zurückgeliefert wird die Zahl S (um acht Bits nach rechts verschoben). Das Verhalten für die gesamte Zahl ist recht gut, die Abfrage einzelner Bits kann jedoch gefährlich sein (im Sinne einer verminderten "Zufälligkeit").

Im Zweifelsfall sollte man sich zunächst eine Reihe von Zahlenwerten ausgeben lassen, die "zufällig" sein sollen!

Deklaration in C:

```
LONG Random (void);
```

Aufruf in Assembler:

```
move.w    #$11,-(sp)      ; Offset 0
trap      #14
addq.l    #2,sp
```

Parameter:

Random(): 24-Bit-Zufallszahl

Rsconf (XBIOS 15)

Setzt Parameter für die serielle Schnittstelle. "Früher" setzte diese Funktion einfach nur die entsprechenden Register im MFP-Baustein.

Seit der Einführung des Atari TT ist nicht mehr ohne weiteres klar, welcher Hardwarebaustein für die serielle Schnittstelle eingesetzt wird (siehe XBIOS-Einführung und "Bconmap()").

Daher muß man "Rsconf()" besonders vorsichtig einsetzen. Das heißt:

1. Aktuelle Einstellungen mit Rsconf (–1, –1, –1, –1, –1, –1) abfragen.
2. Nur die interessanten Bits modifizieren.
3. Die neuen Werte setzen.

Dadurch, daß die serielle Schnittstelle nicht mehr unbedingt durch einen MFP angesteuert wird, werden die benutzbaren Werte folgendermaßen eingeschränkt:

ucr:	Bit 5..6:	Wortlänge (00: 8, 01: 7, 10: 6, 11: 5)
	Bit 3..4:	Stopbits (01: 1, 10: 1.5, 11: 2, 00: ungültig)
	Bit 2:	parity (0: nein, 1: ja)
	Bit 1:	0: odd parity, 1: even parity (nur sinnvoll, wenn auch Bit 2 gesetzt ist)
rsr:	nicht benutzbar	
tsr:	Bit 3:	1: break
scr:	nicht benutzbar	

Technisch nicht mögliche Werte müssen ignoriert werden.

Schon deshalb sollte man sich nie darauf verlassen, daß wirklich alle Einstellungen vorgenommen worden sind (sondern den Rückgabewert inspizieren!).

Deklaration in C:

```
ULONG Rsconf (WORD speed, WORD flowctl, WORD ucr, WORD rsr,
              WORD tsr, WORD scr);
```

Aufruf in Assembler:

```
move.w    scr,-(sp)       ; Offset 12
move.w    tsr,-(sp)       ; Offset 10
move.w    rsr,-(sp)       ; Offset 8
move.w    ucr,-(sp)       ; Offset 6
move.w    flowctl,-(sp)   ; Offset 4
move.w    speed,-(sp)     ; Offset 2
move.w    #$F,-(sp)       ; Offset 0
trap      #14
lea       $E(sp),sp
```

Parameter:

speed:	-1:	nicht ändern
	0:	19200 Baud
	1:	9600 Baud
	2:	4800 Baud
	3:	3600 Baud
	4:	2400 Baud
	5:	2000 Baud
	6:	1800 Baud
	7:	1200 Baud
	8:	600 Baud
	9:	300 Baud
	10:	200 Baud
	11:	150 Baud
	12:	134 Baud
	13:	110 Baud
	14:	75 Baud
	15:	50 Baud
flowctl:	-1:	nicht ändern
	0:	keine Flußkontrolle
	1:	XON/XOFF (CTRL-S, CTRL-Q)
	2:	RTS/CTS
	3:	RTS/CTS und XON/XOFF
ucr, tsr, rsr, scr:		entweder die neuen Werte für die Register oder -1 (nicht verändern)
Rsconf():		in gepackter Form die Werte des ucr-, rsr- und tsr-Registers (Bit 31..24: ucr, Bit 23..16: rsr, Bit 15..8: tsr, Bit 7..0: scr)

Bemerkungen
Ab TOS 1.04 kann man mit "Rsconf(–2,–1,–1,–1,–1)" die letzte eingestellte Baudrate abfragen.

Scrdmp (XBIOS 20)

Druckt den aktuellen Bildschirminhalt aus. Der Druckvorgang kann durch Betätigen von "Alternate/Help" abgebrochen werden (siehe auch "Setprt()").

Deklaration in C:

```
void Scrdmp (void);
```

Aufruf in Assembler:

```
move.w    #$14,-(sp)      ; Offset 0
trap      #14
addq.l    #2,sp
```

Bemerkungen
Springt über den Systemvektor "scr_dump". Die dort eingetragene Funktion ruft normalerweise die XBIOS-Funktion "Prtblk()" auf.

Die Hardcopy-Routine funktioniert nur für eine ausgeschränkte Auswahl von Druckern (siehe "Setprt()") und beim TT auch nicht in allen Auflösungen.

Setcolor (XBIOS 7) – nur für ST/STE/TT-Videohardware

Setzt ein Farbregister auf einen neuen Wert oder fragt nur den Wert ab.

Deklaration in C:

```
WORD Setcolor (WORD colorNum, WORD color);
```

Aufruf in Assembler:

```
move.w    color,-(sp)     ; Offset 4
move.w    colorNum,-(sp)  ; Offset 2
move.w    #7,-(sp)        ; Offset 0
trap      #14
addq.l    #6,sp
```

Parameter:

color:	Neuer Farbwert oder –1, falls das Register nicht geändert werden soll
colorNum:	Nummer des anzusprechenden Farbregisters (0..15)
Setcolor():	Bisheriger Wert des Farbregisters

Setpalette (XBIOS 6) – nur für ST/STE/TT-Videohardware

Neue Farbpalette an die Video-Hardware übergeben (geschieht erst im nächsten Vertical Blank).

Deklaration in C:

```
void Setpalette (WORD *palettePtr);
```

Aufruf in Assembler:

```
pea       palettePtr     ; Offset 2
move.w    #6,-(sp)       ; Offset 0
trap      #14
addq.l    #6,sp
```

Parameter:

palettePtr: Zeiger auf eine Tabelle aus sechzehn 16-Bit-Worten, die die neue Farbpalette enthalten (wobei jeweils die untersten zwölf Bits für die RGB-Werte benutzt werden). Muß auf einer geraden Adresse liegen.

Bemerkungen

Das hier benutzte Format wird bei künftigen Auflösungen (z.B. 320 * 480 in 256 Farben beim Atari TT) nicht mehr ausreichen! Statt dessen verwende man besser die entsprechenden VDI-Funktionen!

"Setpalette" ist ein "*delayed action call*". Zu deutsch: die Daten werden nicht sofort, sondern erst etwas später verarbeitet. Da nur ein Zeiger auf die Farbtabelle übergeben wird, muß man schon selbst dafür sorgen, daß der Zeiger im nächsten Vertical Blank noch auf etwas Sinnvolles zeigt.

Setprt (XBIOS 33)

Setzt oder liest die aktuelle Druckereinstellung. Diese Einstellung wird von fast allen anderen Betriebssystemteilen mit Nichtachtung gestraft (Ausnahmen: "Scrdmp()" und "Prtblk()"). Beim BIOS liegt das an einem Fehler, der leider in allen bekannten TOS-Versionen gleichermaßen vorhanden ist.

In eigenen Programmen sollte man zumindest die Wahl der Schnittstelle (seriell/parallel) und die Einzelblatt-/Endlospapiereinstellung auswerten.

Deklaration in C:

```
WORD Setprt (WORD config);
```

Aufruf in Assembler:

```
move.w    config,-(sp)    ; Offset 2
move.w    #$21,-(sp)      ; Offset 0
trap      #14
addq.l    #4,sp
```

Parameter:

config:	–1: Konfiguration auslesen	
	neues Konfigurationsbyte:	
	Bitbelegung:	
	Bit 0: Matrixdrucker	Typenraddrucker
	Bit 1: monochrom	farbig
	Bit 2: Atari-Drucker (1280 Punkte)	Epson-Drucker (960 Punkte)
	Bit 3: normale Qualität	hohe Qualität (nur bei 1280 P.)
	Bit 4: Centronics	RS 232
	Bit 5: Endlospapier	Einzelblattpapier
	alle weiteren Bits sind zur Zeit nicht benutzt	
Setprt():	aktuelle Konfiguration	

Bemerkungen

Bit 2 ist eigentlich nur für die XBIOS-eigene Hardcopy-Routine interessant.

Setscreen (XBIOS 5) – nur für ST/STE/TT-Videohardware

Dient zum Ändern von Bildschirmspeicheradressen und Auflösung.

Deklaration in C:

```
void Setscreen (void *logLoc, void *physLoc, WORD res);
```

Aufruf in Assembler:

```
move.w    res,-(sp)       ; Offset 10
move.l    physLoc,-(sp)   ; Offset 6
move.l    logLoc,-(sp)    ; Offset 2
move.w    #5,-(sp)        ; Offset 0
trap      #14
lea       $C(sp),sp
```

Parameter:

logLoc:	–1:	Adresse des logischen Bildschirmspeichers nicht ändern
	sonst:	neue Anfangsadresse
physLoc:	–1:	Adresse des physikalischen Bildschirmspeichers nicht ändern
	sonst:	neue Anfangsadresse (muß beim ST auf einer 256-Byte-Grenze liegen)
res:	–1:	Auflösung nicht ändern
	sonst:	neue Bildauflösung

Bemerkungen

Bei Änderungen der Bildauflösung wird der VT52-Emulator automatisch initialisiert (z.B. der Cursor in die linke obere Ecke gesetzt).

Bitte anschließend mit "Physbase()", "Logbase()" und "Getrez()" überprüfen, ob der Funktionsaufruf erfolgreich war: niemand garantiert, daß man mit jedem Bildschirmtreiber und jeder Grafikkarte all diese Parameter verändern kann!

Vorsicht, wenn gleichzeitig die Systemvariable "screenpt" benutzt wird!

Settime (XBIOS 22)

Setzt die Hardware-Uhr auf ein neues Datum. Bei den Mega-STs wird nicht die IKBD-Uhr, sondern die batteriegepufferte Uhr gesetzt.

Deklaration in C:

```
void Settime (LONG datetime);
```

Aufruf in Assembler:

```
move.l    datetime,-(sp) ; Offset 2
move.w    #$16,-(sp)     ; Offset 0
trap      #14
addq.l    #6,sp
```

Parameter:

datetime: Codiertes Datum:

Bit 0..4:	Sekunden (Wert 0..59 durch 2 dividieren)
Bit 5..10:	Minuten
Bit 11..15:	Stunden
Bit 16..20:	Tag
Bit 21..24:	Monat
Bit 25..31:	Jahr (1980 subtrahieren)

Ssbrk (XBIOS 1)

Reserviert Speicherplatz am Ende des physikalischen Speicherbereichs und kann nur vor der Initialisierung von GEMDOS benutzt werden (diese Funktion ist bei sämtlichen ROM-TOS-Versionen nur eine Dummy-Routine, die überhaupt nichts bewirkt!).

Deklaration in C:

```
LONG Ssbrk (WORD amount);
```

Aufruf in Assembler:

```
move.w    amount,-(sp)    ; Offset 2
move.w    #1,-(sp)        ; Offset 0
trap      #14
addq.l    #6,sp
```

Parameter:

amount:	Anzahl der zu reservierenden Bytes
Ssbrk():	Anfangsadresse des allozierten Speicherbereichs

Supexec (XBIOS 38)

Führt ein (durch RTS abgeschlossenes) Unterprogramm im Supervisor-Modus aus.

Deklaration in C:

```
LONG Supexec (LONG (*codeptr)());
```

Aufruf in Assembler:

```
pea       codeptr        ; Offset 2
move.w    #$26,-(sp)     ; Offset 0
trap      #14
addq.l    #6,sp
```

Parameter:

codeptr: Anfangsadresse der betreffenden Routine
Supexec(): Der Return-Wert der aufgerufenen Funktion wird von "Supexec()" an den Aufrufer zurückgereicht.

Vsync (XBIOS 37)

Wartet, bis ein Vertical-Blank-Interrupt aufgetreten ist. Für zeitkritische Abfragen (beispielsweise für Fine-Scrolling) sollte man besser direkt den Zeilenzähler im Shifter abfragen!

Deklaration in C:

```
void Vsync (void);
```

Aufruf in Assembler:

```
move.w    #$25,-(sp)     ; Offset 0
trap      #14
addq.l    #2,sp
```

Xbtimer (XBIOS 31)

Setzt und startet die MFP 68901-Timer.

Deklaration in C:

```
void Xbtimer (WORD timer, WORD control, WORD data, void (*vec)());
```

Aufruf in Assembler:

```
pea       vec             ; Offset 8
move.w    data,-(sp)      ; Offset 6
move.w    control,-(sp)   ; Offset 4
move.w    timer,-(sp)     ; Offset 2
move.w    #$1F,-(sp)      ; Offset 0
trap      #14
lea       $C(sp),sp
```

Parameter:

timer: Nummer des Timers:

0:	Timer A:	kann für eigene Programme benutzt werden
1:	Timer B:	Horizontal-Blank-Interrupt
2:	Timer C:	200 Hz Systemtimer
3:	Timer D:	Baudraten-Generator

control, data: Werte für die entsprechenden Timer-Register

vec: Zeiger auf die dazugehörige Interrupt-Routine

Kapitel 2: GEMDOS

Einleitung

Das erste System, auf dem GEM implementiert wurde, war bekanntlich der IBM PC unter MS-DOS. Als Ende 1984 GEM auf den Motorola-Prozessor portiert wurde, gab es weder fertige STs noch Entwicklungswerkzeuge noch ein Betriebssystem, auf das man hätte aufbauen können.

So nahmen die Entwickler den Umweg über eine Apple "Lisa", auf der CP/M 68K (ebenfalls ein DOS von Digital Research) lief. Wer das damalige Entwicklungspaket von Atari kennt, erinnert sich sicherlich noch an den Alcyon-C-Compiler oder "LINK68" und "RELMOD" (das CP/M-68K-Programme in GEMDOS-Programme konvertierte). Aus verschiedenen Gründen kam CP/M 68K nicht als endgültiges Betriebssystem in Frage (ein Grund war das fehlende hierarchische Dateisystem). Daher entschloß man sich bei Atari und Digital Research, als Grundlage für GEM auf Motorola-Rechnern einen völlig neuen Unterbau zu entwickeln, eben "GEMDOS" ("GEM Disk Operating System"). Und so entwickelte Digital-Research-Programmierer Jason Loveman in kürzester Zeit ein neues DOS, das in den meisten Details MS-DOS 2.0 entspricht. Leider führte die Eile auch zu einer Fülle von Fehlern und Designschwächen, von denen Atari-Programmierer Allan Pratt bis heute leider nur einige beseitigen konnte. Problemkinder waren und bleiben die Speicherverwaltung und die Ein-/Ausgabeumlenkung.

Aber auch MS-DOS (zumindest die Version 1.0) war nicht mehr als eine Weiterentwicklung vom CP/M für den IBM PC und dazu kompatible Rechner. Und so stimmen noch heute viele GEMDOS-Funktionsnummern mit denen der CP/M-Versionen für Z80-Systeme aus dem Jahr 1975 überein. *Also*: obwohl es in vielen Belangen so aussieht wie MS-DOS, ist GEMDOS ein eigenes Betriebssystem mit vielen Tücken im Detail. Oder, um es mit den Worten von Tim Oren zu sagen: "Remember: it looks like a duck, and quacks like a duck, but it's not a duck!" (Professional GEM, Folge 15, "Coping with GEMDOS").

Dateisystem

Dateibezeichnungen

Eine Dateibezeichnung besteht aus einer optionalen Laufwerksangabe, einer optionalen Pfadangabe und dem eigentlichen Dateinamen. Der *Dateiname* besteht aus einem bis acht Zeichen, gegebenenfalls gefolgt von einem Punkt und drei weiteren Zeichen.

Erlaubte Zeichen sind:

```
A bis Z, a bis z, 0 bis 9, _!@#$%^&()+-=~`;"",<>|[]{}
```

Kleine Buchstaben werden automatisch in Großbuchstaben umgewandelt. Alle anderen Zeichen – insbesondere Umlaute – sind nicht erlaubt (auch wenn das AES sie in Datei-Eingabemasken erlaubt).

Der *Pfadname* besteht aus beliebig vielen Dateinamen, jeweils gefolgt von einem Backslash ("\"). Wird am Anfang des Dateinamens ein Backslash eingefügt, beginnt der Pfad im Wurzelverzeichnis des Laufwerks, ansonsten wird vom aktuellen Verzeichnis des Laufwerks ausgegangen. Fehlt der Pfadname ganz, wird statt dessen das aktuelle Verzeichnis des Laufwerks benutzt.

In Pfadnamen sind zwei besondere Dateinamen erlaubt:

"." bezeichnet das *aktuelle* Verzeichnis

".." bezeichnet das eine Ebene über dem aktuellen Verzeichnis liegende Verzeichnis

Ein Pfad heißt *relativ*, wenn er nicht mit dem Wurzelverzeichnis beginnt. Anderenfalls spricht man von einem *absoluten* Pfad.

Vorsicht: Bei Anwendung zu "komplizierter" Pfadbezeichnungen (wie zum Beispiel: "`..\..\.\...\sybex\.\pbuch\`") scheinen die GEMDOS-internen Routinen nicht immer korrekt zu funktionieren. Im Zweifelsfall sollte man eine eigene Routinen vorschalten, die relative in absolute Pfadnamen konvertieren.

Die *Laufwerksbezeichnung* besteht aus einem einzelnen Buchstaben zwischen "A" und "Z", gefolgt von einem Doppelpunkt. Fehlt die Laufwerksbezeichnung, wird das aktuelle Laufwerk gewählt.

GEMDOS selbst unterstützt nur 16 verschiedene logische Laufwerke (also "A" bis "P"). Durch Installation von Meta-DOS wird die Zahl der Laufwerke auf 26 (also bis "Z") erweitert. Welche Laufwerke genau verfügbar sind, stellt man am besten mit folgender Konstruktion fest:

```
Dsetdrv (Dgetdrv());
```

Der *Gesamtlänge* der Dateibezeichnung setzt GEMDOS selbst keine Grenze. Allerdings hat das Desktop eine Begrenzung auf acht Ordnertiefen. Die allermeisten Programme gehen

davon aus, daß 128 Zeichen Länge ausreichen. Zusätzlich gibt es spezielle Dateibezeichnungen für die zeichenorientierten Geräte, auf die man beispielsweise mittels "Fopen()" zugreifen kann:

"CON:", "con:" Bildschirm bzw. Tastatur

"AUX:", "aux:" Serielle Schnittstelle

"PRN:", "prn:" Parallele Schnittstelle

Das Desktop unterstützt zusätzlich das Laufwerk "c:" (kleingeschrieben) für ROM-Module. Dabei handelt es sich allerdings um kein echtes Dateisystem. Weitere Informationen dazu finden Sie bei der Beschreibung der Cartridge-Hardware.

Schlußbemerkung: Man sollte sich – wann immer möglich – nicht auf das genaue Format von Dateinamen verlassen und insgesamt keine Annahmen über

- das Vorhandensein nicht erlaubter Zeichen,
- die Unterscheidung zwischen Groß- und Kleinschrift oder
- die maximale Länge von Dateinamen

machen. Solche Vorsichtsmaßnahmen erleichtern Portierungen und Anpassungen an neue GEMDOS-Versionen wie die Multitasking-Erweiterung "MiNT", bei der zur Zeit (Testversion 0.8) Dateinamen von bis zu 14 Zeichen praktisch beliebiger Art möglich sind (das Dateinamensfeld in der DTA ist dort *nicht zwingend* Null-terminiert).

Aktueller Pfad, aktuelles Laufwerk

GEMDOS merkt sich zu jedem Laufwerk einen aktuellen Pfad, auf den sich alle Pfadangaben stützen, die nicht mit einem "\" beginnen. Außerdem gibt es auch ein aktuelles Laufwerk, das immer dann benutzt wird, wenn eine Laufwerksangabe fehlt. Zum Setzen und Abfragen gibt es "Dgetpath()", "Dsetpath()", "Dgetdrv()" und "Dsetdrv()". Die aktuellen Werte werden an mittels "Pexec()" gestartete Programme vererbt und dabei gemerkt.

GEMDOS-Dateisystem

Das GEMDOS-Dateisystem entspricht weitgehend dem in MS-DOS 2.0 benutzten Dateisystem. Wenn man beim Formatieren der Disketten einige Punkte beachtet (siehe unter "Protobt()"),

kann man die Disketten sogar mit einem MS-DOS-Computer austauschen – selbst wenn man ein High-Density-Laufwerk benutzt.

Intern existiert eine klare Arbeitsteilung zwischen BIOS und GEMDOS. Das BIOS stellt Routinen zum Lesen und Schreiben ("Rwabs()"), zum Erkennen eines Medienwechsels ("Mediach()") und zur Abfrage von Dateisysteminformationen ("Getbpb()") zu Verfügung.

An dieser Stelle sollte man noch erwähnen, daß man sich keineswegs darauf verlassen darf, daß GEMDOS tatsächlich das hier geschilderte Dateisystem benutzt.

Ein Beispiel: Wenn Meta-DOS eingesetzt wird, kann man noch nicht einmal davon ausgehen, daß BIOS-Aufrufe zum Lesen und Schreiben benutzt werden.

GEMDOS benutzt die ersten Sektoren eines Laufwerks für Verwaltungsinformationen. Ein Laufwerk besteht typischerweise aus fünf Teilen: dem optionalen Bootsektor, zwei FATs ("File Allocation Table"), einem Wurzelverzeichnis und dem Clusterbereich (GEMDOS verwaltet die Sektoren in Gruppen mehrerer Sektoren, den Clustern).

Die FAT beschreibt, welche Cluster belegt sind, und stellt die Verkettung der einzelnen Cluster innerhalb einer Datei her. Alle Dateien und Unterverzeichnisse werden im Clusterbereich abgelegt und über die FAT verwaltet.

Nur das Wurzelverzeichnis stellt in dieser Hinsicht eine Ausnahme dar.

Beim ersten Zugriff auf ein Laufwerk ruft GEMDOS die Funktion "Getbpb()" auf. Zurückgeliefert wird eine BPB-Struktur, aus der alle für GEMDOS interessanten Informationen extrahiert werden. Dasselbe passiert übrigens auch nach einem Medienwechsel.

```
typedef struct
{
     WORD  recsiz;    /* Bytes pro Sektor */
     WORD  clsiz;     /* Sektoren pro Cluster (Einheit) */
     WORD  clsizb;    /* Bytes pro Cluster */
     WORD  rdlen;     /* Länge des Wurzelverzeichnis in Sektoren */
     WORD  fsiz;      /* Länge des File Allocation Table (FAT) */
     WORD  fatrec;    /* Startsektor der zweiten FAT */
     WORD  datrec;    /* Sektornummer des ersten freien Clusters */
     WORD  numcl;     /* Gesamtzahl der Cluster auf dem Medium */
     WORD  bflags;    /* Bitvektor, zur Zeit nur Bit 0 belegt:
                         0 (12-Bit-FAT), 1 (16-Bit-FAT) */
} BPB;
```

"recsiz" gibt an, wie groß jeder einzelne Sektor auf dem Medium ist. Mögliche Größen sind 512, 1024, 2048 und 4096. Bei noch größeren Sektorgrößen kommt es in allen bekannten GEMDOS-Versionen zu Problemen mit zu großen Clustergrößen.

"clsiz" muß laut Atari immer 2 sein. Andere Werte wären zum Lesen mancher MS-DOS-Formate wünschenswert, sind aber zur Zeit (GEMDOS 0.19) angeblich nicht möglich (in der Praxis scheinen 1-Sektor-Cluster – benötigt für MS-DOS-High-Density-Disketten – zu funktionieren).

"clsizb" ist die Größe eines Clusters in Bytes und somit eigentlich eine überflüssige Information (da aus "recsiz" und "clsiz" leicht zu errechnen).

"rdlen" beschreibt die Länge des Wurzelverzeichnis in Sektoren. Jeder Verzeichniseintrag belegt 32 Bytes, somit passen "(rdlen * recsiz) / 32" Einträge in das Verzeichnis.

Damit ist das Wurzelverzeichnis im Gegensatz zu Unterverzeichnissen in seiner Aufnahmefähigkeit durch einen fixen Wert beschränkt (bei "normalen" Disketten liegt diese Zahl üblicherweise bei 112 Einträgen).

"fsiz" beschreibt die Länge jeder einzelnen FAT in Sektoren. "fatrec" gibt die Nummer des Anfangssektors der *zweiten* FAT an. In "datrec" findet man die Sektornummer des ersten Clusters. "numcl" legt fest, wieviel Cluster auf dem Medium Platz haben.

"bflags" enthält zusätzliche Informationen über das Dateisystem. Zur Zeit ist nur das unterste Bit belegt: wenn es gesetzt ist, hat man es mit 16 Bit großen FAT-Einträgen zu tun (sonst 12 Bit). Neuere MS-DOS-Versionen erlauben auch 32-Bit-FATS (dort ist die BPB-Struktur auch im Hinblick auf das "High Performance File System" von OS/2 um zusätzliche Felder erweitert worden).

Aus diesen Informationen kann man noch einige weitere Werte berechnen:

Wenn es Bootsektoren gibt, dann belegen sie die Sektoren 0 bis (fatrec – fsiz – 1). Die erste FAT beginnt bei (fatrec – fsiz), und das Wurzelverzeichnis startet bei (fatrec + fsiz).

Verzeichniseinträge

Ein Verzeichniseintrag kann entweder innerhalb des Wurzelverzeichnisses oder in einem Unterverzeichnis auftreten (ein Unterverzeichnis kann als Datei mit Verzeichniseinträgen angesehen werden).

Jeder Eintrag ist insgesamt 32 Bytes lang und hat das folgende Format:

```
typedef struct
{
     char     dir_name[11];  /* Name inkl. Erweiterung ohne Punkt */
     BYTE     dir_attr;      /* Attribut */
     BYTE     dir_dummy[10]; /* unbenutzt */
     UWORD    dir_time;      /* Erstellungszeit */
     UWORD    dir_date;      /* Erstellungsdatum */
     UWORD    dir_stcl;      /* erster Cluster */
     ULONG    dir_flen;      /* Länge der Datei */
} DIR;
```

"dir_time", "dir_date", "dir_stcl" und "dir_flen" sind im Intel-Format, das heißt höher- und niederwertige Bytes sind vertauscht. Bei gelöschten Dateien ist das erste Zeichen des Dateinamens mit dem Wert $E5 überschrieben.

FAT-Einträge

In der FAT finden sich Informationen über jeden einzelnen Cluster des Speichermediums. Dabei sind folgende Fälle möglich:

- Der Cluster ist frei.
- Der Cluster ist belegt, und der FAT-Eintrag enthält einen Verweis auf den nächsten Cluster der Datei.
- Der Cluster ist belegt und ist der letzte von der Datei belegte Cluster.
- Der Cluster ist beschädigt und für weitere Benutzung gesperrt. Viele Programme zum Formatieren von Disketten bzw. Partitionieren von Festplatten benutzen diese Möglichkeit, um als defekt erkannte Sektoren für eine Benutzung zu sperren.

FAT-Einträge können – je nach Größe des Mediums – jeweils 12 oder 16 Bit groß sein. Unter neueren MS-DOS-Versionen werden auch 32-Bit-FATs benutzt, für GEMDOS sind solche Formate allerdings nicht ansprechbar.

Die ersten beiden Einträge der FAT sind unbenutzt, die eigentliche Zählung beginnt also erst bei 2. Dies zieht ein Mißverständnis in der Kommunikation zwischen BIOS und GEMDOS nach sich, welches dazu führt, daß die beiden letzten Cluster eines Mediums nicht benutzt werden können. GEMDOS-Dateisysteme werden von MS-DOS allerdings nur dann korrekt erkannt, wenn sich im ersten Byte jeder FAT eine Kopie des "Mediabytes" (siehe BIOS-Einführung) befindet.

Die möglichen Werte für die Einträge:

12-Bit-FAT	16-Bit-FAT	Bedeutung
$000	$0000	Freier Cluster
$001	$0001	nicht erlaubt
$002 – $FEF	$0002 – $7FFF	Nummer des nächsten Clusters
	$8000 – $FFEF	nicht erlaubt
$FF0 – $FF7	$FFF0 – $FFF7	Unbrauchbarer (defekter) Cluster
$FF8 – $FFF	$FFF8 – $FFFF	Letzter Sektor (Dateieende)

Curiosity killed the FAT...

Der Satz sagt es schon: Manipulationen am Dateisystem sind eine höchstgradig gefährliche Angelegenheit. Allen Programmen, die direkt das Dateisystem manipulieren (zum Beispiel Harddisk-"Optimierer") sollte man mit einer gehörigen Portion Mißtrauen begegnen!

Dateiattribute

Die verschiedenen Eigenschaften einer Datei werden über das Attribut-Byte im Verzeichniseintrag festgelegt. Dabei sind folgende Bits definiert (alle weiteren sind für künftige GEMDOS-Versionen reserviert):

Bit	Bitmaske	Bezeichnung	Erklärung
0	0x01	FA_READONLY	Datei ist gegen Überschreiben geschützt
1	0x02	FA_HIDDEN	Versteckte Datei
2	0x04	FA_SYSTEM	Systemdatei
3	0x08	FA_VOLUME	Laufwerksname (Label)
4	0x10	FA_SUBDIR	Unterverzeichnis
5	0x20	FA_ARCHIVE	Archiv-Bit

Schreibgeschützte Dateien können weder gelöscht werden, noch ist es möglich, schreibend auf sie zuzugreifen. Versteckte Dateien erscheinen normalerweise nicht im Verzeichnis (die GEM-Dateiauswahlbox berücksichtigt dies, das Desktop nicht). Das Attribut "Systemdatei" hat seinen Ursprung bei MS-DOS und kann am besten als "darf nicht verschoben werden" gedeutet werden. Einen Laufwerksnamen darf es auf jedem Laufwerk nur einmal geben. Weitere Attribut-Bits dürfen nicht gesetzt sein (eine Routine zum Setzen des Laufwerksnamens ist unter "Fcreate()" angegeben).

FA_VOLUME und FA_SUBDIR dürfen niemals in Verbindung mit anderen Attributen benutzt werden: in solchen Fällen ist das Verhalten von GEMDOS *undefiniert* (zu deutsch: es kann funktionieren, aber auch zur Detonation des Monitors führen).

Das Archiv-Bit wird erst ab GEMDOS 0.15 (Rainbow-TOS) korrekt benutzt: es wird gesetzt, wenn eine Datei neu ist oder modifiziert worden ist (allerdings nicht, wenn dabei ihre Länge unverändert geblieben ist). Backupprogramme können dieses Bit benutzen, um die veränderten Dateien eines Laufwerks zu finden und zu kopieren. Anschließend sollte es dann mit "Fattrib()" zurückgesetzt werden. Ältere GEMDOS-Versionen haben das Archiv-Bit einfach ignoriert. Einige Backup-Programme haben dies ausgenutzt, indem sie das Bit bei *archivierten* Dateien manuell gesetzt haben.

Medienwechsel (Mediachange)

Wenn eine Diskette oder eine Wechselplatte (oder eine CD...) gewechselt wird, muß GEMDOS natürlich darüber informiert werden. Insbesondere müssen dabei alle dieses Laufwerk betreffenden Informationen (Dateien, aktuelle Verzeichnisse, vom GEMDOS gepufferte Sektoren) wieder zurückgesetzt werden.

Intern läuft dabei ein kompliziertes Protokoll zwischen BIOS und GEMDOS ab. Hier ein möglicher Verlauf eines Diskettenwechsels:

1. Der Benutzer entnimmt die Diskette und legt eine andere ein.

2. GEMDOS versucht, mittels "Rwabs()" einen Sektor zu lesen, das BIOS liefert den Return-Wert E_CHNG (-14) zurück.

3. GEMDOS weiß nun, daß das Medium gewechselt worden ist (oder gewechselt worden sein könnte). Deshalb setzt es alle internen Informationen über das Laufwerk zurück und ruft "Getbpb()" auf, um sich über den Aufbau des neuen Mediums zu informieren.

   Gleichzeitig wird damit dafür gesorgt, daß das BIOS den Medienwechsel als erkannt ansieht und ihn beim nächsten Zugriff nicht mehr meldet.

In den "Release Notes" zum TOS 1.04 hat Atari eine Beispielroutine abgedruckt, die einen solchen Medienwechsel *erzwingt*. Dies ist für zweierlei Arten von Programmen interessant:

1. Programme, die unter Umgehung von GEMDOS direkt auf das Dateisystem zugreifen und dort Daten verändern (Festplattenpartitionierer, Formatierprogramme, Hilfsprogramme zum Reparieren der Dateisystemstruktur).

2. Programme, die aus irgendwelchen Gründen dazu *gezwungen* sind, direkt "Getbpb()" aufzurufen: ansonsten könnte es passieren, daß ein etwaiger Medienwechsel nicht an GEMDOS gemeldet wird und dadurch Informationen auf dem Medium zerstört werden (beispielsweise dadurch, daß noch Sektoren vom "vorherigen" Medium gepuffert sind).

Ein Nachteil sollte allerdings nicht verschwiegen werden: Durch den Medienwechsel gehen alle Informationen über das betreffende Laufwerk verloren; dazu gehören unter anderem die aktuellen Verzeichnisse und offene Dateien.

Daher sollte man diese Möglichkeit nur dann benutzen, wenn es wirklich nicht anders geht und die Effekte für den Anwender verständlich sind: wie zum Beispiel im Desktop beim Drücken der ESC-Taste oder innerhalb eines Festplattenpartitionierers nach dem Löschen einer gesamten Partition.

Nun zu der Beispielroutine: Sie basiert auf dem Verbiegen der entsprechenden BIOS-Funktionen auf eigene Routinen, die einen Medienwechsel vorspiegeln. Anschließend wird eine Datei auf dem betreffenden Laufwerk geöffnet und in der Getbpb-Routine dafür gesorgt, daß alle verbogenen Vektoren wieder zurückgesetzt werden (dazu wird das im Anhang beschriebene XBRA-Verfahren benutzt).

```
; Erzwingen eines Media-Change
; basiert auf der in den "Rainbow TOS Release Notes" vom
; 7. 8. 89 angegebenen Routine

; Binding für TC: extern int cdecl mediach (int drive);

; liefert 1 zurück, wenn GEMDOS nicht Getbpb aufgerufen
; hat (Fehler!), sonst 0 (OK)
     .globl  mediach

mediach:
     move.w  4(sp),d0          ; Laufwerksnummer
     movem.l d1-a6,-(sp)       ; Register retten

     move.w  d0,mydev
     add.b   #"A",d0
     move.b  d0,fspec          ; in Dateinamen eintragen

loop:
     clr.l   -(sp)             ; Supervisor-Modus
     move.w  #$20,-(sp)
```

```
        trap    #1
        addq    #6,sp
        move.l  d0,-(sp)
        move.w  #$20,-(sp)      ; 2. Aufruf vorbereiten

        ; alte Vektoren retten

        move.l  $472,oldgetbpb
        move.l  $47e,oldmediach
        move.l  $476,oldrwabs

        move.l  #newgetbpb,$472
        move.l  #newmediach,$47e
        move.l  #newrwabs,$476

        ; Datei auf dem Laufwerk öffnen

        clr.w   -(sp)
        move.l  #fspec,-(sp)
        move.w  #$3d,-(sp)
        trap    #1
        addq    #8,sp

        ; Datei wieder schließen

        tst.l   d0
        bmi     noclose
        move.w  d0,-(sp)
        move.w  #$3e,-(sp)
        trap    #1
        addq    #4,sp

noclose:
        moveq   #0,d7       ; Return-Wert OK
        cmp.l   #newgetbpb,$472
        bne     done        ; Routine verschwunden?

        moveq   #1,d7       ; sonst ERROR
        move.l  oldgetbpb,$472
        move.l  oldmediach,$47e
        move.l  oldrwabs,$476
```

```
done:
     trap    #1          ; User-Modus
     addq    #6,sp

     move.l  d7,d0
     movem.l (sp)+,d1-a6
     rts

; newgetbpb: wenn es das richtige Laufwerk ist: Vektoren
; wieder aushängen. Auf jeden Fall aber die bisherige
; Routine aufrufen

     .dc.b   "XBRAMDCH"
oldgetbpb:
     .dc.l   1

newgetbpb:
     move.w  mydev,d0    ; richtiges Laufwerk?
     cmp.w   4(sp),d0
     bne     dooldg

     ; Vektoren entfernen
     move.l  oldgetbpb,$472
     move.l  oldmediach,$47e
     move.l  oldrwabs,$476

dooldg:
     move.l  oldgetbpb,a0
     jmp     (a0)

; newmediach: wenn es das richtige Laufwerk ist: 2
; zurückliefern, ansonsten die bisherige Routine
; aufrufen

     .dc.b   "XBRAMDCH"
oldmediach:
     .dc.l   1

newmediach:
     move.w  mydev,d0    ; richtiges Laufwerk?
     cmp.w   4(sp),d0
```

```
     bne     dooldm
     moveq.l #2,d0       ; bestimmt gewechselt!
     rts

dooldm:
     move.l  oldmediach,a0
     jmp     (a0)

; newrwabs: E_CHNG (-14) zurückliefern, wenn es
; das richtige Laufwerk ist

     .dc.b   "XBRAMDCH"
oldrwabs:
     .dc.l   1

newrwabs:
     move.w  mydev,d0
     cmp.w   14(sp),d0   ; Laufwerksnummer?
     bne     dooldr
     moveq.l #-14,d0
     rts

dooldr:
     move.l  oldrwabs,a0
     jmp     (a0)

     .data
fspec:
     .dc.b    "X:\X",0

     .bss

mydev:
     .ds.w   1
     .end
```

GEMDOS-Puffer

GEMDOS verfügt über eine einfache Pufferverwaltung für einmal gelesene Sektoren. In GEMDOS-Version 0.13 (also vor TOS 1.04) wurden diese Puffer allerdings nur sehr ineffektiv

benutzt (zum Beispiel wurden die Pufferinhalte schon beim Schließen einer Datei "vergessen"). Intern werden die Puffer in zwei Listen einfach verzeigerter Pufferkontrollblöcke verwaltet. Jeder dieser BCBs ("Buffer-Control-Block") sieht folgendermaßen aus:

```
typedef struct _bcb
{
     struct _bcb    *b_link;  /* Zeiger auf den nächsten BCB */
     WORD           b_neg1;   /* auf -1 initialisieren */
     WORD           b_private[5];
     void           *b_buf;   /* Zeiger auf eigentlichen Puffer */
} BCB;
```

GEMDOS verwaltet zwei getrennte Listen: eine für die FAT und das Wurzelverzeichnis, die zweite für den Rest der Sektoren (also exakt diejenigen, die mit Hilfe der ersten Liste verwaltet werden).

Normalerweise wird bei der Systeminitialisierung nur für je zwei Sektoren Platz angelegt. Eine Erweiterung der Puffer führt daher zu einer deutlichen Beschleunigung der GEMDOS-Dateifunktionen.

Seit AHDI/HDX 3.0 dürfen Sektoren auch größer als 512 Bytes werden (läßt sich mit der BIOS-Funktion "Getbpb()" abfragen). Beim Hinzufügen von Puffern muß man also wissen, wie groß ein Sektor maximal werden kann. Da es auch wechselbare Medien (Disketten oder Wechselplatten) gibt, ist diese Frage gar nicht so leicht zu beantworten. AHDI 3.0 stellt diese Information in der "pun_ptr"-Struktur (siehe Systemvariablen) zur Verfügung, und jeder Festplattentreiber, der dazu kompatibel sein will, sollte tunlichst genauso verfahren.

Auch wenn es schon genug andere Gründe gibt, den französischen GEMDOS-Ersatz "TurboDos" (GEMDOS 0.14) nicht zu benutzen: "TurboDos" ist *nicht* dazu in der Lage, Sektoren von mehr als 512 Bytes Größe zu verwalten.

Die einfachste Methode zur Erweiterung der Pufferliste ist, das von Atari dafür vorgesehene Programm "CACHENNN.PRG" zu benutzen. Man findet es auf den Systemdisketten zu Atari-Festplatten oder auch (hoffentlich) beim Atari-Fachhändler.

Viele andere Festplattentreiber haben diese Funktion bereits fest eingebaut – so zum Beispiel HUSHI/SCSI-Tool von den Autoren dieses Buchs.

Um selbst neue Puffer hinzuzufügen, muß man also erst einmal die maximal benötigte Sektorgröße kennen. Dazu konsultiert man die vom Festplattentreiber installierte PUN_INFO-Struktur:

```
typedef struct
{
     WORD      puns;
     BYTE      pun[16];
     LONG      part_start[16];
     LONG      P_cookie;
     LONG      *P_cookptr;
     UWORD     P_version;
     UWORD     P_max_sector;
     LONG      reserved[16];
} PUN_INFO;

PUN_INFO *getpun (void)
{
     PUN_INFO *P;
     LONG oldstack;
     oldstack = Super (0L);
     P = *((PUN_INFO **)(0x516L));
     Super ((void *)oldstack);

     if (P)  /* überhaupt gesetzt? */
          if (P->P_cookie == 0x41484449L) /* Cookie gesetzt? */
               if (P->P_cookptr == &(P->P_cookie))
                    if (P->P_version >= 0x300)
                         return P;
     return 0L;
}

WORD getsize (void)
{
     PUN_INFO *P = getpun ();

     if (P)
          return P->P_max_sector;
     else
          return 512;
}
```

Wie installiert man nun einen neuen Puffer?

1. Genügend Speicher für einen Sektor (also "getsize()" benutzen) allozieren.

2. Speicher für einen BCB allozieren.

3. Das "b_neg1"-Feld des BCBs auf –1 setzen.

4. Das "b_bufr"-Feld des BCBs mit einem Zeiger auf den Sektorpuffer füllen.

5. Je nachdem, an welche Liste der Puffer angefügt werden soll: das "b_link"-Feld auf den bisherigen Inhalt von $4B2 bzw. $4B6 setzen und die Adresse des "neuen" BCBs in $4B2 bzw. $4B6 eintragen.

Das Ganze wird für jeden weiteren Puffer wiederholt.

Das Programm darf natürlich nicht normal terminieren, sondern muß mit "Ptermres()" resident installiert werden.

Kanäle

Dateikennungen

Alle Dateien und Geräte werden über sogenannte Handles (Kennungen) angesprochen, bei denen es sich um 16-Bit-Zahlen handelt (sonst könnte man sie nicht von den 32-Bit-Return-Codes der Dateifunktionen unterscheiden).

Insgesamt gibt es drei verschiedene Klassen von Handles:

- Die drei zeichenorientierten Kanäle "CON:", "AUX:" und "PRN:" mit den Handles –1, –2 und –3. Diese Kanäle geben quasi direkt auf den dazugehörigen BIOS-Funktionen aus.

- Die sogenannten Standardkanäle, die man mittels "Fforce()" auch umlenken kann:

stdin (0)	Standardeingabe
stdout (1)	Standardausgabe
stdaux (2)	serielle Schnittstelle
stdprn (3)	parallele Schnittstelle

  Zusätzlich gibt es noch die Standardhandles 4 und 5, die allerdings zur Zeit von GEMDOS nicht benutzt werden.

- die Dateihandles (ab Handle 6 aufwärts).

Ein-/Ausgabeumlenkung

Während die zeichenorientierten Kanäle direkt mit dem BIOS und die eigentlichen Dateihandles direkt mit irgendwelchen geöffneten Dateien verbunden sind, muß man sich unter den Standardkanälen eher Platzhalter vorstellen.

Normalerweise sind die Standardkanäle folgendermaßen belegt:

Name	Nummer	normalerweise verbunden mit
stdin	0	CON: (–1)
stdout	1	CON: (–1)
stdaux	2	AUX: (–2)
stdprn	3	PRN: (–3)

Die Belegung der Kanäle 4 und 5 ist nicht definiert.

Die Funktion "Fforce()" erlaubt es, einem Standardkanal ein neues Ziel zu geben. Mit

```
Fforce (1, -2);
```

könnte man beispielsweise alle Bildschirmausgaben auf die serielle Schnittstelle umlenken.

"Fdup()" macht Kopien von Dateihandles. Meistens wird es benutzt, um nach erfolgter Umlenkung die alte Zuordnung der Standardkanäle wiederherstellen zu können. Das kann zum Beispiel so aussehen:

```
WORD handle, dup;

handle = Fcreate ("TMPFILE", 0);

dup = Fdup (1);      /* Kopie von stdout */
Fforce (1, handle); /* umlenken */

/* ... Kommandos bei umgelenkter Standardausgabe ausführen */

Fclose (handle);
Fforce (1, dup);     /* Umlenkung rückgängig */
```

Das ist leider nur die Theorie. In der Praxis ist die Ein-/Ausgabeumlekung unter GEMDOS (zumindest bis einschließlich GEMDOS 0.19) mit erheblichen Fehlern behaftet:

- Wenn man "Fdup()" anwendet und dann mittels "Pexec()" ein Programm startet, kann es zum Verlust von Handles und zur Zerstörung GEMDOS-interner Strukturen kommen (Problem: GEMDOS kommt praktisch gar nicht mit mehrfach von verschiedenen Prozessen geöffneten Dateien zurecht).

- Vor GEMDOS 0.15 funktionierten weder Umlenkung auf den Drucker noch die Umlenkung aller Funktionen zur Zeichenausgabe ("C....").

Ein-/Ausgabeumlenkung in Shells

Vielleicht das wichtigste Merkmal eines "reinrassigen" Tools in Kommandoshells ist die Beschränkung auf die Zeichenkanäle "stdin" und "stdout". In der C-Standardbibliothek bezeichnet man damit die "Dateien", die (mehr oder weniger) fest mit dem Bildschirm für Ein- und Ausgabe verbunden sind. Auf GEMDOS-Ebene handelt es sich um die Standardkanäle 0 (stdin) und 1 (stdout), die man mit allen GEMDOS-Funktionen direkt benutzen kann, ohne sie erst öffnen zu müssen. Auch die GEMDOS-Standardfunktionen wie "Cconout()" oder "Cconrs()" benutzen immer Kanal 0 und 1.

Das Schöne an den GEMDOS-Standardkanälen ist, daß man sie "umlenken" kann. Ein Beispiel: wenn das Archivierungsprogramm ARC ein Inhaltsverzeichnis ausgibt, benutzt es den GEMDOS-Kanal "stdout" (Kanal 1). In den meisten Shells kann man nun die Ausgabe von "stdout" umlenken. Das sieht dann beispielsweise so aus:

```
ARC V TEST.ARC >FILELIST
```

Die Shell sieht dann das Kommando ">FILELIST" und wertet dies als Aufforderung, auf "stdout" vorgenommene Ausgaben in die Datei "FILELIST" zu schreiben. Also wird diese Datei angelegt, und der Kanal "stdout" mittels der GEMDOS-Funktion "Fforce()" umgeleitet. Dabei sind auch Variationen möglich:

- Außer einem Dateinamen kann man auch jede Dateibezeichnung angeben, die GEMDOS bei "Fopen()" versteht, beispielsweise also "AUX:" (für die serielle Schnittstelle) oder "PRN:" (für die parallele Schnittstelle).

- Ein doppeltes Größerzeichen wird normalerweise so verstanden, daß die angegebene Datei nicht neu angelegt wird, sondern daß die Daten hinten an die bestehende Datei angehängt werden sollen.

Was für "stdout" gilt, stimmt auch für "stdin". Der Komprimierer ZOO kennt zum Beispiel einen Modus, bei dem die Namen der zu komprimierenden Dateien von "stdin" eingelesen werden.

Man könnte damit beispielsweise mit einem Editor eine Textdatei FILES.TXT mit den Namen der zu komprimierenden Dateien anlegen und dann mittels

```
ZOO aI ZOOFILE.ZOO <FILES.TXT
```

von ZOO verarbeiten lassen. Hier wird also ganz analog das Kleinerzeichen als Symbol für die Eingabeumlenkung benutzt.

Unter UNIX gibt es nun eine ganze Reihe von Standardprogrammen, die einfach nur Zeichen von "stdin" lesen, irgendwie verarbeiten und dann auf "stdout" ausgeben. So ein Programm nennt man dann "Filter". Viele Programme machen dies davon abhängig, ob man sie mit oder ohne Parameter aufruft.

Besonders praktisch sind solche Programme, wenn man eine Shell besitzt, die sogenannte "Pipes" erlaubt. Durch eine Pipe kann man die Standardausgabe eines Programms mit der Standardeingabe eines zweiten Programms verbinden. Bei einem Multitasking-Betriebsystem laufen die einzelnen Programme dann sogar tatsächlich gleichzeitig ab.

Und wieder ein Beispiel:

```
ls | zoo aI zoofile
```

Der senkrechte Balken ist das sogenannte Pipe-Symbol, wie es jede UNIX- und UNIX-artige Shell versteht. Die Ausgabe des ersten Kommandos (ls) wird direkt als Eingabe für ZOO benutzt.

Echte Pipes sind unter GEMDOS natürlich nicht möglich. Mit Hilfe der Ein-/Ausgabeumlenkung über eine temporäre Datei kann man allerdings ein ähnliches Ergebnis erzielen:

```
ls >TEMPFILE
zoo aI <TEMPFILE
```

Pipes können selbstverständlich auch mehr als zwei Stufen haben. Das Schöne an diesen Mechanismen: solange die Programme nur sauber über "stdin" und "stdout" operieren, brauchen Sie selbst nichts darüber zu wissen: die Hauptarbeit liegt bei der Shell.

Ein Konzept für stderr

Ganz problemlos ist dies allerdings nicht; nehmen wir mal den folgenden Fall:

```
ARC V ARCHIV.ARC >output
```

Mal angenommen, die Archivdatei ARCHIV.ARC ist gar nicht vorhanden: Wie soll dann ARC dem Benutzer mitteilen, daß ein Fehler vorliegt? Die einfache Lösung unter UNIX: zusätzlich zum Standardausgabekanal "stdout" gibt es den Standard-Fehler-Kanal "stderr", auf dem man Fehler und Warnungen ausgeben kann. Die wichtigsten Eigenschaften von "stderr":

- "stderr" ist normalerweise mit dem Bildschirm verbunden.
- Ein technisches Detail: bei Verwendung der C-Standardbibliothek ist "stderr" immer ungepuffert (während "stdin" und "stdout" zeilenweise gepuffert arbeiten).
- Unter UNIX ist "stderr" Standardkanal 2; die Umlenkung in der Shell läuft mittels "2>" bzw. "2<".

Ein besonderer Leckerbissen: wie eben schon angedeutet, kann man von "stderr" auch Zeichen lesen. Wozu man das nun wieder braucht? Das Standardtool "more" liest beispielsweise Zeilen von "stdin" und bietet diese seitenweise (zum Blättern) an. Nur: woher soll "more" seine Tastendrücke bekommen, wenn "stdin" schon mit einer Datei verbunden ist? Die einzige Möglichkeit ist, auch das Lesen von "stderr" zu erlauben!

Kommen wir zu dem unvermeidlichen (?) Haken an der Sache: Leider hat Atari im GEMDOS keinen Standardfehlerkanal vorgesehen. Die Hersteller der C-Compiler haben deshalb verschiedene Methoden ersonnen, um einen Ersatz anzubieten.

Da gäbe es zunächst die Compiler, die "stderr" und "stdout" einfach auf den gleichen Kanal lenken, womit natürlich fast alle Vorteile zunichte gemacht werden. Nicht zur Übernahme empfohlen.

Eine zweite Möglichkeit ist, "stderr" fest mit dem Bildschirm zu verbinden (über GEMDOS-Kanal −1). Damit sind zwar "stdout" und "stderr" voneinander getrennt, aber "stderr" kann dadurch nicht mehr umgelenkt werden. Ferner gehen viele UNIX-Programme, die man gerne portieren möchte (z. B. "compress"), davon aus, daß "stderr" Kanal 2 ist. Diese Lösung wird meines Wissens von der Turbo-C-Library benutzt. Elegant, aber nicht ganz problemlos!

Die offensichtliche Lösung ist, für "stderr" wie unter UNIX Standardkanal 2 zu benutzen. Kanal 2 wird von GEMDOS normalerweise für die serielle Schnittstelle benutzt.

Allerdings hat Atari bereits 1986 im "GEMDOS Reference Manual" darauf hingewiesen, daß man besser direkt über das BIOS auf die serielle Schnittstelle zugreifen sollte. Ansonsten kann man auch einen der "freien" Standardkanäle benutzen (nachdem man ihn mit "Fforce()" auf die serielle Schnittstelle umgelenkt hat).

Alle zum Mark-Williams-C-Compiler gehörigen Tools und auch die von diesem Compiler erzeugten Programme nutzen für "stderr" Kanal 2.

Der Nachteil: normalerweise ist Kanal 2 eben mit der seriellen Schnittstelle verbunden – nur wenn man in seiner Shell Kanal 2 auf den Bildschirm umlenkt, bekommt man eventuelle Fehlermeldungen zu Gesicht (davon sind auch die meisten GNU-Programme betroffen).

Nichtsdestotrotz gibt es keine bessere Alternative. Atari-Programmierer Allan Pratt machte daher folgenden Vorschlag: beim Start sollte jedes Programm testen, ob Kanal 2 bereits auf eine Datei umgelenkt ist. Dazu kann man die Funktion "isatty()" (wie in der GEMDOS-Funktionsübersicht beschrieben) verwenden.

Wenn dem nicht so ist, sollte man mittels "Fforce (2, –1)" die Ausgabe auf den Bildschirm umlenken.

Damit sind schon mal ein paar Probleme gelöst. Startet man ein solches Programm vom Desktop oder von einer "unwissenden" Shell, dann wird die Ausgabe von "stderr" auf den Bildschirm umgelenkt.

Hat man eine intelligente Shell, und möchte man "stderr" in eine Datei umlenken, dann geht das auch.

Ein kleiner Haken bleibt jedoch. Ein Kommando wie

```
ARC X TESTFILE 2>PRN:
```

also: "Bitte gib Deine Fehlermeldungen auf dem Drucker aus" wird nicht richtig verstanden.

Aber auch dieses Problem läßt sich leicht aus der Welt schaffen. In jeder "guten" Shell kann man Environmentvariablen setzen.

Wenn der Startupcode das Vorhandensein der Environmentvariablen "STDERR" als Aufforderung wertet, Kanal 2 einfach dort zu lassen, wo er ist, sind alle Probleme beseitigt.

Das Schöne an dieser Methode im Überblick:

1. "stderr" verhält sich hier praktisch genauso wie unter UNIX.
2. Mit einer guten Shell kann man mit "stderr" alles machen, was man sich wünschen kann.
3. Diese Methode kollidiert nicht mit alten Programmen und benötigt ausschließlich vorhandene und funktionierende Funktionen des GEMDOS.

Speicherverwaltung

Einleitung

GEMDOS verwaltet den Speicher mittels zweier verketteter Listen (je eine für freie und eine für belegte Speicherblöcke). In den existierenden GEMDOS-Versionen wird diese Liste allerdings in einem statischen Speicherbereich geführt, so daß es eine feste Grenze für die maximale Zahl von freien und belegten Speicherblöcken gibt (mehr dazu bei der Erklärung des "GEMDOS-Pools"). Für jeden belegten Block wird vermerkt, welcher Prozeß ihn alloziert hat. Damit hat GEMDOS die Möglichkeit, bei Programmende alle angeforderten Speicherbereiche freizugeben.

Zur Suche nach freien Speicherblöcken benutzt GEMDOS ein "rotating first-fit"-Verfahren. Dabei bedeutet "first-fit", daß GEMDOS immer den ersten ausreichend großen Block nimmt (und nicht etwa denjenigen, dessen Größe der angeforderten am nächsten kommt). "rotating" steht dafür, daß GEMDOS diese Suche nicht immer am Anfang, sondern bei der letzten Fundstelle beginnt.

Die GEMDOS-Speicherverwaltung ist speziell für relativ wenige, große Speicherblöcke geeignet. Wer in seinem Programm viele, kleine Blöcke braucht, sollte den Speicher in größeren Mengen anfordern und dann selbst verwalten (wie es die meisten C-Bibliotheken bei "malloc()" machen). Dies ist übrigens kein GEMDOS-eigenes Problem – andere Betriebssysteme wie UNIX funktionieren genauso.

Weitere Hinweise zur Speicherverwaltung finden sich bei der Erklärung der GEMDOS-Funktion "Malloc()". Eine Regel ist allerdings so wichtig, daß sie hier nochmals wiederholt wird: "Du sollst nicht auf Speicherbereiche zugreifen, die Dir nicht gehören!"

Der GEMDOS-Pool

Vorsicht! In diesem Abschnitt geht es um höchst komplexe Vorgänge tief im Innern des GEMDOS, für die man sich eigentlich gar nicht interessieren müßte – wäre da nicht eine beeindruckende Zahl von Fehlern, Patchprogrammen, Relikten aus alter Zeit und anderen gräßlichen Dingen im Spiel.

Eine der Aufgaben von GEMDOS ist die Verwaltung des Systemspeichers. Daraus ergibt sich natürlich eine Einschränkung für die Möglichkeiten, die GEMDOS selbst hat: anders als die höherliegenden Betriebssystemschichten AES und VDI kann GEMDOS eben *keine* Aufrufe von "Malloc()" tätigen.

Nun hat GEMDOS leider eine Menge von Aufgaben zu erledigen, die dynamischer Natur sind. Dazu gehört nicht nur die Verwaltung freier und belegter Speicherblöcke, sondern auch die Buchführung über offene Dateien oder aktuelle Verzeichnisse. Dabei sollte man nicht vergessen, daß jeder GEMDOS-Prozeß für jedes GEMDOS-Laufwerk ein aktuelles Verzeichnis kennt.

Zur Unterbringung solcher Informationen verfügt GEMDOS über den "GEMDOS Pool", der (wen wundert's?) eine fixe Größe hat und bei Bedarf *nicht* automatisch vergrößert werden kann.

Wer schon mal eine Speicherverwaltung programmiert hat, wird nun zu Recht einwenden, daß man zumindest für eine ganz normale Implementation von "Malloc()" gar keinen eigenen Speicher braucht. Schließlich müssen ja nur Listen von freien und belegten Blöcken geführt werden, und dazu könnte man eigentlich problemlos jeweils die ersten Bytes eines Blocks mißbrauchen (dieses einfache Verfahren wurde auch von Kernighan und Ritchie, den Schöpfern der Sprache C, publiziert).

Leider legt GEMDOS seine Informationen im Pool ab, und so ist schon von daher die maximale Anzahl von durch "Malloc()" allozierten Speicherblöcken durch die Größe des Pools beschränkt. Warum Atari das noch nicht geändert hat? Viele Programme verlassen sich darauf, daß nacheinander allozierte Speicherblöcke auch an aufeinanderliegenden Adressen liegen – was natürlich nicht nur gar nicht immer der Fall ist, sondern natürlich auch nie dokumentiert war. Daher zur Zeit diese Rücksichtnahme auf fehlerhafte Programme von gestern, die Atari hoffentlich bei der nächsten größeren Überarbeitung von GEMDOS aufgeben wird.

Zurück zur Belegung des Pools: Jedes "aktive" Verzeichnis belegt zwei Einträge. Ein Verzeichnis ist aktiv, wenn es ein aktuelles Verzeichnis oder ein Wurzelverzeichnis ist oder wenn es eine offene Datei oder ein aktives Unterverzeichnis enthält. Wenn man eine Datei öffnet, die innerhalb einer zehnfach verschachtelten Ordnerhierarchie steckt, sind dadurch insgesamt zehn Verzeichnisse aktiv.

Zu beachten ist dabei, daß jedes Verzeichnis nur einmal im Pool auftaucht, egal aus wie vielen Gründen es offen ist. Zusätzlich braucht jede offene Datei einen Eintrag und jeder freie oder belegte Speicherblock 1/4 eines Eintrags.

Der "40-Ordner-Fehler"

Alte GEMDOS-Versionen (vor 0.15 im TOS 1.04) hatten nun ein großes Problem: War ein Verzeichnis einmal aktiv geworden, dann wurde der entsprechende Eintrag nie wieder aus dem GEMDOS-Pool entfernt. Und so konnte man im TOS 1.00 maximal 40 Ordner öffnen. Im TOS 1.02 (Blitter-TOS) entschärfte man das Problem, indem man die Größe des Pools vergrößerte (und damit natürlich den freien Speicherplatz verminderte).

Wie äußerte sich nun das Problem? Einmal konnte es bei Aufrufen von "Malloc()" zu Fehlermeldungen kommen, obwohl eigentlich noch Speicher vorhanden gewesen wäre. Der andere Effekt war, daß geöffnete Verzeichnisse scheinbar leer waren, weil einfach nicht genug Speicher zum Verwalten der Verzeichnisinformationen da war.

Was ist nun bei Atari unternommen worden?

Zunächst führte man ab TOS 1.02 einen Zeiger im ROM-Header (siehe "_sysbase") ein, über den diese Liste nachträglich erweitert werden kann. Dieser Zeiger (oder die fixe Adresse im TOS 1.00) wird vom Programm "FOLDR100.PRG" benutzt, um den GEMDOS-Pool zu erweitern. Für "100" kann eine beliebige Zahl eingesetzt werden – "FOLDR100.PRG" erweitert den GEMDOS-Pool um 200 Einträge zur Aufnahme von 100 aktiven Verzeichnissen. "FOLDR100.PRG" befindet sich auf der Systemdiskette zu Atari-Festplatten, in praktisch allen Mailboxen und sollte auch bei jedem Atari-Händler zu bekommen sein. Viele Harddisktreiber ("AHDI", "HUSHI" und andere) sorgen bereits selbsttätig für eine Aufstockung des Pools.

In GEMDOS 0.15 (TOS 1.04) hat Atari viel verbessert. Der Pool hat noch immer eine feste Größe, wird aber erheblich besser genutzt. Insbesondere werden Einträge von nicht mehr aktiven Verzeichnissen wieder freigegeben, und auch die Wiederverwendung freigewordener Einträge ist geschickter geregelt. Konsequenterweise hat man den GEMDOS-Pool auch wieder auf die ursprüngliche Länge zurückgestutzt.

Durch das Öffnen vieler Verzeichnisse kann man nun eigentlich nicht mehr in Bedrängnis kommen, wohl aber durch allzu viele "Malloc()"-Aufrufe. Es kann also immer noch vorkommen, daß man bei "Malloc()" eine Fehlermeldung erhält, obwohl eigentlich noch Speicher da wäre. Statt dessen sollte man natürlich eine Laufzeitbibliothek benutzen, die den Speicher in Stücken von mindestens 16 Kilobyte per "Malloc()" anfordert und dann die Verwaltung selbst erledigt (alle mir bekannten C-Bibliotheken funktionieren so). Immerhin erhält man eine sinnvolle Fehlermeldung, die man eben nur richtig interpretieren muß.

Das Problem der aktiven Verzeichnisse wird durch die verbesserten Algorithmen bereits stark gelindert. Wenn tatsächlich trotzdem mal der GEMDOS-Pool überläuft, bekommt man eine Fehlermeldung auf den Bildschirm, und das System steht:

```
*** OUT OF INTERNAL MEMORY:
*** USE FOLDR100.PRG TO GET MORE.
*** SYSTEM HALTED ***
```

Das ist zwar nicht schön, aber besser als ein Weiterarbeiten in einem undefinierten Systemzustand. Der Tip, "FOLDR100" zu benutzen, ist allerdings kaum hilfreich: nach menschlichem Ermes-

sen kann das Problem im normalen Betrieb gar nicht auftreten (es sei denn, man versucht tatsächlich, viele sehr tief geschachtelte Dateien zu öffnen). Der Grund ist mit sehr hoher Wahrscheinlichkeit eine Zerstörung interner GEMDOS-Variablen durch ein fehlerhaftes Programm.

Ein solcher Übeltäter war pikanterweise GEMDOS selbst (in der Version 0.15, die man in TOS 1.04 und TOS 1.06 findet). Lange Rede, kurzer Sinn: bei diesen TOS-Versionen sollte man das Programm "POOLFIX3.PRG" im AUTO-Ordner haben (Bezugsquelle: Systemdiskette, Mailboxen, Atari-Händler).

Wenn man dennoch die Fehlermeldung zu sehen bekommt, dann liegt das meist daran, daß GEMDOS abgrundtief verwirrt ist. Selbst unter GEMDOS 0.19 kann man solche Effekte in Abhängigkeit mit der (nicht vollständig korrekt funktionierenden) Ein-/Ausgabeumlenkung provozieren.

Alternate RAM

Bei der Entwicklung der TT-Reihe stand Atari vor einem großen Problem: Wie kompatibel sollte die Hardware der neuen Rechnern zu den "alten" ST-Modellen sein? Erste Prototypen von 68020-Systemen, die auf Ebene der Hardware nicht vollständig kompatibel waren, wurden deshalb nie zur Marktreife entwickelt.

Hauptproblem waren die vielen Programme, die aus verschiedenen Gründen direkt Hardwarebausteine des ST ansteuern. In manchen Fällen hatten die Programmierer keine andere Wahl (Gerätetreiber), in vielen Fällen jedoch war es einfach die mangelnde Voraussicht der Programmierer.

So entschloß sich Atari, im TT praktisch sämtliche Bausteine des ursprünglichen ST beizubehalten: eine Entscheidung, die nicht nur die Fertigstellung massiv verzögert, sondern auch mit Sicherheit die Kosten des Rechners unnötig erhöht hat.

Nun war der ST eigentlich ein 24-Bit-System, und die "alte" Architektur ist nicht in der Lage, mit den 32 Bit großen Speicheradressen des TT zurechtzukommen.

Daher verpaßte man dem TT zwei verschiedene Arten von Speicher: das ST-RAM (das immer im 24-Bit-Adreßbereich, also in den ersten 16 MByte liegt) und das restliche RAM – das "Alternate RAM".

Als Faustregel gilt: Die "alte" ST-kompatible Hardware kann ausschließlich auf Adressen im ST-RAM zugreifen.

Dazu gehören:

- der Videochip. Daher darf man "Setscreen()" nur für Adressen im ST-RAM benutzen. Das Ganze stellt aber kein ernsthaftes Problem dar, weil "ordentliche" Grafikprogramme sowieso nur ausschließlich über VDI und AES ausgeben dürfen.

- der DMA-Soundchip: für diesen Soundchip (im STE und TT) gibt es bislang leider keine Softwareunterstützung durch das Betriebssystem.

- ACSI-DMA: der Direktzugriff auf Festplatten am ACSI-Port sollte sowieso den dafür spezialisierten Treibern überlassen bleiben. Die Laserdrucker der SLM-Reihe sollten ausschließlich via VDI oder Diablo-Treiber angesteuert werden.

- ...und schließlich darf man selbstverständlich *nicht* davon ausgehen, daß – wie auf einem 68000er-System – die obersten acht Bits einer Adresse unbenutzt sind und anderweitig benutzt werden können.

Sämtliches RAM, das *nicht* ST-kompatibel ist, heißt "Alternate" (also "anderes") RAM. Weitere Informationen hat man nicht. Im TT kann "Alternate RAM" sowohl das schnelle TT-RAM (auf das man per SCSI-DMA zugreifen kann) als auch ganz besonders langsames RAM auf einer VME-Bus-Karte sein (etwa bei einer Grafikkarte). "Maddalt()" erlaubt es, auch noch nach der Systeminitialisierung Blöcke von Alternate RAM in die GEMDOS-Speicherliste einzuhängen.

GEMDOS verwaltet ST-RAM und Alternate RAM in zwei völlig voneinander getrennten Listen. Wichtige Folge: Auf einem TT ist der freie Speicher bereits beim Systemstart segmentiert – in zwei große Blöcke aus ST-RAM bzw. Alternate RAM.

GEMDOS nimmt sich dieser Problematik mit zwei verschiedenen Strategien an. Einerseits kann man im Header einer Programmdatei festlegen, ob der Programmcode im Alternate RAM liegen und ob Speicher aus dem Alternate RAM alloziert werden darf (mehr dazu später). Zusätzlich hat man mit "Mxalloc()" eine genaue Kontrolle darüber, welche RAM-Art benutzt wird.

In den letzten Jahren hat sich die Qualität der Software signifikant verbessert. Daher kann man davon ausgehen, daß die "nächste" Rechnerfamilie von Atari auf Ebene der Hardware nicht mehr voll kompatibel sein wird (von der CPU aus der 68000-Familie abgesehen). Also: keinerlei direkte Zugriffe auf ST- oder TT-Hardware machen! Ein Programm, das heute nicht einwandfrei im Alternate RAM laufen kann, wird auf der nächsten Rechnergeneration erst recht nicht funktionieren. Wer direkt die Hardware ansprechen muß, sollte dies immer über eine Treiberschnittstelle – zum Beispiel über "Meta-DOS" – machen!

Prozesse

Was ist ein GEMDOS-Prozeß?

GEMDOS ist kein Multitasking-System, das heißt, es kann nicht mehr als ein Programm gleichzeitig ausgeführt werden. Dennoch sind die grundlegenden Strukturen dafür vorhanden: jedes geladene Programm wird von GEMDOS als eigenständiger *Prozeß* verwaltet. Mit Hilfe von "Pexec()" können weitere Programme nachgeladen und wie ein Unterprogramm ausgeführt werden. Währenddessen wird das aufrufende Programm unterbrochen.

GEMDOS merkt sich sämtliche prozeßspezifischen Daten in der *Basepage*, einer Datenstruktur, die automatisch beim Programmstart angelegt wird. Zu den in der Basepage abgelegten Informationen gehören:

- Zeiger auf die Basepage-Struktur des aufrufenden Prozesses
- Informationen über die verschiedenen Bestandteile des Programms nach dem Laden
- Informationen über aktuelles Laufwerk und aktuelle Verzeichnisse
- Informationen über die Zuordnung der Standardkanäle
- Informationen über die Tauglichkeit des Programms im "Alternate RAM"
- Inhalte der Register, die nicht von GEMDOS verändert werden dürfen

Alle diese Informationen werden also für jeden Prozeß separat gespeichert (auch wenn einige an nichtdokumentierten Stellen der Basepage verwahrt sind). Beim Starten eines Programms werden sie als aktuelle Einstellungen an den *Kindprozeß* weitervererbt.

Dies ist auch eine der ganz wenigen Stellen, an denen GEMDOS über das Vorbild MS-DOS herausragt: dort werden die aktuellen Verzeichnisse nur einmal global verwaltet. Die Folge ist, daß nach Start eines Programms aktuelle Verzeichnisse und aktuelles Laufwerk verstellt sein können.

Zu jedem Prozeß gibt es einen *Elternprozeß*, der ihn gestartet hat. Einzige Ausnahme ist der *Urprozeß*, der beim Aktivieren von GEMDOS angelegt wird. Damit ergibt sich eine *Prozeßhierarchie*. Mit den aktuellen GEMDOS-Versionen kann man nur neue Prozesse erzeugen, indem man den aktuellen Prozeß bis zur Beendigung des aufgerufenen Programms suspendiert. Damit kann man sich die Prozeßhierarchie als Stapel vorstellen. Unter einem multitaskingfähigen

Betriebssystem wird das Ganze eher wie ein von einem *Wurzelprozeß* ausgehender Baum aussehen.

GEMDOS kennt immer nur einen *aktuellen Prozeß*. Dazu gibt es im GEMDOS-Inneren die Zeigervariable "act_pd", die auf die entsprechende Basepage-Struktur zeigt. "act_pd" ist keine echte Systemvariable, sondern vielmehr eine lokale Variable von GEMDOS.

Zur Ermittlung ihrer Position gibt es von Atari ein Programmbeispiel (Quelle: "Rainbow TOS Release Notes"), das wir hier in abgewandelter Form abdrucken:

```
/* Ermittle Adresse der GEMDOS-Variable act_pd. Die Funktion
   liefert also einen Zeiger auf einen BASEPAGE-Zeiger.

   Ab TOS 1.02 enthält der ROM-Header den Zeiger. In TOS 1.00 liegt
   der Zeiger bei Adresse $602C, es sei denn, es handelt sich um
   ein spanisches TOS (dann bei $873C).

   Wichtiger Hinweis: funktioniert unter Umständen nicht mit
   gepatchten Versionen von TOS 1.00 */

BASEPAGE **GetRun (void)
{
     LONG savessp = Super (0L);
     OSHEADER *O = *((OSHEADER **)(0x4f2L));
     Super ((void *) savessp);

     O = O->os_beg;        /* wegen Fehlers in alter AHDI-Version */

     if (O->os_version < 0x102)
     {
          if ((O->os_conf >> 1) == 4)     /* PAL-Modus wegshiften */
             return ((BASEPAGE **)0x873c); /* Spanisches TOS 1.0 */
          else
             return ((BASEPAGE **)0x602c);
     }
     else
          return O->p_run;
}
```

Prinzipiell ist es damit möglich, zwischen zwei parallel laufenden GEMDOS-Prozessen hin- und herzuschalten. Doch Vorsicht: Diese Variable darf laut Atari *auf keinen Fall* verändert

werden. Probleme mit künftigen multitaskingfähigen GEMDOS-Versionen wären vorprogrammiert!

Die Basepage

Die Basepage ist insgesamt 256 Bytes lang und hat folgende Struktur:

```
typedef struct basepage
{
     void *p_lowtpa;     /* Anfangsadresse der TPA (Offset 0) */
     void *p_hitpa;      /* Erstes Byte nach dem Ende der TPA
                            (Offset 4) */
     void *p_tbase;      /* Anfangsadresse des Programmcodes (TEXT-
                            Abschnitt) (Offset 8) */
     LONG  p_tlen;       /* Länge des Programmcodes (Offset 12) */
     void *p_dbase;      /* Adresse des Bereichs für
                            vorinitialisierte Daten (DATA-Abschnitt)
                            (Offset 16) */
     LONG  p_dlen;       /* Länge des Datenabschnitts (Offset 20) */
     void *p_bbase;      /* Adresse des Variablenbereichs (BSS-
                            Abschnitt) (Offset 24) */
     LONG  p_blen;     /* Länge des Variablenbereichs (Offset 28) */
     DTA  *p_dta;      /* Zeiger auf Default-DTA (Vorsicht! Zeigt
                          erst in die Kommandozeile) (Offset 32) */
     struct basepage
          *p_parent;      /* Zeigt auf die Basepage (BASEPAGE) des
                             aufrufenden Prozesses (Offset 36) */
     LONG  p_resrvd0;     /* reserviert */
     CHAR *p_env;         /* Adresse der Environment-Strings
                             (Offset 44) */
     CHAR  p_resrvd1[80]; /* reserviert */
     CHAR  p_cmdlin[128]; /* Kommandozeile (dabei wird im ersten
                             Byte die Anzahl der Zeichen einge-
                             setzt. Die maximale Länge der
                             Kommandozeile beläuft sich
                             erstaunlicherweise nicht auf 127,
                             sondern auf 124 Zeichen.)
                             (Offset 128) */

} BASEPAGE;
```

Wichtiger Hinweis: der Zeiger "p_parent" ist zwar offiziell dokumentiert, sollte aber nicht dazu mißbraucht werden, in den Speicherbereich des Aufrufers zu schauen.

Dies wäre ein eklatanter Verstoß gegen die Regel: "Du sollst nur auf Speicher zugreifen, der Dir selbst gehört".

TPA und Programmstart

Die TPA ("Transient Program Area") kommt beim Laden eines Programms ins Spiel. Die folgenden Informationen zum Ladevorgang beschreiben den (dokumentierten) IST-Zustand von GEMDOS.

Manche Details könnten sich allerdings in künftigen GEMDOS-Versionen ändern.

- Beim Programmstart wird der größte zusammenhängende Speicherbereich alloziert. Die Basepage wird in den ersten 256 Bytes der TPA angelegt.

- Anschließend werden allen zu "vererbenden" Informationen aus der Basepage des aufrufenden Programms übertragen. Der neuerzeugte Prozeß wird als Inhaber des durch die TPA belegten Speicherblocks eingetragen. Damit "gehört" der durch die TPA belegte Speicher dem gestarteten Programm.

- Anschließend wird der Header der Programmdatei ausgewertet (eine detaillierte Schilderung des Programmformats folgt später).

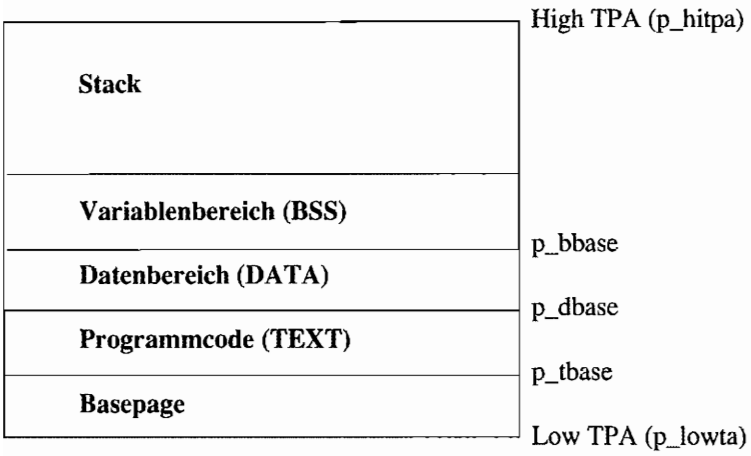
- Die Programmdatei wird geladen und die verbleibenden Felder in der Basepage gesetzt. Je nach Programmtyp wird das BSS-Segment oder der ganze verbleibende Rest der TPA gelöscht.

- Dann bekommt der neue Prozeß einen eigenen Stack, der direkt am Ende der TPA beginnt (und bekanntlich nach unten – also nach den niedrigeren Adressen hin – wächst).

- Als Parameter werden eine Null (LONG) und die Anfangsadresse der Basepage auf dem Stack übergeben.

- Zuletzt wird der neue Prozeß zum aktuellen Prozeß gemacht und durch einen Sprung zum ersten Byte des Textsegments aktiviert.

Den Aufbau der TPA entnehmen Sie bitte der Abbildung auf der folgenden Seite.



Je nach Speicheraufteilung beim Programmstart kann es also ohne weiteres passieren, daß der gesamte freie Speicher für das Programm reserviert worden ist. Das darf natürlich nicht so bleiben:

- Viele Betriebssystemfunktionen (VDI, "fsel_input()", "Pexec()") verbrauchen selbst Speicher!

- Unter einem multitaskingfähigen Betriebssystem ist es ausgesprochen unpopulär, den gesamten freien Speicherplatz des Systems zu blockieren. Bereits jetzt gibt es einige TOS-Erweiterungen, unter denen dies zu berücksichtigen ist (beispielsweise das "Multi-GEM" der Maxon GmbH).

Die Abhilfe ist einfach:

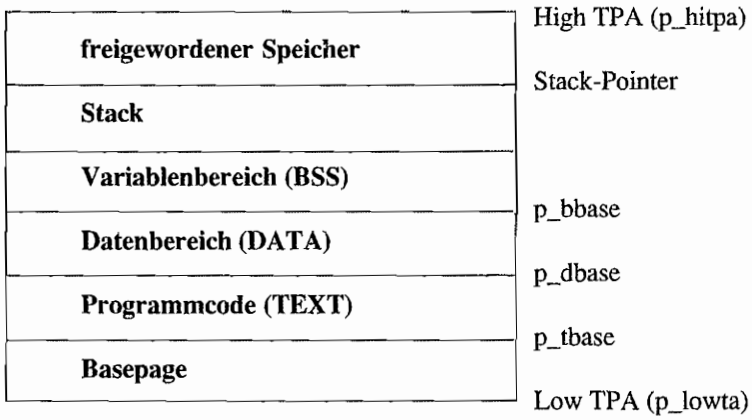
1. Eigenen Stack einrichten.

2. Überflüssigen Speicher wieder freigeben.

3. Bei Bedarf innerhalb des Programms per "Malloc()" Speicher nachfordern.

Beim Einrichten des eigenen Stacks hat man verschiedene Möglichkeiten. Einerseits kann man von Beginn an ein entsprechend großes Stück der BSS freihalten. Andererseits kann man sich natürlich auch einen Teil der "freien" TPA abknapsen. In beiden Fällen sollte man aber daran denken, daß der Stack zu niedrigeren Adressen hin wächst und der Stack-Pointer daher zu Beginn auf das Ende des freigehaltenen Speicherblocks zeigen muß!

Die TPA hat nunmehr diese Aufteilung (ob der Stack in oder über der BSS liegt, spielt in diesem Zusammenhang keine Rolle):



Nun muß noch der freigewordene Speicher an GEMDOS zurückgegeben werden. Kein Problem, da ja der durch die TPA belegte Speicherplatz dem Programm selbst gehört:

1. Man berechne, wieviel Speicherplatz man braucht. Dazu summiere man die Länge der Basepage (256), des Text-, Data- und BSS-Segments sowie gegebenfalls des Stacks auf. Die Längen sind in der Basepage angegeben, deren Anfangsadresse als Parameter auf dem Stack (4(sp)) mitgeteilt wird.

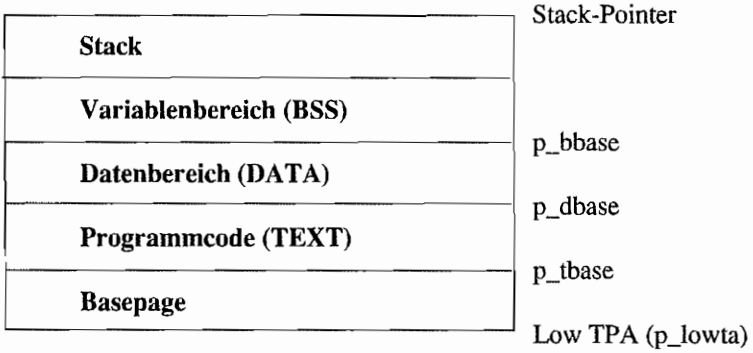
2. Der so erhaltene Wert ist die Anzahl der Bytes, auf die die TPA nun mit "Mshrink()" geschrumpft werden kann.

Und so könnte der Code am Beginn des Textsegments aussehen:

```
; Startup und Prozeßbeendigung
; Assembler: Madmac

          .text
move.l    4(sp),a0        ; Zeiger auf BASEPAGE-Struktur
lea       MyStack,sp      ; Stack-Pointer setzen
move.l    #$100,d0        ; Länge der Basepage
add.l     $c(a0),d0       ; Länge Text-Segment
add.l     $14(a0),d0      ; Länge Daten-Segment
add.l     $1c(a0),d0      ; Länge BSS
```

```
move.l    d0,-(sp)        ; Ergebnis auf den Stack
move.l    a0,-(sp)        ; BASEPAGE-Adresse auf den Stack
clr.w     -(sp)           ; Füllparameter
move.w    #$4a,-(sp)      ; GEMDOS Mshrink
trap      #1              ; GEMDOS-Trap
lea       $c(sp),sp       ; Stack aufräumen
jsr       main            ; Hauptprogramm aufrufen
move.w    d0,-(sp)        ; Return-Wert des Programms
move.w    #$4c,-(sp)      ; GEMDOS Pterm
trap      #1              ; GEMDOS-Trap

          .bss

          .ds.l     1000 ; 4000 Bytes Stack
MyStack:  .ds.l     2
```

Damit wäre dann die Anpassung der TPA beendet:



GEMDOS kann den so freigegebenen Speicher nun wieder für andere Zwecke nutzen – zum Beispiel zum Starten weiterer Programme oder um Speicheranforderungen mittels "Malloc()" zu bedienen.

Nach Beendigung des eigenen Programms wird dann die gesamte TPA wieder freigegeben.

Es sei noch auf zwei Sonderfälle hingewiesen:

TSR-Programme ("Terminate und Stay Resident") sind eine beliebte Methode, um Betriebssystemerweiterungen oder -veränderungen vorzunehmen. Beispiele sind "GDOS" (Betriebs-

systemergänzung bzw. -vervollständigung), "Meta-DOS" (Betriebssystemerweiterung) oder "POOLFIX3" (Betriebssystemmodifikation). Besonderes Kennzeichen von TSR-Programmen ist, daß sie Betriebssystem-Vektoren modifizieren und dann resident im Speicher verankert werden. Dafür gibt es die Funktion "Ptermres()", die ein Programm beendet, dabei aber einen bestimmten Anteil der TPA nicht wieder freigibt. Die Anzahl der zu schützenden Bytes kann man genauso wie für "Mshrink()" berechnen.

Wer die BASEPAGE-Struktur aufmerksam angesehen hat, wird festgestellt haben, daß die drei Programmsegmente gar nicht unbedingt in aufeinanderfolgenden Adreßbereichen liegen müssen. Damit wäre es prinzipiell denkbar, das Textsegment im ROM und die beiden anderen Segmente im RAM abzulegen.

Dazu muß man allerdings bei Programmstart die Zeiger aus der Basepage beachten und nur indirekt über die Zeiger auf Variablen zugreifen. Wegen der eher geringen Bedeutung ROM-gestützter Programme bieten leider nur wenige Programmiersprachen die Möglichkeit, solche Programme zu erzeugen.

Environment

Auch die Environment-Strings sind ein Erbe von MS-DOS. Man beachte, daß es mehrere Environment-Strings geben kann, denn "p_env" zeigt auf eine Liste von Strings, die mit einer "0" abgeschlossen ist (am Ende der Strings stehen also zwei Nullen hintereinander).

Bei jedem Programmstart mittels "Pexec()" wird eine Kopie des aktuellen Environments gemacht und dem gestarteten Programm übergeben. Möchte man dabei ein geändertes Environment übergeben, dann muß man selbst eine erweiterte Kopie anlegen und bei "Pexec()" einen Zeiger darauf übergeben. Das Environment von *Parent*-Prozessen kann *nicht* verändert werden.

GEM hat ein eigenes Environment, das beim Starten der AES nach Ausführung der Programme im AUTO-Ordner festgelegt wird (weitere Informationen dazu bei der AES-Funktion "shel_envrn()").

Das Standardformat für einen Environment-String sieht so aus:

```
VARIABLE=Wert
```

Variablenname und Wert sind also durch ein Gleichheitszeichen voneinander getrennt. Die einzige erlaubte Ausnahme gibt es beim ARGV-Verfahren zur Übergabe erweiterter Kommandozeilen.

Das ARGV-Verfahren

Das ARGV-Verfahren zur Übergabe erweiterter Kommandozeilen ist nach langer (kontroverser) Diskussion im Herbst 1989 von Ken Badertscher (Atari Sunnyvale) offiziell spezifiziert worden (Ken Badertscher, "GEMDOS Extended Argument (ARGV) Specification", in: INFO-ATARI 16 in Digest 595/89, (2. November 1989)). Es bietet folgende Vorteile gegenüber den herkömmlichen Kommandozeilen:

- Die Anzahl und Größe der Parameter ist praktisch nur durch den freien Speicher begrenzt.

- Einzelne Parameter dürfen auch Leerzeichen enthalten (in der normalen Kommandozeile sind die einzelnen Parameter untereinander durch Leerzeichen getrennt).

- Als "nullter" Parameter kann der Name (mit Pfad) des gestarteten Programms übergeben werden. Das erlaubt es, in C-Programmen "argv[0]" zu setzen und den Namen, unter dem das Programm gestartet worden ist, festzustellen.

Und so funktioniert das Ganze: Die Übergabe der erweiterten Kommandozeile erfolgt über das Environment. Die Environmentvariable ARGV (groß geschrieben!) zeigt an, daß dieses Verfahren benutzt wird. ARGV kann einen beliebigen Wert haben, allein seine Anwesenheit ist entscheidend. ARGV muß die letzte Environmentvariable sein, damit das aufgerufene Programm den "vorderen" Teil als sein normales Environment weiterbenutzen kann.

Die erweitertete Kommandozeile wird als Folge von Null-terminierten Strings hinter ARGV ins Environment geschrieben. Der erste String (entspricht dem bislang nicht benutzten argv[0]) enthält den Namen des gestarteten Programms, wie man ihn auch an "Pexec()" übergibt.

Die weiteren Strings enthalten die einzelnen Parameter, in denen auch Leerzeichen auftauchen dürfen. Das Ende der Liste wird durch eine doppelte 0 gekennzeichnet (wie bei einem "normalen" Environment). Leider heißt das auch: leere Parameter können *nicht* übergeben werden (es gibt tatsächlich Fälle, in denen man so etwas braucht!).

Zusätzlich übergibt man bei Pexec() als Längenbyte (erstes Byte in der Kommandozeile) den Wert 127, der wegen der existierenden Längenbeschränkung auf 125 Bytes bislang nicht angenommen werden konnte.

Der Längenwert von 127 ermöglicht es dem aufgerufenen Programm, sicherzustellen, daß die im Environment übergebenen Werte tatsächlich gültig sind und nicht etwa von einem Programm, das den ARGV-Standard nicht kannte, übriggelassen wurden.

ARGV beim Aufrufer

Um via ARGV-Parameter übergeben zu können, muß zunächst ein neues Environment für das aufzurufende Programm angelegt werden. Dazu berechnet man beispielsweise die Länge des bereits vorhandenen Environments, addiert die Länge der Kommandozeile und alloziert entsprechend viele Bytes. Dann wird das bestehende Environment kopiert (dabei gegebenenfalls eine bereits bestehende ARGV-Variable entfernt), die neue Variable ARGV und die Kommandozeilenparameter nacheinander angehängt (immer Null-terminiert). Eine letzte 0 schließt dann das Environment legal ab. Beim Aufruf mittels "Pexec()" übergibt man im Längenbyte der Kommandozeile den Wert 127.

ARGV beim gestarteten Programm

Zunächst muß man überprüfen, ob im Environment die Variable "ARGV" auftritt. Ist dies der Fall, und ist das Kommandozeilen-Längenbyte 127, dann findet man nach der ersten 0 nach "ARGV" (denn "ARGV" kann ja einen Wert haben) die einzelnen Kommandozeilenparameter. Nachdem der Startupcode die Argument-Zeiger (in C: argv[i]) gesetzt hat, sollte noch der erste Buchstabe von "ARGV" auf 0 gesetzt werden – nun hat das Environment wieder die Standardform.

Programmformat unter GEMDOS

Das GEMDOS-Programmformat hebt sich erheblich von dem einfacherer Betriebssysteme wie CP/M oder Atari-DOS (8-Bit) ab, da beim Programmstart das Programm an eine beliebige freie Stelle des Speichers geladen wird und dort ablauffähig sein muß. Dazu werden nicht die MC68K-spezifischen Adressierungsarten für indirekte Adressierung verwendet, sondern das geladene Programm kurzerhand für die entsprechende Stelle reloziert. Daher enthält eine GEMDOS-Programmdatei (wie sie mit "Pexec()" gestartet wird) zusätzlich noch Informationen für den Relozierungsvorgang. Außerdem kann man zu Debugging-Zwecken auch die Symbole (Labels) in die endgültige Datei übernehmen. Alle dazu notwendigen Informationen stellt im Normalfall der Linker bereit, so daß der Programmierer selten damit in Berührung kommt.

Eine GEMDOS-Programmdatei besteht also aus den aufeinanderfolgenden Komponenten:

- Dateikopf
- Text-, DATA- und BSS-Abschnitt
- eventuell Symboltabelle
- eventuell Relozierungstabelle

Der Datei-Kopf hat folgende Form:

```
typedef struct
{
     WORD ph_branch;      /* Branch an den Anfang des Programms
                             (0x601A) */
     LONG ph_tlen;        /* Länge des TEXT-Abschnitts */
     LONG ph_dlen;        /* Länge des DATA-Abschnitts */
     LONG ph_blen;        /* Länge des BSS-Abschnitts */
     LONG ph_slen;        /* Länge der Symboltabelle */
     LONG ph_resl;        /* reserviert; muß 0 sein */
     LONG ph_prgflags;    /* spezielle Flags */
     WORD ph_absflag;  /* 0: Relozierungsinformationen vorhanden */
} PH;
```

Im Feld "ph_prgflags" kann man weitere spezielle Flags angeben:

Fastload-Flag

Normalerweise löscht GEMDOS bei "Pexec()" die gesamte angelegte TPA. Das kann bei entsprechend viel RAM ziemlich viel Zeit in Anspruch nehmen (vor TOS 1.02 war zu allem Überfluß die Löschroutine besonders langsam). Wenn man Bit 0 in "ph_prgflags" setzt, wird lediglich die BSS vorinitialisiert. Programme, die davon ausgehen, daß allozierte Speicherblöcke auch tatsächlich leer sind, fallen dann allerdings auf die Nase (ein Zeichen für unsaubere Programmierung!). Doch Vorsicht: Anscheinend begehen auch Teile des Betriebssystems diesen Fehler, daher sollte man bei mindestens einem Auto-Ordner-Programm und einem Accessory das Fastload-Bit nicht setzen!

TT-Speicher

Beim TT kann man nicht mit allen Speicherbereichen das tun, was man vom ST gewohnt ist. Beispielsweise kann die Video-Hardware und die ACSI-DMA nur auf das ST-RAM zugreifen (mehr dazu bei der Hardware). Daher gibt es einen Mechanismus, mit dem man GEMDOS für *ein* bestimmtes Programm vorgeben kann, wohin es geladen werden soll und von wo aus Malloc()-Anforderungen bedient werden sollen. Wenn man Bit 1 in "ph_prgflags" setzt, heißt das für GEMDOS: Dieses Programm darf in das schnelle Alternate RAM geladen werden. Ein gesetztes Bit 2 signalisiert: Auch Malloc()-Anforderungen dieses Programms dürfen aus dem Alternate RAM bedient werden.

Eine besondere Situation ergibt sich, wenn im Programmkopf angegeben ist, daß das Programm in das Alternate RAM geladen werden darf, jedoch im ST-RAM mehr Speicher als

im Alternate RAM frei ist. Für diesen Fall kann man festlegen, wieviel Alternate RAM "genug" für das Programm ist. Das TPA-Größen-Feld liegt in den obersten vier Bits von "ph_prgflags". Seine 16 möglichen Werte legen in Schritten zu 128 KByte den benötigten Speicher fest (0 steht dabei für 128 KByte, 15 für 2 MByte). Zum so ermittelten Wert wird außerdem noch die Summe der drei Programmsegmente (Code, Daten und BSS) addiert.

Ein sauber geschriebenes Programm reserviert sich nach dem Programmstart in der TPA lediglich Platz für den Stack (Beispiel: der Startup-Code von Turbo-C). Bei solchen Programmen ist das TPA-Größen-Feld nur dann von Interesse, wenn ein ganz besonders großer Stack benötigt wird. Anders bei Programmen, die den größten Teil der TPA für sich behalten und nicht wieder mittels "Mshrink()" an GEMDOS zurückgeben.

Symboltabelle

Eine Symboltabelle im Digital-Research-Format besteht aus jeweils 14 Bytes langen Einträgen, die aus dem Symbolnamen (bis zu maximal acht Bytes; nur bei kürzeren Namen mit 0 abgeschlossen!!), dem Symboltyp (zwei Bytes) und dem eigentlichen Symbolwert (vier Bytes) besteht. Folgende Symboltypen werden unterstützt:

Wert	Symboltyp
$0100	in der BSS
$0200	im Programmtext
$0280	TEXT FILE – Start eines Objektmoduls (nur mit "ALN")
$02C0	TEXT FILE ARCHIVE – Start einer Library (nur mit "ALN")
$0400	im DATA-Bereich
$0800	External
$1000	Register
$2000	Globales Symbol
$4000	Equated
$8000	Defined

Viele Programmiersprachen benutzen allerdings ein eigenes Symbolformat (Mark-Williams-C, Turbo-C 2.0).

Die Relozierungsinformationen

Nach dem Ladevorgang wird das Programm automatisch reloziert. Dabei ist nur die Relozierung von 32-Bit-Werten (also Adressen) möglich. Vor GEMDOS 0.15 durften die Relozierungsinformationen maximal 32 Kbyte einnehmen.

GEMDOS kennt Programmdateien mit und ohne Relozierungsinformationen. Zur Unterscheidung dient das Feld "ph_absflag" im Programmkopf. Nur wenn es 0 ist, wird die folgende Relozierungstabelle ausgewertet.

Alte GEMDOS-Versionen (vor 0.15) gehen mit solchen Dateien allerdings nicht ganz korrekt um, daher empfiehlt Atari, statt dessen Dateien mit leerer Relozierungstabelle (siehe unten) zu verwenden.

Die Relozierungstabelle beginnt mit einem 32-Bit-Wert, der den Offset des ersten zu relozierenden Wertes relativ zum Beginn des Textsegments kennzeichnet.

Für alle folgenden Offsets (zwischen den zu relozierenden Werten) werden einzelne Bytes benutzt. Auch für Abstände über dem Betrag eines Bytes (255) ist gesorgt: wird als Offset eine 1 gefunden, was natürlich aufgrund der Eigenheiten des MC68K ausgeschlossen ist, wird automatisch zum Offset 254 addiert.

Für besonders große Abstände zwischen zu relozierenden Werten kann dieser Vorgang auch mehrfach wiederholt werden. Ist die Relozierungstabelle leer, dann findet man als ersten Long-Wert eine 0.

Weitere Standardformate

Sehr ähnlich wie eine Programmdatei sieht eine Objektdatei im Digital-Research-Format aus:

```
typedef struct
{
     WORD magic;          /* $601A */
     LONG tsize;          /* Größe des Textsegments */
     LONG dsize;          /* Größe des DATA-Segments */
     LONG bsize;          /* Größe der BSS */
     LONG ssize;          /* Größe der Symboltabelle */
     CHAR reserved[10];   /* alle auf 0 setzen */
} OSHEADER;
```

Schließlich noch das Format für das DR-Format für Archivdateien (oder auch Bibliotheken): eine Archivdatei enthält normalerweise weitere Objektdateien (kann aber auch beliebige andere Dateien enthalten). Der Dateikopf besteht lediglich aus dem Word $FF65.

Es folgen beliebig viele Dateien, jeweils eingeleitet mit einer ARHEADER-Struktur. Das Ende der Archivdatei wird durch $0000 gekennzeichnet.

```
typedef struct
{
     CHAR a_fname[14];   /* Dateiname */
     LONG a_modti;       /* Zeitpunkt des letzten Zugriffs */
     BYTE a_userid;      /* nicht benutzt */
     BYTE a_gid;         /* nicht benutzt */
     WORD a_fimode;      /* Filemodus */
     LONG a_fsize;       /* Dateilänge */
     WORD reserved;      /* 0 */
} ARHEADER;
```

GEMDOS-Vektoren

Die Prozessoren der 68000er-Reihe kennen 256 Vektoren, die für Dinge wie Traps, Exceptions oder Interrupts von I/O-Bausteinen verwendet werden.

Ihre Adressen liegen in den ersten 1024 Bytes des Adreßraums (genaugenommen sind es nur die Vektoren 2 bis 255, und die Vektoren beginnen erst bei Adresse 8...).

Zusätzlich sind für GEMDOS acht *logische* Vektoren (256 bis 263) definiert. Ihre Adressen liegen bei Adresse $400 (1024), sollten aber immer nur mit "Setexc()" verändert werden – ihre tatsächliche Adresse muß *nicht* bei $400 liegen! Am Ende jeder Routine sollte zu der Adresse gesprungen werden, die "Setexc()" als bisherigen Inhalt des entsprechenden Vektors zurückgeliefert hat.

Derzeit sind nur die ersten drei Vektoren benutzt – alle anderen sind für künftige Benutzung durch GEMDOS reserviert.

Vektor 256: Systemtimer (etv_timer)

Dieser Vektor wird periodisch aufgerufen. Bei allen bisher bekannten Maschinen tritt der Aufruf 50mal pro Sekunde auf (intern wird dazu der 200-Hz-Timer benutzt). Darauf sollte man sich allerdings nicht verlassen, erhält man doch als einzigen Parameter auf dem Stack (4(sp)) die Anzahl der vergangenen Millisekunden.

Hier installierte Routinen sollten möglichst kurz gehalten werden – immerhin beeinflussen sie die Gesamtleistung des Rechners. Um das Retten der Register kümmert sich das System selbst, daher darf man alle Prozessorregister benutzen.

GEMDOS benutzt den Systemtimer, um eine eigene Uhr zu führen.

Vektor 257: Critical Error Handler (etv_critic)

Dieser Vektor gehört eigentlich eher zum BIOS – aber da ihn Atari auch unter den GEMDOS-Vektoren führt (GEMDOS Reference Manual), haben wir ihn auch hier dokumentiert.

Das BIOS benutzt diesen Vektor, um auf *kritische* Fehler zu reagieren. Kritisch sind Fehler, die auf Probleme mit der Hardware (zum Beispiel nicht lesbare Sektoren) zurückzuführen sind.

Wer nur ein Diskettenlaufwerk hat, weiß, daß das BIOS Zugriffe auf Laufwerk "B:" selbst abfängt und den Benutzer zum Einlegen der entsprechenden Diskette auffordert. Auch dieser Mechanismus wird mit Hilfe des Critical Error Handlers realisiert: man hat einfach den Zustand "andere Diskette muß eingelegt werden" als BIOS-Fehler definiert.

Als ersten Parameter auf dem Stack (also bei 4(sp)) erhält man die Fehlernummer (als 16-Bit-Wort). Bei der Rückkehr aus dem Handler darf Register D0 folgende Werte enthalten:

Wert in d0	Bedeutung
$00010000	Zugriff wiederholen ("Retry")
$00000000	Fehler ignorieren
$FFFFFFXX	Mit BIOS-Fehler abbrechen (der Fehler wird dann im Normalfall an GEMDOS weitergereicht)

Der Handler darf die Register D3 bis D7 und A3 bis A6 verändern. Aufrufe von GEMDOS- oder AES-Funktionen sind *nicht* erlaubt. Begründung: GEMDOS ist nicht re-entrant, und der Fehler könnte von einem GEMDOS-Aufruf ausgelöst worden sein.

Aufrufe von AES-Funktionen könnten zu einer AES-Prozeß-Umschaltung führen – und wer weiß schon, ob der *andere* AES-Prozeß nicht vielleicht als nächstes einen GEMDOS-Aufruf vornimmt?

Der Critical-Error-Handler in älteren GEM-Versionen (vor 3.0) hat leider einen Fehler, der in ungünstigen Fällen zum Systemabsturz führen kann: während die Alert-Box zu sehen ist, ist das AES-Prozeß-Switching nicht unterbunden. Daher könnte *währenddessen* ein Programm per Timer-Event zum Zuge kommen und selbst wieder einen GEMDOS-Aufruf tätigen.

Ist die auf dem Stack übergebene Fehlernummer "–1" (also die Fehlernummer für "allgemeine Fehler"), *und* ist der installierte Handler der des AES, dann werden noch einige zusätzliche Werte auf dem Stack benutzt (dies ist nicht offiziell dokumentiert!):

Stack-Offset	Bedeutung
8(sp)	Laufwerksnummer (0 ist A: etc)
6(sp)	BIOS-Fehlernummer

Und hier ein Beispiel für einen Critical Error Handler, wie man ihn in rein textorientierten Programmen (ohne GEM und Maus) benutzen kann, und der dem von MS-DOS ähnelt:

```
; toscrit.s
; Critical-Error Handler für Programme ohne Maus!
; 30.09.89

        .globl  toscritic
        .text

toscritic:
        lea     Message,a0      ; Fehlermeldung ausgeben
        bsr     PrintIt

        move.l  #$20002,-(sp)
        trap    #13             ; Bconin von CON:
        addq.l  #4,sp
        and.w   #$5f,d0

        cmp.b   #"A",d0
        beq     IstA
        cmp.b   #"R",d0
        beq     IstR
        cmp.b   #"I",d0
        bne     toscritic       ; keine der drei Tasten -> von vorne

        clr.l   d0              ; Fehler ignorieren
        rts
IstA:
        move.w  4(sp),d0        ; Fehlernummer an GEMDOS melden
        ext.l   d0
        rts
IstR:
        moveq   #1,d0           ; Retry zurückmelden
        swap    d0
        rts
```

```
PrintIt:                        ; zeichenweise per BIOS ausgeben
        clr.l   d0
        move.b  (a0)+,d0
        tst.b   d0
        beq     endprint

        move.l  a0,-(sp)
        move.w  d0,-(sp)
        move.w  #$2,-(sp)
        move.w  #$3,-(sp)
        trap    #13
        addq.l  #6,sp
        move.l  (sp)+,a0
        bra     PrintIt
endprint:
        rts

        .data
Message:
        .dc.b   13,10           ; Zeilenvorschub
        .dc.b   "BIOS-Error: (A)bort, (R)etry, or (I)gnore?",0
```

Vektor 258: Terminate Handler (etv_term)

Dieser Vektor wird vor Prozeßbeendigung einmal durchsprungen (Merke: Programme, die per GEMDOS auf der Konsole ausgeben, können mit CTRL-C abgebrochen werden). Dies ist mithin die geeignete Stelle, um gegebenenfalls installierte Funktionen wieder zu deinstallieren. Nicht möglich ist es hingegen, irgendwelche GEMDOS-Aufrufe zu machen (GEMDOS ist *nicht* re-entrant, und an dieser Stelle steckt man mitten in einem GEMDOS-Aufruf) oder gar die Beendigung des Programms sauber zu verhindern.

GEMDOS-Erweiterungen

GEMDOS hat eine in vielerlei Hinsicht altmodische Architektur, die Erweiterungen recht schwer macht. Atari hat sich dieses Problems mit zweierlei Konzepten angenommen:

Netzwerk-Standard und File-Locking

Aus den oben genannten Gründen ist es ein schwieriges Unterfangen, Netzwerksoftware auf GEMDOS-Basis zu entwickeln. Lange Zeit kam erschwerend hinzu, daß es keinerlei Standard

in Hinsicht auf die zu implementierenden Funktionsaufrufe gab. Damit ist seit Frühjahr 1991 Schluß: Atari hat die entsprechenden unter MS-DOS üblichen Definitionen übernommen (siehe in der Befehlsreferenz). Sie sind genau dann verfügbar, wenn der "_FLK"-Cookie gesetzt ist. Ein "_NET"-Cookie informiert über die Anwesenheit eines Netzwerks (nicht jedes Netzwerk beherrscht File-Locking, und File-Locking kann auch ohne Netzwerk wichtig sein).

Meta-DOS

Meta-DOS ist eine GEMDOS-Erweiterung, die die Installation vielfältiger Gerätetreiber erlaubt. Dabei wird zwischen physikalischen Treibern (die man über erweiterte XBIOS-Aufrufe steuern kann) und logischen Treibern (die das Dateisystem übernehmen) unterschieden.

In Hinsicht auf GEMDOS-Aufrufe verhält sich Meta-DOS fast transparent. Bekannte Einschränkungen bzw. Unterschiede (Stand: Meta-DOS 1.7) sind:

- die Laufwerksnummer darf bis "Z:" gehen
- die Umlenkung der Standardkanäle auf von Meta-DOS verwaltete Dateien ist nicht möglich
- Datei-Locking wird nicht unterstützt

Meta-DOS kann von Entwicklern direkt von Atari bezogen werden und soll künftig allen Arten von GEMDOS-Treibern (CD-ROM, fremde Dateisysteme, Netzwerke) als Basis dienen.

Bei Interesse sollten man sich beim Softwaresupport der jeweiligen Atari-Niederlassung melden. Die Linzenzbedingungen entsprechen denen von GDOS (einmalige Zahlung einer eher symbolischen Summe).

GEMDOS mit Multitasking

Aufgrund der an sich günstigen Ausgangsbasis haben sich auch unabhängige Entwickler der Weiterentwicklung vom GEMDOS angenommen. Schon seit langer Zeit gibt es "Micro-RTX" von Beckemeyer Development Tools, das GEMDOS in ein Echzeit-Multitasking-System (also mit möglichst kurzen und garantierten Prozeßwechselzeiten) zu verwandeln sucht.

Ein anderer Versuch ist "MiNT" ("MiNT is Not TOS") von Eric Smith, das als "offenes" Projekt (jeder kann die C-Quelltexte bekommen) entsteht. "MiNT" versucht, möglichst viele Eigenschaften von UNIX nachzubilden (Signale, Pipes etc).

GEMDOS-Bindings

Ebenso wie BIOS und XBIOS empfängt auch GEMDOS seine Parameter auf dem Stack. Dabei wird das letzte Argument aus der Parameterliste als erstes auf den Stack gelegt.

Schließlich wird durch die Anweisung "TRAP #1" der GEMDOS-Trap-Dispatcher aktiviert.

Funktionsergebnisse werden im Prozessorregister D0 zurückgeliefert. GEMDOS zerstört die Register D0–D2 und A0–A2. Nur bei den Registern D3–D7 und A3–A7 darf man sich darauf verlassen, daß sie nicht verändert werden.

Vorsicht bei Aufrufen mit fehlerhaften Parametern! GEMDOS ist mit Sicherheit der instabilste Teil von TOS und ist äußerst leicht mit fehlerhaften Eingabedaten zu verwirren ("Garbage in -> Crash!").

Fehlermeldungen

GEMDOS-Fehlermeldungen sind negative LONG-Werte zwischen –32 und –127. Bei den einzelnen Funktionen sind mögliche Fehlercodes angegeben.

Man sollte sich allerdings *nicht* darauf verlassen, daß auch wirklich nur diese Fehlermeldungen auftreten können! Neben den folgenden Fehlermeldungen können auch alle BIOS-Return-Codes zurückgeliefert werden!

0: E_OK ("OK (no error)")

Funktion erfolgreich ausgeführt (kein Fehler aufgetreten).

–32: EINVFN ("Invalid function number")

Unbekannte Funktionsnummer. Man erhält diese Meldung, wenn man eine undefinierte GEMDOS-Funktion aufruft (unter MS-DOS: Fehlercode 1).

–33: EFILNF ("File not found")

Datei nicht gefunden (unter MS-DOS: Fehlercode 2).

–34: EPTHNF ("Path not found")

Angesprochener Ordner nicht gefunden (unter MS-DOS: Fehlercode 3).

–35: ENHNDL ("Handle pool exhausted")

Keine Dateihandles mehr (zu viele Dateien geöffnet, unter MS-DOS: Fehlercode 4).

–36: EACCDN ("Access denied")

Zugriff nicht erlaubt (unter MS-DOS: Fehlercode 5).

–37: EIHNDL ("Invalid handle")

Das Dateihandle war nicht korrekt (unter MS-DOS: Fehlercode 6).

–39: ENSMEM ("Insufficient memory")

Nicht genügend Speicher vorhanden (unter MS-DOS: Fehlercode 8).

–40: EIMBA ("Invalid memory block address")

Die Adresse des Speicherblocks war nicht gültig (unter MS-DOS: Fehlercode 9).

–46: EDRIVE ("Invalid drive specification")

Die Laufwerksbezeichnung war ungültig (unter MS-DOS: Fehlercode 15).

–48: ENSAME ("Not the same drive")

Dateien sind auf verschiedenen logischen Laufwerken.

–49: ENMFIL ("No more files")

Es können keine Dateien mehr geöffnet werden (unter MS-DOS: Fehlercode 18).

–58: ELOCKED ("Record is locked")

Es wurde versucht, auf einen gelockten Bereich einer Datei zuzugreifen (nur im Zusammenhang mit einer Netzwerk-GEMDOS-Version).

–59: ENSLOCK ("No such lock")

Es wurde versucht, einen nicht existierenden Lock (falscher Offset oder/und falsche Länge) zu entfernen (nur im Zusammenhang mit einer Netzwerk-GEMDOS-Version).

–64: ERANGE ("Range error")

Dateizeiger in ungültigem Bereich.

–65: EINTRN ("GEMDOS internal error")

Interner Fehler im GEMDOS (hoffentlich passiert's nie!).

–66: EPLFMT ("Invalid executable file format")

Das angesprochene Programm hat nicht das korrekte Format, um geladen zu werden.

–67: EGSBF ("Memory block growth failure")

Es wurde versucht, einen allozierten Speicherblock zu vergrößern.

GEMDOS-Referenz

Cauxin (GEMDOS 3) – Read character from standard AUX:

Mit dieser Funktion kann man Zeichen von Standardkanal 2 (das ist normalerweise die serielle Schnittstelle) empfangen (dabei wird gegebenenfalls so lange gewartet, bis ein Zeichen an der Schnittstelle vollständig eingetroffen ist!).

Atari empfiehlt übrigens zu diesem Zweck die Verwendung der entsprechenden BIOS-Funktion.

Deklaration in C:

```
WORD Cauxin (void);
```

Aufruf in Assembler:

```
move.w    #3,-(sp)   ; Offset 0
trap      #1
addq.l    #2,sp
```

Parameter:

Cauxin(): Empfangenes Zeichen

Bemerkungen
Bei Ein-/Ausgabeumlenkung wird zur Zeit beim Dateiende ein undefiniertes Zeichen zurückgegeben.

Cauxis (GEMDOS 18) – Check status of standard AUX: input

Mit dieser Funktion können Sie feststellen, ob von der seriellen Schnittstelle (d.h. vom Standardkanal 2) ein Zeichen empfangen werden kann.

Atari empfiehlt die Verwendung der entsprechenden BIOS-Funktion.

Deklaration in C:

```
WORD Cauxis (void);
```

Aufruf in Assembler:

```
move.w    #$12,-(sp)      ; Offset 0
trap      #1
addq.l    #2,sp
```

Parameter:

Cauxis():	–1:	Zeichen ist verfügbar
	0:	Es liegt kein Zeichen vor

Bemerkungen
Arbeitet bei Ein-/Ausgabeumlenkung erst ab GEMDOS-Version 0.15 korrekt.

Cauxos (GEMDOS 19) – Check status of standard AUX: output

Prüft, ob über die serielle Schnittstelle (Standardkanal 2) ein Zeichen ausgegeben werden kann oder nicht.

Atari empfiehlt die Verwendung der entsprechenden BIOS-Funktion.

Deklaration in C:

```
WORD Cauxos (void);
```

Aufruf in Assembler:

```
move.w    #$13,-(sp)      ; Offset 0
trap      #1
addq.l    #2,sp
```

Parameter:

Cauxos():	–1:	Zeichen kann ausgegeben werden
	0:	Zeichen kann nicht ausgegeben werden

Cauxout (GEMDOS 4) – Write character to standard AUX:

Zeichen wird auf Standardkanal 2 (dem seriellen Port) ausgegeben.

Atari empfiehlt die Verwendung der entsprechenden BIOS-Funktion.

Deklaration in C:

```
void Cauxout (WORD c);
```

Aufruf in Assembler:

```
move.w    c,-(sp)    ; Offset 2
move.w    #4,-(sp)   ; Offset 0
trap      #1
addq.l    #4,sp
```

Parameter:

c: Auszugebendes Zeichen (ASCII-Code in Bits 0..7, alle anderen Bits auf 0 setzen).

Bemerkungen
Arbeitet bei Ein-/Ausgabeumlenkung erst ab GEMDOS-Version 0.15 korrekt.

Cconin (GEMDOS 1) – Read character from standard input

Ein Zeichen wird von stdin (also Standardkanal 0) gelesen (sobald eines vorliegt). Ist stdin die Tastatur, dann wird in den Bits 0..7 der ASCII-Code der gedrückten Taste und in den Bits 16..23 der Scan-Code zurückgeliefert.

Durch Setzen von Bit 3 der Systemvariablen "conterm" kann man veranlassen, daß in den Bits 24..31 der Wert von "Kbshift()" zurückgegeben wird.

Deklaration in C:

```
LONG Cconin (void);
```

Aufruf in Assembler:

```
move.w    #1,-(sp)   ; Offset 0
trap      #1
addq.l    #2,sp
```

Parameter:

Cconin(): eingelesenes Zeichen

Bemerkungen

Man hat weder die Möglichkeit festzustellen, ob I/O-Redirection stattfindet, noch die, das Dateiende zu erkennen. Daher definieren viele C-Bibliotheken analog zu MS-DOS den ASCII-Code 26 (also <Ctrl><Z>) als Zeichen für das Dateiende.

Cconis (GEMDOS 11) – Check status of standard input

Diese Funktion überprüft, ob von stdin (Standardkanal 0, normalerweise die Tastatur) ein Zeichen eingelesen werden kann.

Deklaration in C:

```
WORD Cconis (void);
```

Aufruf in Assembler:

```
move.w    #$B,-(sp) ; Offset 0
trap      #1
addq.l    #2,sp
```

Parameter:

Cconis():	0:	Zeichen nicht verfügbar
	−1:	Zeichen verfügbar

Cconos (GEMDOS 16) – Check status of standard output

Liefert den Ausgabestatus von stdout (Standardkanal 1, normalerweise der Bildschirm).

Deklaration in C:

```
WORD Cconos (void);
```

Aufruf in Assembler:

```
move.w    #$10,-(sp)      ; Offset 0
trap      #1
addq.l    #2,sp
```

Parameter:

Cconos(): 1 (o.k.) oder 0 (Es kann kein Zeichen ausgegeben werden)

Bemerkungen

Arbeitet bei Ein-/Ausgabeumlenkung erst ab GEMDOS-Version 0.15 korrekt.

Cconout (GEMDOS 2) – Write character to standard output

Ein Zeichen wird auf stdout (Handle 1, normalerweise der Bildschirm) ausgegeben.

Deklaration in C:

```
void Cconout (WORD c);
```

Aufruf in Assembler:

```
move.w    c,-(sp)    ; Offset 2
move.w    #2,-(sp)   ; Offset 0
trap      #1
addq.l    #4,sp
```

Parameter:

c: Auszugebendes Zeichen als 16-Bit-Wort (ASCII-Code in Bits 0..7, alle anderen Bits müssen 0 sein)

Bemerkungen
Arbeitet bei Ein-/Ausgabeumlenkung erst ab GEMDOS-Version 0.15 korrekt.

Cconrs (GEMDOS 10) – Read edited data from standard input

Mit dieser Funktion kann eine ganze Zeichenkette vom Standardeingabekanal, im Normalfall also der Tastatur, gelesen werden.

Dazu schreibt man in das erste Element der LINE-Struktur die Anzahl der einzulesenden Zeichen (vermindert um 1)."Cconrs()" bricht die Eingabe in dem Moment ab, in dem der Anwender "Return" drückt oder die Maximallänge überschritten wird.

Ein eventuelles Dateiende wird nicht erkannt! Umlaute (genauer: Zeichen mit ASCII-Code größer als 127) werden erst ab GEMDOS-Version 0.15 korrekt behandelt.

Eingabefunktionen:

<Return>, CTRL-J	Ende der Eingabe
<Backspace>, CTRL-H	Letztes Zeichen löschen
CTRL-U, CTRL-X	Ganze Zeile löschen
CTRL-R	Zeile neu eingeben
CTRL-C	Programm abbrechen (siehe dazu auch "etv_term")

Deklaration in C:

```
typedef struct
{
     BYTE maxlen;        /* maximale Länge der Eingabe */
     BYTE actuallen;     /* tatsächliche Länge, Offset 1 */
     CHAR buffer[255];   /* eingelesene Zeichen, Offset 2 */
} LINE;

void Cconrs (LINE *buf);
```

Aufruf in Assembler:

```
pea       buf       ; Offset 2
move.w    #$A,-(sp) ; Offset 0
trap      #1
addq.l    #6,sp
```

Parameter:

buf->maxlen:	Anzahl der einzulesenden Zeichen (-1)
buf->actuallen:	Tatsächlich gelesene Zeichen
buf->buffer:	Die eingelesene Zeichenkette

Bemerkungen
Gelesene Zeichen werden selbst dann auf dem Bildschirm ausgegeben, wenn die Standardausgabe auf eine Datei umgelenkt worden war.

Cconws (GEMDOS 9) – Write string to standard output

Mit dieser Funktion kann eine ganze Zeichenkette auf stdout (also normalerweise dem Bildschirm) ausgegeben werden. Als Endmarke für den String wird – wie gewohnt – die 0 verwendet.

Deklaration in C:

```
void Cconws (const char *str);
```

Aufruf in Assembler:

```
pea       str       ; Offset 2
move.w    #9,-(sp)  ; Offset 0
trap      #1
addq.l    #6,sp
```

Parameter:

str: auszugebende Zeichenkette (muß mit 0 abgeschlossen sein)

Bemerkungen

Einige Dokumentationen (auch zurückliegende Auflagen des Profibuchs) geben als Rückgabewerte die Anzahl der ausgegebenen Zeichen an. Dieses Verhalten ist jedoch offiziell *nicht* dokumentiert und sollte daher nicht ausgenutzt werden!

Cnecin (GEMDOS 8) – Read character from standard input, no echo

Entspricht "Crawcin()" mit der Ausnahme, daß Steuerzeichen (<Ctrl><S> Bildschirmausgabe stoppen, <Ctrl><Q> Bildschirmausgabe fortsetzen, <Ctrl><CC Abbruch usw.) korrekt interpretiert werden.

Deklaration in C:

```
LONG Cnecin (void);
```

Aufruf in Assembler:

```
move.w    #8,-(sp)   ; Offset 0
trap      #1
addq.l    #2,sp
```

Parameter:

Cnecin(): das eingelesene Zeichen

Cprnos (GEMDOS 17) – Check status of standard PRN:

Liefert den Ausgabestatus der parallelen Schnittstelle (Standardkanal 3).

Es empfiehlt sich dringend, diese Funktion vor jeder Druckausgabe zu nutzen, da bis zum Erkennen des Drucker-"Timeouts" eine halbe Ewigkeit (ungefähr eine halbe Minute) vergeht und die Geduld der Anwender meist sehr eingeschränkt ist.

Deklaration in C:

```
WORD Cprnos (void);
```

Aufruf in Assembler:

```
move.w    #$11,-(sp)      ; Offset 0
trap      #1
addq.l    #2,sp
```

Parameter:

Cprnos():	0:	Drucker nicht empfangsbereit
	–1:	Drucker empfangsbereit

Cprnout (GEMDOS 5) – Write character to standard PRN:

Ein Zeichen wird auf Standardkanal 3 (dem Drucker) ausgegeben. Dabei werden die über "Setprt()" vorgenommenen Einstellungen (die man zum Beispiel über das Kontrollfeld vornehmen kann) aufgrund eines Fehlers in allen bekannten TOS-Versionen *nicht* berücksichtigt.

Es empfiehlt sich, vorher mittels "Cprnos()" festzustellen, ob der Drucker überhaupt Zeichen empfangen kann.

Deklaration in C:

```
WORD Cprnout (WORD c);
```

Aufruf in Assembler:

```
move.w    c,-(sp)    ; Offset 2
move.w    #5,-(sp)   ; Offset 0
trap      #1
addq.l    #4,sp
```

Parameter:

c:	auszugebendes Zeichen in Bits 0..7, alle anderen Bits müssen auf 0 gesetzt sein
Cprnout():	Es wird der Rückgabewert der entsprechenden BIOS-Funktion zurückgeliefert (0: o.k., sonst: Fehler). Der ist allerdings nur dann definiert, wenn tatsächlich auf die parallele Schnittstelle ausgegeben wurde!

Bemerkungen
Arbeitet bei Ein-/Ausgabeumlenkung erst ab GEMDOS-Version 0.15 korrekt.

Crawcin (GEMDOS 7) – Raw input from standard input

Ein Zeichen wird von stdin (der Tastatur) gelesen, ohne daß es dabei sofort auf dem Bildschirm ausgegeben wird. <Ctrl><C> wird dabei ignoriert.

Deklaration in C:

```
LONG Crawcin (void);
```

Aufruf in Assembler:

```
move.w    #7,-(sp)   ; Offset 0
trap      #1
addq.l    #2,sp
```

Parameter:

Crawcin(): das eingelesene Zeichen

Bemerkungen

Bei Ein-/Ausgabeumlenkung wird zur Zeit beim Dateiende ein undefiniertes Zeichen zurückgegeben.

Crawio (GEMDOS 6) – Raw I/O to standard input/output

Dies ist eine Vielzweckfunktion zur Ein- und Ausgabe auf stdin (Tastatur) und stdout (Bildschirm). <Ctrl><C> wird dabei ignoriert.

Deklaration in C:

```
LONG Crawio (WORD w);
```

Aufruf in Assembler:

```
move.w    w,-(sp)    ; Offset 2
move.w    #6,-(sp)   ; Offset 0
trap      #1
addq.l    #4,sp
```

Parameter:

w:	$FF:	Zeichen von stdin einlesen (entspricht "Cconin()")
	sonst:	Zeichen auf stdout ausgeben (entspricht "Cconout()")
Crawio():	(für w=$FF) das eingelesene Zeichen wie bei "Cconin()" oder 0 (kein Zeichen verfügbar).	

Bemerkungen
Arbeitet bei Ein-/Ausgabeumlenkung erst ab GEMDOS-Version 0.15 korrekt.

Dcreate (GEMDOS 57) – Create directory

Legt neue Ordner (Subdirectories) an.

Deklaration in C:

```
WORD Dcreate (const char *pathname);
```

Aufruf in Assembler:

```
pea       pathname        ; Offset 2
move.w    #$39,-(sp)      ; Offset 0
trap      #1
addq.l    #6,sp
```

Parameter:

pathname:	Zeiger auf einen gültigen Ordnernamen	
Dcreate():	einige mögliche Return-Werte:	
	OK (0):	alles in Ordnung
	EPTHNF (–34):	Ordner nicht gefunden
	EACCDN (–36):	Zugriff verwehrt

Bemerkungen

Vor GEMDOS 0.15 machte "Dcreate()" praktisch keine Fehlerbehandlung. Dadurch konnte sogar im schlimmsten Fall das Dateisystem in Mitleidenschaft gezogen werden. Ebensowenig wurde vorher geprüft, ob es nicht bereits eine Datei mit gleichem Namen gibt (die Datei wurde dann einfach gelöscht).

Ddelete (GEMDOS 58) – Delete directory

Löscht einen Ordner (der dazu leer sein muß).

Deklaration in C:

```
WORD Ddelete (const char *pathname);
```

Aufruf in Assembler:

```
pea       pathname        ; Offset 2
move.w    #$3A,-(sp)      ; Offset 0
trap      #1
addq.l    #6,sp
```

Parameter:

pathname:	Zeiger auf den entsprechenden Ordnernamen.	
Ddelete():	einige mögliche Return-Werte:	
	OK (0):	alles in Ordnung
	EPTHNF (–34):	Ordner nicht gefunden
	EACCDN (–36):	Zugriff verwehrt (Ordner enthält Dateien, die vorher einzeln gelöscht werden müssen)
	EINTRN (–65):	Interner Fehler (?)

Bemerkungen

Vor GEMDOS 0.15 funktionierte ein "Dcreate()" unmittelbar gefolgt von einem "Ddelete()" nicht (erst ein weiterer Aufruf von "Ddelete()" führte zum gewünschten Ergebnis).

Dfree (GEMDOS 54) – Get drive free space

Mit "Dfree()" kann man diverse Informationen über ein bestimmtes logisches Laufwerk erfragen. Die Informationen werden in einer DISKINFO-Struktur abgelegt:

```
typedef struct
{
     LONG b_free;   /* Anzahl der freien Cluster */
     LONG b_total;  /* Gesamtzahl der Cluster */
     LONG b_secsiz; /* Bytes pro Sektor */
     LONG b_clsiz;  /* Sektoren pro Cluster /*
} DISKINFO;
```

Deklaration in C:

```
LONG Dfree (DISKINFO *buf, WORD drv);
```

Aufruf in Assembler:

```
move.w    drv,-(sp)       ; Offset 6
pea       buf             ; Offset 2
move.w    #$36,-(sp)      ; Offset 0
trap      #1
addq.l    #8,sp
```

Parameter:

drv:	0:	aktuelles Laufwerk
	1:	A:
	2:	B: (usw.)
buf:	Anfangsadresse des DISKINFO-Puffers	
Dfree():	einige mögliche Return-Werte:	
	OK (0):	alles in Ordnung

Bemerkungen

Auf GEMDOS-Versionen vor 0.15 ("Rainbow-TOS") war diese Funktion lächerlich langsam.

Dgetdrv (GEMDOS 25) – Get default drive

Liefert die Nummer des aktuellen Laufwerks.

Deklaration in C:

```
WORD Dgetdrv (void);
```

Aufruf in Assembler:

```
move.w    #$19,-(sp)      ; Offset 0
trap      #1
addq.l    #2,sp
```

Parameter:

Dgetdrv():	0:	Laufwerk A:
	1:	Laufwerk B:
	2:	Laufwerk C:
	usw.	

Dgetpath (GEMDOS 71) – Get current directory

Liefert den Pfadnamen für das aktuelle Teilverzeichnis.

Es ist eine konzeptionelle Schwäche des GEMDOS, daß die maximale Länge von Verzeichnisnamen nicht begrenzt ist, aber auch nicht vorher abgefragt werden kann.

Ursache für dieses merkwürdige Verhalten ist, daß GEMDOS erst bei "Dgetpath()" den vollständigen Pfad zusammenkopiert. Es bleibt also nichts anderes übrig, als "Dgetpath()" einen "ausreichend" langen Puffer zu übergeben.

Ein sinnvoller Wert dürfte beispielsweise 256 Zeichen sein, da das Desktop selbst auch keine längeren Pfade verwalten kann.

Deklaration in C:

```
WORD Dgetpath (char *buf, WORD driveno);
```

Aufruf in Assembler:

```
move.w    driveno,-(sp)   ; Offset 6
pea       buf             ; Offset 2
move.w    #$47,-(sp)      ; Offset 0
trap      #1
addq.l    #8,sp
```

Parameter:

buf:	Zeiger auf einen Puffer, der nach dem Aufruf den aktuellen Zugriffspfad enthält	
drv:	0:	aktuelles Laufwerk
	1:	A:
	2:	B:
	usw.	
Dgetpath():	einige mögliche Return-Werte:	
	OK (0):	alles in Ordnung
	EDRIVE (–46):	Falsche Laufwerksnummer

Dsetdrv (GEMDOS 14) – Set default drive

Die Nummer des aktuellen Laufwerks wird gesetzt.

Deklaration in C:

```
LONG Dsetdrv (WORD drv);
```

Aufruf in Assembler:

```
move.w    drv,-(sp) ; Offset 2
move.w    #$E,-(sp) ; Offset 0
trap      #1
addq.l    #4,sp
```

Parameter:

drv:	Nummer des neuen aktuellen Laufwerks (0: A, 1: B...)
Dsetdrv():	Bitvektor mit den vorhandenen Laufwerken (Bit 0: A:,...)

Bemerkungen
Vorsicht! Bei vielen Compilern ist der Return-Wert dieser Funktion fälschlich als "WORD" deklariert.

Und noch einmal Vorsicht! GEMDOS macht bei dieser Funktion überhaupt keine Plausibilitätstests, so daß eine falsche Laufwerksnummer (außerhalb des erlaubten Bereichs) zu schweren Schäden in GEMDOS-internen Strukturen führen kann!

Einen Bitvektor mit den von GEMDOS unterstützen Laufwerken beschafft man sich am besten mit:

```
Dsetdrv (Dgetdrv ());
```

Zur Zeit ist der zurückgelieferte Bitvektor das Resultat des BIOS-Aufrufs "Drvmap()". Dies ist jedoch *nicht* dokumentiert und könnte sich ohne weiteres eines Tages ändern.

Dsetpath (GEMDOS 59) – Set current directory

Setzt den neuen aktuellen Zugriffspfad für das aktuelle Laufwerk. GEMDOS merkt sich für jedes logische Laufwerk einen aktuellen Zugriffspfad. "Dsetpath()" sollte nur für das aktuelle Laufwerk benutzt werden. Zum Setzen des aktuellen Pfades auf einem anderen Laufwerk sollte man etwa folgende Sequenz benutzen:

1. Aktuelles Laufwerk abfragen
2. Gewünschtes Laufwerk setzen
3. Pfad für das Laufwerk setzen
4. Gemerktes Laufwerk wieder zum aktuellen Laufwerk machen

Deklaration in C:

```
WORD Dsetpath (const char *path);
```

Aufruf in Assembler:

```
pea       path          ; Offset 2
move.w    #$3B,-(sp)    ; Offset 0
trap      #1
addq.l    #6,sp
```

Parameter:

path:	Pfadname für den neuen aktuellen Zugriffspfad	
Dsetpath():	einige mögliche Return-Werte:	
	OK (0):	alles in Ordnung
	EPTHNF (–34):	Ordner nicht gefunden

Bemerkungen

Auf älteren GEMDOS-Versionen führt das häufige Setzen nicht existenter Pfade zu GEMDOS-internen Störungen (Verlust von Pfadkontrollstrukturen). Mögliche Folge ist, daß selbst

```
Dsetpath ("\\");    /* Pfad auf Wurzelverzeichnis setzen */
```

zu einem Fehler führt.

Fattrib (GEMDOS 67) – Get/Set file attributes

Liest oder setzt die Dateiattribute einer Datei.

Deklaration in C:

```
WORD Fattrib (const char *fname, WORD wflag, WORD attribs);
```

Aufruf in Assembler:

```
move.w    attribs,-(sp)   ; Offset 8
move.w    wflag,-(sp)     ; Offset 6
pea       fname           ; Offset 2
move.w    #$43,-(sp)      ; Offset 0
trap      #1
lea       $A(sp),sp
```

Parameter:

fname: Dateiname der betreffenden Datei (ggfs. mit Zugriffspfad, sonst im aktuellen Teilverzeichnis)

wflag:
- 0: die aktuellen Attribute werden nur ausgelesen und zurückgeliefert (attribs wird ignoriert)
- 1: die Dateiattribute der angegebenen Datei werden auf den in attribs angegebenen Wert gesetzt

attribs: Dateiattribute:
- Bit 0: Datei schreibgeschützt
- Bit 1: Datei versteckt ("hidden")
- Bit 2: Systemdatei
- Bit 3: Diskettenname
- Bit 4: Teilverzeichnis (Ordner)
- Bit 5: Archiv-Bit

Fattrib(): die bisherigen Dateiattribute oder:
- EFILNF (–33): Datei nicht gefunden
- EPTHNF (–34): Ordner nicht gefunden

oder eine andere Fehlermeldung

Fclose (GEMDOS 62) – Close file

Schließt eine mit "Fopen()" oder "Fcreate()" geöffnete Datei.

Deklaration in C:

```
WORD Fclose (WORD handle);
```

Aufruf in Assembler:

```
move.w    handle,-(sp)    ; Offset 2
move.w    #$3E,-(sp)      ; Offset 0
trap      #1
addq.l    #4,sp
```

Parameter:

handle:	Kennung der zu schließenden Datei	
Fclose():	einige mögliche Return-Werte:	
	OK (0):	alles in Ordnung
	EIHNDL (–37):	Dateikennung war falsch!

Bemerkungen

Was beim Schließen von Standardkanälen passiert, ist erst ab GEMDOS 0.15 definiert: es wird wieder der ursprüngliche zeichenorientierte Kanal eingesetzt.

Fcreate (GEMDOS 60) – Create file

Unter GEMDOS gibt es zwei Möglichkeiten, eine Datei zum Schreiben zu öffnen: mit "Fcreate()" kann man eine völlig neue Datei erzeugen. Existiert dabei bereits eine Datei mit diesem Namen, wird sie dabei überschrieben. Mit "Fopen()" dagegen kann man nur bereits existierende Dateien öffnen.

Die Moral von der Geschicht': Ausgabedateien öffnet man mit "Fopen()" nicht!

Deklaration in C:

```
WORD Fcreate (const char *fname, WORD attribs);
```

Aufruf in Assembler:

```
move.w    attribs,-(sp)   ; Offset 6
pea       fname           ; Offset 2
move.w    #$3C,-(sp)      ; Offset 0
trap      #1
addq.l    #8,sp
```

Parameter:

fname:	Zeiger auf den Dateinamen (dabei kann auch der Zugriffspfad angegeben werden, ansonsten wird das aktuelle Teilverzeichnis benutzt).
attribs:	Bit 0: Datei schreibgeschützt Bit 1: Versteckte Datei Bit 2: Systemdatei Bit 3: Diskettenname (sollte auf jeder Diskette nur einmal vorhanden sein!)
Fcreate():	Kennung der Datei oder EPTHNF (–34): Zugriffspfad war nicht korrekt ENHNDL (–35): Keine freien Dateikennungen (handles) mehr EACCDN (-36): Kein Schreibzugriff möglich oder eine andere Fehlermeldung

Bemerkungen

Vorsicht bei der Benutzung von Wildcards: eine Dateiangabe wie "TEST*.*" wird von GEMDOS intern zunächst zu "TEST????.???" expandiert. Unter diesem (nicht erlaubten) Namen wird die Datei dann auch tatsächlich angelegt!

Bei Verwendung eines netzwerkfähigen GEMDOS (Cookie "_FLK" testen!) ist die Datei exklusiv für den anlegenden Prozeß zugänglich.

Den Diskettennamen kann man folgendermaßen neu setzen:

```
IF GEMDOS-Version < 0.15
     IF Volume-Label schon vorhanden
          Datei gleichen Namens mit Fcreate() anlegen, mit Fclose()
          schließen und mit Fdelete() wieder löschen
     ENDIF
ENDIF
Diskettennamen mit Fcreate() anlegen und das zurückgelieferte
Handle mit Fclose() wieder freigeben.
```

Fdatime (GEMDOS 87) – Get/Set file timestamp

Setzt oder ermittelt Uhrzeit und Datum der Erstellung einer Datei (die man zuvor mit "Fopen()" oder "Fcreate()" geöffnet hat). Zur Zeitübergabe wird folgende Struktur benutzt:

```
typedef struct
{
     UWORD time;     /* Zeit wie in "Tgettime" */
     UWORD date;     /* Datum wie in "Tgetdate" */
} DOSTIME;
```

Deklaration in C:

```
void Fdatime (DOSTIME *timeptr, WORD handle, WORD wflag);
```

Aufruf in Assembler:

```
move.w    wflag,-(sp)     ; Offset 8
move.w    handle,-(sp)    ; Offset 6
pea       timeptr         ; Offset 2
move.w    #$57,-(sp)      ; Offset 0
trap      #1
lea       $A(sp),sp
```

Parameter:

handle:	Dateikennung der betreffenden Datei	
timeptr:	Zeiger auf eine DOSTIME-Struktur	
wflag:	0:	Aktuelle Werte in DOSTIME-Struktur übertragen
	1:	Werte aus DOSTIME-Struktur in Directory eintragen

Bemerkungen

Wegen einiger Fehler in alten GEMDOS-Versionen sollte man "Fdatime()" stets auf folgende Art und Weise benutzen:

1. Datei öffnen
2. "Fdatime()" aufrufen
3. Datei sofort wieder schließen

Fdelete (GEMDOS 65) – Delete file

Löscht die angegebene Datei.

Deklaration in C:

```
WORD Fdelete (const char *fname);
```

Aufruf in Assembler:

```
pea       fname          ; Offset 2
move.w    #$41,-(sp)     ; Offset 0
trap      #1
addq.l    #6,sp
```

Parameter:

fname:	Name der betreffenden Datei (evtl. mit Zugriffspfad)	
Fdelete():	einige mögliche Return-Werte:	
	OK (0):	alles in Ordnung
	EFILNF (–33):	Datei nicht gefunden
	EACCDN (–36):	Zugriff verwehrt (Datei schreibgeschützt?)

Bemerkungen

Es ist nicht möglich, geöffnete Dateien zu löschen – es sei denn, man selbst ist der einzige Prozeß, der die Datei geöffnet hat. In diesem Fall wird die Datei erst geschlossen und dann gelöscht. Leider tritt dabei ein Fehler auf: Die Dateikennung wird nicht wieder freigegeben. *Also:* geöffnete Dateien nicht löschen!

Fdup (GEMDOS 69) – Duplicate file handle

Für den angegebenen Standardkanal (0..5) wird eine zweite Kennung (handle) geliefert (siehe auch "Fforce()").

In allen bekannten GEMDOS-Versionen (bis einschließlich 0.19) treten Fehler auf, wenn nach "Fdup()" noch ein Programm mittels "Pexec()" gestartet wird.

Deklaration in C:

```
WORD Fdup (WORD handle);
```

Aufruf in Assembler:

```
move.w    handle,-(sp)    ; Offset 2
move.w    #$45,-(sp)      ; Offset 0
trap      #1
addq.l    #4,sp
```

Parameter:

stdhandle: Nummer des Standardkanals:
- 0: Tastatur (stdin)
- 1: Bildschirm (stdout)
- 2: Serielle Schnittstelle (stdaux)
- 3: Parallele Schnittstelle (stdprn)

Fdup(): Neue Kennung oder:
- ENHNDL (–35): keine freien Kennungen mehr
- EIHNDL (–37): falsche Kennung (handle)
- oder eine andere Fehlermeldung

Fforce (GEMDOS 70) – Force file handle

Der Schlüssel zur "Input/Output-Redirection": biegt einen Standardkanal (0..5) auf einen beliebigen anderen Kanal um.

Deklaration in C:

```
WORD Fforce (WORD stdh, WORD nonstdh);
```

Aufruf in Assembler:

```
move.w    nonstdh,-(sp)   ; Offset 4
move.w    stdh,-(sp)      ; Offset 2
move.w    #$46,-(sp)      ; Offset 0
trap      #1
addq.l    #6,sp
```

Parameter:

stdh: Nummer des umzuleitenden Kanals:
- 0: Tastatur (stdin)
- 1: Bildschirm (stdout)
- 2: Serielle Schnittstelle (stdaux)
- 3: Parallele Schnittstelle (stdprn)

nonstdh: Nummer des ersetzenden Kanals (muß vorher mit "Fopen()" oder "Fcreate()" geöffnet worden sein. Mit "Fdup()" kann man ein "nonstdhandle" für einen Standardkanal besorgen). Mit "Fforce (1, Fdup (3))" könnte man also beispielsweise alle Bildschirmausgaben an den Drucker umleiten!

Fforce(): einige mögliche Return-Werte:
- OK (0): alles in Ordnung
- EIHNDL (–37): Falsche Kennung (handle)

Fgetdta (GEMDOS 47) – Get DTA

Liefert die Anfangsadresse der aktuellen DTA (Siehe auch unter "Fsfirst()" und "Fsnext()"):

```
typedef struct
{
     BYTE d_reserved[21];     /* fürs GEMDOS reserviert */
     BYTE d_attrib;           /* Dateiattribut */
     UWORD d_time;            /* Uhrzeit */
     UWORD d_date;            /* Datum */
     LONG d_length;           /* Dateilänge */
     char d_fname[14];        /* Dateiname */
} DTA;
```

Deklaration in C:

```
DTA *Fgetdta (void);
```

Aufruf in Assembler:

```
move.w    #$2F,-(sp)      ; Offset 0
trap      #1
addq.l    #2,sp
```

Parameter:

Fgetdta(): Zeiger auf die aktuelle DTA

Flock (GEMDOS 92) – Lock file

Diese Funktion ist nur unter einem netzwerkfähigen GEMDOS ("_FLK"-Cookie testen!) verfügbar.

Sie dient dazu, Teile von Dateien gegen den Zugriff von anderen Prozessen (Benutzern) zu schützen.

Deklaration in C:

```
LONG Flock (WORD handle, WORD mode, LONG start, LONG length);
```

Aufruf in Assembler:

```
move.l    length,-(sp)    ; Offset 10
move.l    start,-(sp)     ; Offset 6
move.w    mode,-(sp)      ; Offset 4
move.w    handle,-(sp)    ; Offset 2
move.w    #$5C,-(sp)      ; Offset 0
trap      #1
lea       $C(sp),sp
```

Parameter:

handle:	Kennung der betreffenden Datei	
mode:	0:	"length" Bytes ab Position "start" blockieren ("locken")
	1:	Blockierung aufheben. "length" und "start" müssen exakt zu einer existierenden Blockierung passen!
start:	Startposition in der Datei	
length:	Anzahl der Bytes	
Flock():	einige mögliche Return-Werte:	
	OK (0):	alles in Ordnung
	ELOCKED (–58):	Blockierung (Lock) bereits vorhanden (Dateiausschnitt ist bereits blockiert bzw. überlappt einen blockierten Ausschnitt.)
	ENSLOCK (–59):	Entsprechende Blockierung existiert nicht (Es wurde versucht, eine nicht vorhandene Blockierung aufzuheben.)

Bemerkungen

Siehe Hinweise zum Atari-Netzwerkstandard in der GEMDOS-Einführung.

Fopen (GEMDOS 61) – Open file

Öffnet (existierende) Dateien. Dazu gehören auch die folgenden zeichenorientierten Geräte:

PRN: Parallele Schnittstelle (liefert handle –3)
AUX: Serielle Schnittstelle (liefert handle –2)
CON: Konsole (liefert handle –1)

Für die wichtigsten Geräte gibt es außerdem auch schon Standardkanalnummern, so daß man "Fopen()" überhaupt nicht aufzurufen braucht:

0: Tastatur (stdin)
1: Bildschirm (stdout)
2: Serielle Schnittstelle (stdaux)
3: Parallele Schnittstelle (stdprn)

Laut Atari sind die Standardkanalnummern 4 und 5 auch reserviert – wofür ist zur Zeit nicht bekannt. "Normale" Kanalnummern für Dateien beginnen bei 6.

Deklaration in C:

```
WORD Fopen (const char *fname, WORD mode);
```

Aufruf in Assembler:

```
move.w    mode,-(sp)      ; Offset 6
pea       fname           ; Offset 2
move.w    #$3D,-(sp)      ; Offset 0
trap      #1
addq.l    #8,sp
```

Parameter:

fname:	Dateiname der zu öffnenden Datei (evtl. mit Zugriffspfad, sonst wird im aktuellen Teilverzeichnis gesucht)	
mode:	0:	nur lesen
	1:	nur schreiben
	2:	lesen und schreiben
Fopen():	Entweder die Datei-Kennung (handle) oder	
	EFILFN (–33):	Datei nicht gefunden

ENHDNL (–35): keine freien Datei-Kennungen
EACCDN (–36): Zugriff nicht erlaubt
oder eine andere Fehlernummer

Bemerkungen
Unter einem GEMDOS mit File-Locking-Erweiterungen ("_FLK"-Cookie testen!) gelten zusätzlich folgende Definitionen:

Der mode-Parameter ist als Bitvektor mit folgender Belegung definiert:

Bit 7: Vererbungs-Flag
Bit 4..6: Sharing-Modus
Bit 3: reserviert (auf Null setzen)
Bit 0..2: Zugriffsmodus

Der Zugriffsmodus entspricht dem normalen Modusparameter unter Standard-GEMDOS:

0 (binär 000): nur lesen
1 (binär 001): nur schreiben
2 (binär 010): lesen und schreiben

Der Sharing-Modus legt fest, auf welche Art andere Prozesse auf die Datei zugreifen dürfen, wenn sie erst einmal geöffnet ist:

0 (binär 000): Kompatiblitätsmodus (logischerweise der Modus, der von Programmen benutzt wird, die nichts von den Netzwerkerweiterungen wissen)

1 (binär 001): Lesen und Schreiben verbieten (die Datei darf kein zweites Mal geöffnet werden)

2 (binär 010): Schreiben verbieten (die Datei darf nur noch zum Lesen geöffnet werden)

3 (binär 011): Lesen verbieten (die Datei darf nur noch zum Schreiben geöffnet werden)

4 (binär 100): Alles ist erlaubt

Beim zweiten Versuch, die Datei zu öffnen, wird dann überprüft, ob sich der verlangte Modus mit dem bereits aktiven Modus unter einen Hut bringen läßt.

Für im Kompatibilitätsmodus geöffnete Dateien gelten folgende Regelungen:

- Die Datei kann anschließend nur noch vom gleichen Prozeß geöffnet werden – und auch nur ebenfalls wieder im Kompabilitätsmodus.

- Die Ausnahme von der Regel: Wird eine schreibgeschützte Datei im Kompabilitätsmodus zum Lesen geöffnet, dann wird automatisch Sharing-Modus 2 ("Schreiben verbieten") angenommen.

Achtung: Das File-Locking kann *beratend* implementiert sein ("advisory file locking"), d.h., daß es möglich sein kann, auf Bereiche trotz eines vorgenommenen Lockings zuzugreifen. Programme, die File-Locking benutzen, sollten daher *vor* einem Zugriff selbst einen Lock setzen und diesen unmittelbar *nach* dem Zugriff wieder freigeben. So ist garantiert, daß etwaige Locks anderer Seiten respektiert werden.

Fread (GEMDOS 63) – Read from file

Liest eine bestimmte Anzahl von Bytes aus einer Datei an eine gegebene Adresse.

Deklaration in C:

```
LONG Fread (WORD handle, LONG count, void *buffer);
```

Aufruf in Assembler:

```
pea     buffer          ; Offset 8
move.l  count,-(sp)     ; Offset 4
move.w  handle,-(sp)    ; Offset 2
move.w  #$3F,-(sp)      ; Offset 0
trap    #1
lea     $C(sp),sp
```

Parameter:

handle:	Dateikennung der betreffenden Datei
count:	Anzahl der zu lesenden Bytes
buffer:	Anfangsadresse des Puffers
Fread():	Entweder die Gesamtzahl der tatsächlich gelesenen Bytes (Vorsicht bei Überprüfung – das ist ein *LONG*-Wert; 0 bedeutet Dateieende) oder EIHNDL (–37): Dateikennung falsch ELOCKED (–58): Dateiausschnitt ist blockiert oder eine andere Fehlermeldung

Bemerkungen

In GEMDOS-Versionen vor 0.15 ("Rainbow-TOS") kommt es zum Systemhalt, wenn man als "count" 0 übergibt. Beim Lesen von Standardkanälen dürfen maximal 16383 Zeichen gelesen werden.

Frename (GEMDOS 86) – Rename file

Eine Datei wird umbenannt. Der neue Name darf auch einen kompletten Zugriffspfad innerhalb des betreffenden logischen Laufwerks enthalten – die Datei wird dann verschoben ("gemoved")!

Deklaration in C:

```
WORD Frename (const char *oldname, const char *newname);
```

Aufruf in Assembler:

```
pea       newname        ; Offset 8
pea       oldname        ; Offset 4
move.w    #0,-(sp)       ; Offset 2
move.w    #$56,-(sp)     ; Offset 0
trap      #1
lea       $C(sp),sp
```

Parameter:

oldname:	Zeiger auf bisherigen Dateinamen	
newname:	Zeiger auf neuen Dateinamen (ggfs. mit Zugriffspfad)	
Frename():	OK (0):	alles in Ordnung
	EPTHNF (–34):	Ordner nicht gefunden
	EACCDN (–36):	Zugriff verwehrt (Datei schreibgeschützt)
	ENSAME (–48):	nicht auf dem gleichen logischen Laufwerk
	oder eine andere Fehlermeldung	

Bemerkungen

Ab GEMDOS 0.15 kann man mit dieser Funktion auch Ordner umbenennen (allerdings nicht in der Ordnerhierarchie verschieben).

Man sollte nicht versuchen, eine Datei umzubenennen, die bereits geöffnet ist (es gelten die gleichen Gründe wie bei "Fdelete()").

Fseek (GEMDOS 66) – Seek file pointer

Mit "Fseek()" kann der Zeiger für Zugriffe innerhalb einer Datei neu gesetzt werden (sehr praktisch beispielsweise beim Überspringen von Dateiteilen).

Deklaration in C:

```
LONG Fseek (LONG offset, WORD handle, WORD seekmode);
```

Aufruf in Assembler:

```
move.w    seekmode,-(sp) ; Offset 8
move.w    handle,-(sp)   ; Offset 6
move.l    offset,-(sp    ; Offset 2
move.w    #$42,-(sp)     ; Offset 0
trap      #1
lea       $A(sp),sp
```

Parameter:

offset:	Anzahl von Bytes, die übersprungen werden sollen	
handle:	Dateikennung der betreffenden Datei	
seekmode:	0: positionieren	ab Dateistart
	1:	ab aktueller Position
	2:	ab Dateiende
Fseek():	Neue absolute Position in der Datei, oder	
	EIHNDL (–37):	falsche Datei-Kennung (handle)

Bemerkungen

Mit "Fseek()" kann man auch feststellen, ob eine Ein-/Ausgabeumlenkung auf den Bildschirm vorliegt:

```
WORD isatty (WORD handle)
{
      LONG oldoffset, rc;
      oldoffset = Fseek (0L, handle, 1);
      rc = Fseek (1L, handle, 0);
      Fseek (oldoffset, handle, 0);
      return (rc != 1);
}
```

Leider klappt diese Methode nicht, wenn die betreffende Datei leer ist. Dieser Fall tritt häufig in Shells mit Ein-/Ausgabeumlenkung (bzw. Pipes) auf. Abhilfe für die Shell: Ausgabeumlenkung immer nur für nicht-leere Dateien machen. Bei Ausgabedateien immer gleich ein <Ctrl><Z> (End-Of-File, ASCII 26) hineinschreiben und mit "Fseek()" wieder an den Dateianfang zurückgehen.

Fsetdta (GEMDOS 26) – Set DTA

Setzt die Anfangsadresse der aktuellen DTA (wird von "Fsfirst()" und "Fsnext()" benutzt):

```
typedef struct
{
     BYTE d_reserved[21];     /* fürs GEMDOS reserviert */
     BYTE d_attrib;           /* Dateiattribut */
     UWORD d_time;            /* Uhrzeit */
     UWORD d_date;            /* Datum */
     LONG d_length;           /* Dateilänge */
     char d_fname[14];        /* Dateiname */
} DTA;
```

Deklaration in C:

```
void Fsetdta (DTA *ptr);
```

Aufruf in Assembler:

```
pea       ptr            ; Offset 2
move.w    #$1A,-(sp)     ; Offset 0
trap      #1
addq.l    #6,sp
```

Parameter:

ptr: Anfangsadresse der neuen DTA

Bemerkungen

Bei Programmstart wird die DTA ab Offset $80 (128) in der Basepage eingerichtet (das ist am Anfang der Kommandozeile...)!

Fsfirst (GEMDOS 78) – Search first

Diese Funktion erlaubt es, Informationen über Dateien zu erfragen. Die zu übergebende Dateispezifikation darf eine beliebige Dateibezeichnung (also mit Laufwerk oder ohne, mit absolutem oder relativem Pfad) enthalten.

Je nach Wert wird dann in dem angegebenen Verzeichnis oder im aktuellen Verzeichnis gesucht.

Die Dateibezeichnung darf auch einen unvollständigen Dateinamen enthalten. Als Platzhalter (Wildcards) kommen dabei das "?" (steht für einen beliebigen Buchstaben) und das "*" (steht für mehrere beliebige Buchstaben außer dem Punkt) zum Einsatz.

"Fsfirst()" liefert Informationen über die erste Datei, auf die das Muster paßt (Informationen über weitere Dateien kann man dann mit "Fsnext()" erfragen).

Einige wichtige Hinweise:

- Wildcards dürfen nur im eigentlichen Dateinamen, nicht aber im Pfadnamen auftauchen.

- Die GEMDOS-internen Routinen zur Wildcard-Suche sind nicht ganz fehlerfrei (so paßt "A*.**" beispielsweise auf *jede* Datei). Im Zweifelsfall ist es sicherer, die Auswertung selbst zu machen und GEMDOS nach "*.*" (alle Dateien) suchen zu lassen. Nicht umsonst verfahren die AES in der Dateiauswahlbox genauso.

- Wenn der angegebene Pfad nicht existiert, erhält man nicht etwa eine Fehlermeldung, sondern es werden lediglich keine Dateien gefunden.

Das zu suchende Dateiattribut kann über "attribs" angegeben werden. Dabei werden nacheinander folgende Regeln ausgewertet:

1. Falls "attribs" 8 ist, erscheint die Datei nur dann, wenn ihr Attributbyte auch den Wert 8 hat (zu deutsch: wenn man das Bit für Diskettennamen gesetzt hat, werden auch nur die Diskettennamen zurückgeliefert!).

2. Wenn das Dateiattribut "Archiv" und "schreibgeschützt" enthält oder wenn es 0 ist, erscheint die Datei.

3. Wenn es Attributbits gibt, die sowohl im Dateiattribut als auch in "attribs" gesetzt sind, erscheint die Datei.

Das klingt nicht nur kompliziert – das ist es auch. Daher ein Blick in die C-Routine im GEMDOS, die für die Auswertung zuständig ist (vielen Dank an Atari für die Offenlegung dieser Zeilen!). Sei "da" das Dateiattribut der untersuchten Datei:

```
if ( ((da != 0) && (attribs != 8)) || ((attribs | 0x21) & da)))
{
     /* Datei anzeigen */
}
```

Folge: Versteckte Dateien und Systemdateien werden nur dann korrekt als solche behandelt, wenn bei ihnen nicht das "Schreibschutz"- oder das "Archiv"-Bit gesetzt ist (noch ein Grund, die Auswertung selbst vorzunehmen...). Verzeichnisse und Diskettennamen erscheinen nur dann, wenn auch das passende Bit in "attribs" gesetzt ist. Welche Bits muß man nun setzen, um alle Dateien und Ordner, nicht aber die Diskettennamen zu bekommen? Dazu nochmal die symbolischen Konstanten:

```
#define   FA_RDONLY 0x01
#define   FA_HIDDEN 0x02
#define   FA_SYSTEM 0x04
#define   FA_LABEL  0x08
#define   FA_DIREC  0x10
#define   FA_ARCH   0x20
#define   FA_ATTRIB (FA_DIREC|FA_RDONLY|FA_HIDDEN|FA_SYSTEM)
```

Mit "FA_ATTRIB" (0x17) als Attributmuster bekommt man also alle "interessanten" Dateien. Das Ergebnis wird in der DTA ("Disk Transfer Address") abgelegt, deren Adresse man mit "Fgetdta()" lesen und mit "Fsetdta()" verändern kann:

```
typedef struct
{
     BYTE d_reserved[21];     /* fürs GEMDOS reserviert */
     BYTE d_attrib;           /* Dateiattribut */
     UWORD d_time;            /* Uhrzeit */
     UWORD d_date;            /* Datum */
     LONG d_length;           /* Dateilänge */
     char d_fname[14];        /* Dateiname */
} DTA;
```

Deklaration in C:

```
WORD Fsfirst (const char *fspec, WORD attribs);
```

Aufruf in Assembler:

```
move.w    attribs,-(sp)   ; Offset 6
pea       fspec           ; Offset 2
move.w    #$4E,-(sp)      ; Offset 0
trap      #1
addq.l    #8,sp
```

Parameter:

fspec:	Zeiger auf den Dateinamen
attribs:	Jeweils für den gesuchten Dateityp das entsprechende Bit setzen:
	Bit 0: Datei schreibgeschützt
	Bit 1: Datei versteckt ("hidden")
	Bit 2: Systemdatei
	Bit 3: Diskettenname ("volume name")
	Bit 4: Teilverzeichnis (Ordner)
	Bit 5: Archiv-Bit
Fsfirst():	einige mögliche Return-Werte:
	OK (0): alles in Ordnung
	EFILNF (–33): Datei nicht gefunden
	ENMFIL (–49): Keine weiteren Dateien

Fsnext (GEMDOS 79) – Search next

Setzt die mit "Fsfirst()" begonnene Suche fort.

Deklaration in C:

```
WORD Fsnext (void);
```

Aufruf in Assembler:

```
move.w    #$4F,-(sp)      ; Offset 0
trap      #1
addq.l    #2,sp
```

Parameter:

Fsnext():	OK (0):	alles in Ordnung
	ENMFIL (–49):	Keine weiteren Dateien!

Fwrite (GEMDOS 64) – Write to file

Schreibt eine Anzahl von Bytes aus einem Puffer in eine gegebene Datei.

Deklaration in C:

```
LONG Fwrite (WORD handle, LONG count, void *buffer);
```

Aufruf in Assembler:

```
pea       buffer          ; Offset 8
move.l    count,-(sp)     ; Offset 4
move.w    handle,-(sp)    ; Offset 2
move.w    #$40,-(sp)      ; Offset 0
trap      #1
lea       $C(sp),sp
```

Parameter:

handle: Dateikennung der betreffenden Datei
count: Anzahl der zu schreibenden Bytes
buf: Anfangsadresse des Puffers
Fwrite(): Entweder die Gesamtzahl der tatsächlich geschriebenen Bytes (wichtig, falls der Platz auf der Diskette nicht gereicht hat) oder
EACCDN (–36): Zugriff verwehrt (Datei schreibgeschützt?)
EIHNDL (–37): Falsche Dateikennung
ELOCKED (–58): Dateiausschnitt ist blockiert
oder eine andere Fehlermeldung

Maddalt (GEMDOS 20) – Inform GEMDOS of "alternative" memory

Mit diesem Aufruf (erhältlich ab GEMDOS 0.19) kann man einen Block von "Alternate RAM" in die GEMDOS-Speicherliste aufnehmen lassen. GEMDOS verwaltet den Block dann ganz so, wie alles andere "Alternate RAM", das bereits beim Systemstart gefunden wurde. Ist der Block einmal installiert, kann er GEMDOS nicht wieder weggenommen werden.

Mögliche Anwendung: VME-Bus-Karten für den TT, deren Speicher für GEMDOS zugänglich gemacht werden soll (das kann, aber muß nicht eine Speichererweiterungskarte sein: auch für überflüssiges RAM auf einer Grafikkarte kann das sinnvoll sein). Logischerweise darf der Aufruf nur einmal pro Block stattfinden (am besten also im AUTO-Ordner).

Deklaration in C:

```
LONG Maddalt (void *start, LONG size);
```

Aufruf in Assembler:

```
move.l    size,-(sp)      ; Offset 6
move.l    start,-(sp)     ; Offset 2
move.w    #$14,-(sp)      ; Offset 0
trap      #1
lea       $A(sp),sp
```

Parameter:

start:	Anfangsadresse des Speicherbereichs
size:	Länge des Speicherbereichs
Maddalt():	0 (alles o.k.) oder eine Fehlermeldung

Malloc (GEMDOS 72) – Allocate memory

Reserviert Speicherplatz oder berechnet verfügbaren Speicherplatz.

Deklaration in C:

```
void *Malloc (LONG amount);
```

Aufruf in Assembler:

```
move.l    amount,-(sp)    ; Offset 2
move.w    #$48,-(sp)      ; Offset 0
trap      #1
addq.l    #6,sp
```

Parameter:

amount:	–1:	Berechne Länge des größten verfügbaren Speicherblocks
	sonst:	Anzahl der zu reservierenden Bytes (muß eine *positive* Zahl sein, ansonsten wird es zu Fehlverhalten von GEMDOS kommen!)
Malloc():	für amount = –1:	Länge des größten verfügbaren Speicherblocks
	sonst:	Anfangsadresse des reservierten Speicherbereiches (oder im Fehlerfall 0)

Bemerkungen

Bei Programmstart wird der größte verfügbare Speicherblock für das gestartete Programm alloziert. Im Programm muß man also erst "Mshrink()" benutzen, bevor man "Malloc()" wieder sinnvoll anwenden kann.

GEMDOS unterstützt nur eine begrenzte Anzahl von allozierten Speicherblöcken (ausprobieren!). Daher sollte man diese Funktion nicht allzu häufig (etwa 30mal) innerhalb eines Programmes verwenden, sondern statt dessen größere Speicherblöcke (von mindestens 16 Kilobyte) reservieren und diese dann selbst verwalten (siehe dazu die Diskussion des GEMDOS-Pools)!

Die Speicherverwaltung wird besonders ineffektiv, wenn man viele kleinere Blöcke alloziert und diese dann nicht in der gleichen Reihenfolge wieder freigibt.

"Malloc(0)" liefert vor GEMDOS 0.15 keine sinnvollen Ergebnisse, ab 0.15 wird korrekterweise ein Fehler gemeldet (0).

Man darf sich keinesfalls darauf verlassen, daß man tatsächlich soviel Bytes wie mit "Malloc (–1)" festgestellt allozieren kann. Insbesondere Konstruktionen wie "Malloc (Malloc (–1))" können ohne weiteres scheitern.

Der Grund: Unter einem multitaskingfähigen TOS (bzw. unter einer multitaskingfähigen GEMDOS-Erweiterung wie "MiNT") kann zwischen den beiden Aufrufen ein Prozeßwechsel stattfinden.

Und ein letzter Hinweis: man darf keinerlei Annahmen über Lage und Inhalt des allozierten Speichers machen!

- Der Speicherblock muß nicht leer sein (siehe "Fastload"-Flag im Programmkopf).

- Nacheinander allozierte Speicherblöcke müssen nicht unbedingt zusammenhängen.

- Man darf *nie* Speicherbereiche benutzen, die nicht dem eigenen Prozeß gehören (Original-Ton Atari im "Pexec Cookbook": "Thou shalt not mess with memory thou ownest not"). Dies ist insbesondere in Hinsicht auf eine geplante TOS-Version mit Memory Protection wichtig ("Memory Protection" durch die PMMU erlaubt es, Schreibzugriffe auf "fremden" Speicher durch eine Exception zu verhindern).

Mfree (GEMDOS 73) – Release memory

Ein allozierter Speicherblock wird wieder freigegeben.

Deklaration in C:

```
WORD Mfree (void *saddr);
```

Aufruf in Assembler:

```
pea       saddr          ; Offset 2
move.w    #$49,-(sp)     ; Offset 0
trap      #1
addq.l    #6,sp
```

Parameter:

saddr:	Anfangsadresse des freizugebenden Speicherbereiches	
Mfree():	OK (0):	alles in Ordnung
	EIMBA (-40):	Falsche Speicherblockadresse (nicht mit "Malloc()" reserviert...)

Bemerkungen

In den aktuell vorliegenden GEMDOS-Versionen wird *nicht* überprüft, ob der Speicherblock dem betreffenden Prozeß selbst gehört. Daher kann man theoretisch auch von anderen Prozessen reservierte Speicherbereiche freigeben. In Hinblick auf künftige multitaskingfähige TOS-Versionen ist dies aber sicherlich keine gute Idee!

Mshrink (GEMDOS 74) – Shrink size of allocated block

Verkürzt einen durch "Malloc()" reservierten Speicherbereich.

Deklaration in C:

```
WORD Mshrink (void *block, LONG newsize);
```

Aufruf in Assembler:

```
move.l    newsize,-(sp)   ; Offset 8
pea       block           ; Offset 4
move.w    #0,-(sp)        ; Offset 2; siehe Anmerkung
move.w    #$4A,-(sp)      ; Offset 0
trap      #1
lea       $C(sp),sp
```

Parameter:

newsize:	Neue (verkürzte) Länge des Speicherblocks	
block:	Anfangsadresse des zu verkürzenden Speicherblocks	
Mshrink():	einige mögliche Return-Werte:	
	OK (0):	alles in Ordnung
	EIMBA (–40):	falsche Blockadresse
	EGSBF (–67):	Speicherblock würde vergrößert

Bemerkungen

Üblicherweise wird beim C-Binding (zum Beispiel in "osbind.h") der Nullparameter automatisch hinzugefügt, so daß man ihn beim Aufruf weglassen muß!

Mxalloc (GEMDOS 68) – Allocate memory (with preference)

Dieser Aufruf entspricht "Malloc()" (und existiert ab GEMDOS 0.19). Der einzige zusätzliche Parameter erlaubt die Auswahl zwischen den beiden verschiedenen RAM-Arten (siehe GEMDOS-Einführung):

Modus	Bedeutung
0	nur aus dem ST-RAM
1	nur aus dem Alternate RAM
2	egal, aber lieber aus dem ST-RAM
3	egal, aber lieber aus dem Alternate RAM

In Modus 0 und 1 wird versucht, einen Block des verlangten Speichertyps zu allozieren. Wenn dazu nicht genug Speicher da ist, geht die Anforderung schief. In Modus 2 und 3 wird erst unter den freien Blöcken des angegebenen Typs gesucht, ansonsten aber auf die andere Speicherart ausgewichen. Je nach Einstellung im Programmkopf (siehe Erläuterungen zu "ph_prgflags" in der GEMDOS-Einführung) werden "Malloc()"-Anforderungen als "Mxalloc (0,...)" (wenn kein Alternate RAM angefordert werden darf) oder "Mxalloc (3...)" (sonst) ausgeführt. Auch die Speicherplatzabfrage (mit –1L als "Länge") erlangt dadurch eine neue Bedeutung: in den Modi 0 und 1 wird der größte Block der entsprechenden RAM-Art zurückgeliefert. Die beiden anderen Modi sind logischerweise in diesem Zusammenhang identisch und liefern den größten Block ungeachtet des Typus zurück.

Deklaration in C:

```
void *Mxalloc (LONG amount, WORD mode);
```

Aufruf in Assembler:

```
move.w    mode,-(sp)      ; Offset 6
move.l    amount,-(sp)    ; Offset 2
move.w    #$44,-(sp)      ; Offset 0
trap      #1
addq.l    #8,sp
```

Parameter:

amount:	siehe "Malloc()"
mode:	siehe oben
Mxalloc():	siehe "Malloc()"

Pexec (GEMDOS 75) – Load/Execute Process

Ermöglicht das Laden und Starten von Programmen, ohne daß das aktuelle Programm aus dem Speicher entfernt wird. So geladene Programme werden von den AES nicht als separate Applikationen angesehen!

Deklaration in C:

```
LONG Pexec (WORD mode, const char *file, const char *cmdlin,
            const char *env);
```

Aufruf in Assembler:

```
pea       env             ; Offset 12
pea       cmdlin          ; Offset 8
pea       file            ; Offset 4
move.w    mode,-(sp)      ; Offset 2
move.w    #$4B,-(sp)      ; Offset 0
trap      #1
lea       $10(sp),sp
```

Parameter:

file: Name der Programmdatei

cmdlin: Kommandozeile für das Programm als Pascal-String: im ersten Byte steht die Länge der Kommandozeile, die erst ab dem zweiten Byte folgt. Die Maximallänge für die Kommandozeile beträgt 124 Zeichen – mit Hilfe des ARGV-Verfahrens kann man auch mehr Informationen übergeben.

env: Zeiger auf das Environment. Entweder 0 – dann legt GEMDOS eine Kopie des aktuellen Environments an – oder ein Zeiger auf das zu übergebende Environment.

Hier die verschiedenen Modi (keine Sorge, im allgemeinen ist nur Modus 0 von Interesse!):

LOAD AND GO

```
LONG Pexec (0, const char *file, const char *cmdlin,
            const char *env);
```

Legt Environment und Basepage an, lädt und reloziert das in "file" angegebene Programm und startet es. Der Return-Wert ist entweder ein negativer 32-Bit-Wert (wenn innerhalb von

"Pexec()" ein Fehler aufgetreten ist) oder ein 16-Bit-Wert (der Wert, den das gestartete Programm mit "Pterm()", "Ptermres()" bzw. "Pterm0()" zurückgeliefert hat).

Bei einem Programmabsturz werden die allseits bekannten Bomben auf dem Bildschirm gemalt und der Wert 0x0000FFFF (–1 als WORD) zurückgeliefert (vor GEMDOS 0.15: 0).

Bei einem Programmabbruch durch CTRL-C erhält man 0x0000FFE0 (also –32 als WORD) zurück (leider ist dies augenscheinlich nicht offiziell dokumentiert).

Einige mögliche GEMDOS-Fehlermeldungen:

EINVFN (–32):	Falsche Funktionsnummer
EFILNF (–33):	Programm nicht gefunden
ENSMEN (–39):	Nicht genügend Speicher
EPLFMT (–66):	Keine GEMDOS-Programmdatei

LOAD, DON'T GO

```
BASEPAGE *Pexec (3, const char *file, const char *cmdlin,
                 const char *env);
```

Dieser Modus wird meist zum Starten von Overlays oder zum Laden von Programmen in einen Debugger benutzt.

Er entspricht Modus 0, mit dem Unterschied, daß das Programm nicht sofort gestartet wird, sondern statt dessen ein Zeiger auf die Basepage zurückgeliefert wird.

Wenn der so geladene Prozeß mehr als einmal gestartet werden soll, muß man folgende Probleme beachten:

- Das Programm darf weder Text- noch Datensegment verändern (im Zweifelsfall sollte man gleich zu Beginn eine Kopie des Datensegments anlegen und vor jedem Start in das eigentliche Programm kopieren).

- Das BSS-Segment muß vor jedem Start gelöscht werden.

- Der Zustand der Standardkanäle (Ein-/Ausgabeumlenkung) wird bereits bei diesem Aufruf (und nicht erst beim eigentlichen Start) vererbt.

  Daher dürfen sie sich vor dem eigentlichen Aufruf mit Modus 4 oder 6 nicht mehr verändern (und natürlich auch nicht *zwischen* den Aufrufen).

JUST GO

```
LONG Pexec (4, 0L, BASEPAGE *basepage, 0L);
```

Startet ein bereits geladenes Programm. Die Basepage sollte mit Modus 3 oder 5 erzeugt worden sein. Der Returnwert entspricht dem von Modus 0.

Dieser Modus hat ein schwerwiegendes Problem: durch einen Fehler im GEMDOS bleibt der aufrufende Prozeß Besitzer des durch Environment und TPA belegten Speichers. Die Folge: nach Rückkehr aus dem Prozeß muß man selbst für die Freigabe sorgen:

```
void FreeForMode4 (BASEPAGE *child)
{
     Mfree (child->p_env);
     Mfree (child);
}
```

CREATE BASEPAGE

```
BASEPAGE *Pexec (5, 0L, const char *cmdlin, const char *env);
```

Zunächst legt GEMDOS ein neues Environment an. Dann wird der größte zusammenhängende Speicherblock alloziert und in den ersten 256 Bytes eine Basepage angelegt. Start und Länge der einzelnen Programmsegmente werden noch nicht eingetragen (wie auch?). Für diesen Modus gibt es mehrere sinnvolle Anwendungen: einmal kann man das Programm selbst laden, relozieren und die fehlenden Werte in der Basepage eintragen.

Dazu ein Beispiel:

```
typedef struct
{
     LONG offset; /* Offset zum ersten zu relozierenden Wert */
     BYTE info[]; /* Byte-Tabelle */
} RELOCINFO;

BASEPAGE *P;
PH *Image;        /* Zeiger auf geladenes Programm */
LONG ProgSize;    /* Länge der Programmdatei */
RELOCINFO *reloc; /* Zeiger auf Relozierungsinformationen */
LONG *toreloc;
BYTE *info;
```

```
P = (BASEPAGE *)Pexec (5, 0L, "", 0L);

/* Programm in neue TPA kopieren */

memcpy ((void *)((long)P + sizeof(BASPAG)),
     (void *)((long)Image + sizeof(PH)), ProgSize - sizeof(PH));

/* Basepage-Zeiger eintragen */

P->p_tbase = ((void *) ((long)P + sizeof(BASEPAGE)));
P->p_dbase = ((void *) ((long)P->p_tbase + (long)Image->ph_tlen));
P->p_bbase = ((void *) ((long)P->p_dbase + (long)Image->ph_dlen));
P->p_tlen = Image->ph_tlen;
P->p_dlen = Image->ph_dlen;
P->p_blen = Image->ph_blen;
P->p_dta = (DTA *)&(P->p_cmdlin);

/* und relozieren... */

reloc = (RELOCINFO *)P->p_bbase;
toreloc = (long *) ((long)P->p_tbase + reloc->offset);
info = reloc->info;
*toreloc += (long)P->p_tbase;

while (*info)
{
     BYTE offset;
     offset = *info++;
     if (offset == 1)
          toreloc = (long *) ((long)toreloc + 254L);
     else
     {
          toreloc = (long *) ((long)toreloc + (long)offset);
          *toreloc += (long)P->p_tbase;
     }
}

/* und starten... */

Pexec (4, 0L, P, 0L);
```

Auch zum Starten eines Programms im ROM kann Modus 5 benutzt werden (genauso wird auch GEM gestartet). Dabei sollte man allerdings nicht vergessen, daß für die TPA der größte verfügbare Speicherblock alloziert wird. Sollte dies ein Problem sein (zum Beispiel weil das zu startende Programm keinen "Mshrink()" macht), dann muß man selbst "Mshrink()" benutzen und den Zeiger auf das TPA-Ende entsprechend korrigieren.

JUST GO, THEN FREE – ab GEMDOS 0.15

```
LONG Pexec (6, 0L, BASEPAGE *basepage, 0L);
```

Entspricht Modus 4, mit dem entscheidenden Unterschied, daß allozierter Speicher dem gestarteten Prozeß und nicht dem Starter gehört. Zusammen mit Modus 3 kann man also den gleichen Effekt wie durch Modus 0 erreichen.

CREATE BASEPAGE, RESPECTING PRGFLAGS – ab GEMDOS 0.19

```
LONG Pexec (7, LONG prgflags, const char *cmdlin, const char *env);
```

Wie Modus 5, nur kann in "prgflags" die gleiche Information wie im Feld "ph_prgflags" des Programmkopfs übergeben werden. Dieser Modus wird vom BIOS des TT benutzt, um GEM zu starten (GEM erhält dabei die "Erlaubnis", Speicher aus dem Alternate RAM zu allozieren).

Pexec() von innen

Um zu verstehen, warum die verschiedenen Modi so und nicht anders funktionieren, ist folgende Übersicht des Ablaufs eines Programmstarts via Modus 0 nützlich:

1. Programmdatei finden.

2. Environment, TPA und Basepage anlegen:

2a. Es wird Platz für das Environment angelegt und dieses vom Aufrufer kopiert (entweder dessen eigenes oder das als Parameter übergebene).

2b. Der größte verfügbare Speicherblock wird belegt und zur TPA des neuen Prozesses.

2c. In den ersten 256 Bytes der TPA werden erste Teile der Basepage initialisiert (TPA-, DTA- und Environmentzeiger).

2d. Dateikennungen, aktuelles Laufwerk und aktuelle Verzeichnisse werden vom Aufrufer vererbt und in die Basepage eingetragen.

2e. Die Kommandozeile wird in die Basepage kopiert.

3. Der neue Prozeß wird "Besitzer" des Environments und der TPA.

4. Die Programmdatei wird geladen und reloziert und die Zeiger auf die Programmsegmente in der Basepage eingetragen.

5. Das Programm wird gestartet:

5a. Der Basepagezeiger "p_parent" wird gesetzt.

5b. Am Ende der TPA wird ein Stack eingerichtet und dort die Basepageadresse abgelegt.

5c. Der neuerzeugte Prozeß wird zum aktuellen Prozeß.

5d. Der Prozeß wird gestartet.

Vorsicht: Diese Information gilt nicht unbedingt für künftige GEMDOS-Versionen!

"LOAD, DON'T GO" (Modus 3) macht die Arbeitsschritte 1, 2 und 4. Da Schritt 3 übersprungen wurde, gehört der allozierte Speicher weiterhin dem Aufrufer.

"JUST GO" (Modus 4) führt nur Schritt 5 aus. Da Schritt 3 vergessen wurde, gehören Environment und TPA weiterhin dem aufrufenden Prozeß. Alle weiteren Speicheranforderungen werden dann jedoch richtig bearbeitet.

"CREATE BASEPAGE" (Modus 5) führt einzig und allein Schritt 2 aus.

"JUST GO, THEN FREE" (Modus 6) macht die Arbeitsschritte 3 und 5. Daher treten die Probleme von Modus 4 nicht auf.

"CREATE BASEPAGE, RESPECTING PRGFLAGS" entspricht Modus 5, nur werden zusätzlich die Programmflags aus dem ersten Parameter beachtet.

Soweit die Erklärungen zu den einzelnen Modi. Daß der am häufigsten benötigte Modus (einfach nur ein Programm laden und starten) problemlos funktioniert, haben wir auch gesehen.

Wie steht es aber mit diffizileren Wünschen?

Problem: Programm laden, modifzieren und einmal starten.

Dies ist eine typische Aufgabe für einen Debugger: das zu untersuchende Programm wird zunächst geladen und dann modifiziert, etwa durch Setzen eines Breakpunkts. Anschließend soll es genau einmal gestartet werden.

Lösung: Mit Modus 3 (LOAD, DON'T GO) laden und dann später mit Modus 6 "JUST GO, THEN FREE" starten. Auf GEMDOS-Versionen vor 0.15 kann man sich auch mit Modus 4 (JUST GO) behelfen, wenn man sich abschließend selbst um die Freigabe von Environment und TPA kümmert.

Problem: Programm laden und dann mehrfach starten.

Gerade für eine Shell wäre ein solches Verfahren interessant: häufig benutzte Module werden nur einmal geladen und können dann immer wieder aufs neue aktiviert werden (wobei sich die Frage stellt, wie wichtig dies heutzutage angesichts der hohen Festplattenkapazitäten und -geschwindigkeiten noch ist).

In diesem Fall würde es sich anbieten, das Programm wieder mit Modus 3 (LOAD, DON'T GO) einzulesen und sich dann den Fehler in Modus 4 zunutze zu machen. Probleme sind dabei jedoch vorprogrammiert, einige haben wir bereits bei der Erklärung von Modus 3 genannt.

Die Idee, die Basepage nur einmal zu erzeugen und dann den Prozeß mehrmals zu aktivieren, führt aber noch zu weiteren Schwierigkeiten. So wird der aufgerufene Prozeß vermutlich beim ersten Programmstart mit "Mshrink()" den nicht benötigten Teil seiner TPA freigeben. Beim nächsten Ausführen legt GEMDOS jedoch den Stack wieder an der gleichen Adresse an – diesmal im "Niemandsland" des Speichers, denn dieser Abschnitt kann ja längst für andere Zwecke vergeben worden sein. Irgend jemand müßte also dafür sorgen, daß der Zeiger "p_hitpa" in der Basepage auf das *neue* Ende der TPA gesetzt wird.

Der Aufrufer ist damit allerdings überfordert – er weiß ja nicht, wie viele Bytes der TPA der geladene Prozeß behalten hat. Der aufgerufene Prozeß *könnte* das tun – nur müßte man ihn dafür speziell anpassen, und das ist im allgemeinen mangels Quelltext nicht möglich.

Doch es gibt eine Lösung, die auch noch andere Probleme löst (auf die wir gleich noch zu sprechen kommen):

A. Programm einfach mit "Fopen()-Fread()-Fclose()" in den Speicher laden.

B. Basepage mit Pexec-Modus 5 (CREATE BASEPAGE) erzeugen.

C. Text- und Datensegment in die TPA kopieren, relozieren und fehlende Basepagezeiger setzen.

D. BSS löschen.

E. Programm mittels Modus 4 bzw. 6 ausführen.

Zur nochmaligen Ausführung können einfach die Schritte B bis E wiederholt werden. Eine Beispielfunktion zum Anlegen der Basepage sowie zum Relozieren und Starten des Programms haben wir bei den Erklärungen zu Modus 5 angegeben. Einziger Schönheitsfehler: zum Zeitpunkt der Programmausführung liegen Textsegment und Datensegment doppelt im Speicher, was natürlich bei langen Programmen nicht wünschenswert ist. Für Programme, die *nicht* darauf angewiesen sind, daß sie alle drei Programmsegmente in einem Stück vorfinden, gibt es eine elegantere Lösung:

A. Programm einfach mit "Fopen()-Fread()-Fclose()" in den Speicher laden.

B. Programmcode dort, wo er steht, relozieren.

C. Basepage mit Pexec-Modus 5 (CREATE BASEPAGE) erzeugen.

D. Zeiger auf Programm- und Datensegment in Basepage eintragen.

E. BSS löschen.

F. Programm mittels Modus 4 bzw. 6 ausführen.

Durch Wiederholung von Schritt C bis F kann man das Programm dann "beliebig" oft starten.

Kommen wir schließlich zu dem anderen Problem, das die Idee "einmal mit Modus 3 laden und immer wieder mit Modus 4 starten" zunichte macht. GEMDOS merkt sich bekanntlich in der Basepage einige Informationen, die beim Nachstarten von Prozessen "vererbt" werden. Dazu gehören beispielsweise die Zuordnungen der Standardkanäle und die aktuellen Verzeichnisse.

Wie man sich denken kann, stehen in der Basepage nicht die eigentlichen Informationen, sondern nur Verweise auf GEMDOS-interne Strukturen. Nehmen wir die Standardkanäle als Beispiel. Wenn sie auf eine "echte" Datei umgelenkt worden sind, enthält die Basepage einen Verweis auf einen Dateikontrollblock im Innern des GEMDOS.

Für jeden Dateikontrollblock merkt sich GEMDOS in einem Zähler, wie "oft" die Datei tatsächlich geöffnet ist (fachenglisch: "usage count"). Diese Information wird benötigt, damit GEMDOS weiß, wie oft die Datei geschlossen werden muß, bevor der Dateikontrollblock wieder freigegeben werden kann.

Die Vererbung dieser Informationen (und damit die Erhöhung der Zähler) findet in Schritt 2 des Pexec-Vorgangs statt. Beim Terminieren des Prozesses werden analog dazu die Zähler wieder vermindert.

Und da haben (bzw. hätten) wir den Salat: die Zähler werden nur einmal erhöht, aber mehrfach vermindert. Die Folge wäre, daß GEMDOS-interne Strukturen freigegeben würden, obwohl sie noch an anderer Stelle benötigt würden. Abstürze oder Systemhalt ("Out of internal memory...") wären dann vorhersehbar.

Aktuelle GEMDOS-Versionen (bis einschließlich 0.19) gehen also fälschlicherweise davon aus, daß jedes einmal geladene Programm auch genau einmal ausgeführt wird. Die Vererbung wird mithin zu früh gemacht – eigentlich sollte sie erst beim tatsächlichen Start des Programms vorgenommen werden (in Schritt 5 des Pexec-Vorgangs).

Dies setzt voraus, daß jedes gestartete Programm früher oder später auch beendet wird (sicherlich eine vernünftige Annahme). Atari hat angekündigt, daß künftige GEMDOS-Versionen exakt so arbeiten werden.

Bemerkungen
In älteren GEMDOS-Versionen (mindestens einschließlich 0.15) passieren unschöne Dinge, wenn während des Ladens der Programmdatei BIOS-Error (etwa defekte Sektoren) auftreten: zum Beispiel kann es leicht passieren, daß die Datei nicht wieder geschlossen wird und damit die Dateikennung verlorengeht.

Pterm0 (GEMDOS 0) – Terminate Process

Ist identisch mit "Pterm(0)".

Deklaration in C:

```
void Pterm0 (void);
```

Aufruf in Assembler:

```
clr.w     -(sp)      ; Offset 0
trap      #1
```

Pterm (GEMDOS 76) – Terminate process

Der laufende Prozeß wird beendet und aus dem Speicher entfernt. Dem aufrufenden Programm wird der angegebene Wert als Return-Status zurückgemeldet. Das kann eine normale GEMDOS-Fehlermeldung oder auch ein selbstdefinierter Fehler-Code sein.

Alle offenen Dateien (auch die Standardkanäle) werden geschlossen, und mittels "Malloc()" oder "Mxalloc()"wird reservierter Speicher freigegeben. Bei Megas und TTs wird die Uhrzeit aus der Hardware-Uhr in die Systemuhr übertragen. Außerdem wird ein Sprung durch "etv_term" durchgeführt.

Deklaration in C:

```
void Pterm (WORD retcode);
```

Aufruf in Assembler:

```
move.w    retcode,-(sp)   ; Offset 2
move.w    #$4C,-(sp)      ; Offset 0
trap      #1
```

Parameter:

retcode: Status, der an das aufrufende Programm zurückgegeben wird. Gebräuchliche Return-Werte sind:

- 0: alles in Ordnung
- 1: allgemeine Fehlermeldung
- 2: Fehler in den übergebenen Parametern
- −1: sollte man nicht selbst benutzen, da dieser Wert ab GEMDOS 0.15 bei einem Programmabsturz zurückgeliefert wird
- −32: sollte man *auch* nicht selbst benutzen, da man diesen Wert bei mit <Ctrl><C>abgebrochenen Programmen erhält

Ptermres (GEMDOS 49) – Terminate and stay resident

Der laufende Prozeß wird beendet. Dabei kann man eine beliebige Anzahl von Bytes resident im Speicher halten (gerechnet ab Anfangsadresse der Basepage).

Ebensowenig werden die mittels "Malloc()" oder "Mxalloc()" allozierten Speicherbereiche freigegeben: im Gegenteil – sie werden vollständig aus der GEMDOS-Speicherliste entfernt und können nie wieder per GEMDOS benutzt werden (außer, man hängt sie wieder mittels "Maddalt()" in die Speicherverwaltung ein).

Deklaration in C:

```
void Ptermres (ULONG keep, WORD ret);
```

Aufruf in Assembler:

```
move.w    ret,-(sp)       ; Offset 6
move.l    keep,-(sp)      ; Offset 2
move.w    #$31,-(sp)      ; Offset 0
trap      #1
```

Parameter:

keep: Anzahl der resident zu haltenden Bytes (ab Anfang der Basepage, mindestens 128 Bytes)

ret: Status, der an das aufrufende Programm zurückgegeben wird. Gebräuchliche Return-Werte sind:

- 0: alles in Ordnung
- 1: allgemeine Fehlermeldung
- 2: Fehler in den übergebenen Parametern
- –1: sollte man nicht selbst benutzen, da dieser Wert ab GEMDOS 0.15 bei einem Programmabsturz zurückgeliefert wird
- –32: sollte man *auch* nicht selbst benutzen, da man diesen Wert bei mit Ctrl><C> abgebrochenen Programmen erhält

Super (GEMDOS 32) – Get/Set/Inquire supervisor mode

Schaltet zwischen User- und Supervisor-Modus hin und her und liefert Funktionen über den momentanen Prozessorstatus.

Deklaration in C:

```
LONG Super (void *stack);
```

Aufruf in Assembler:

```
pea       stack          ; Offset 2
move.w    #$20,-(sp)     ; Offset 0
trap      #1
addq.l    #6,sp
```

Parameter:

stack: 1: Es wird der momentane Prozessorstatus (0 = User-Modus, –1 = Supervisor-Modus) zurückgegeben.

0: Ist der Prozessor im User-Modus, dann wird er in den Supervisor-Modus geschaltet und als SSP (Supervisor Stack Pointer) der bisherige USP (User Stack Pointer) benutzt. Zurückgeliefert wird der bisherige SSP. War der Prozessor zuvor im Supervisor-Modus, wird nichts Vernünftiges zurückgeliefert.

>1: Der Prozessor-Modus wird umgeschaltet. Der Funktionswert ist der bisherige Wert des SSP. "stack" wird neuer aktueller Stack-Pointer. In diesem Fall sollte man logischerweise auf die anschließende Stack-Korrektur verzichten!

Bemerkungen

Zur Klärung: Man schaltet beispielsweise mittels "suret = Super (0L)" in den Supervisor-Modus. Am Ende des Programmteils schaltet man dann mit "Super (suret)" in den User-Modus zurück.

Sversion (GEMDOS 48) – Get version number

Liefert Versionsnummer vom GEMDOS. Folgende GEMDOS-Versionen sind zur Zeit bekannt:

0.13 (0x1300)
Dieses GEMDOS befindet sich in TOS 1.00 und TOS 1.02 ("Blitter"-TOS). Es zeichnet sich durch zahllose Fehler bei der Umlenkung der Standardkanäle und durch sehr mäßige Arbeitsgeschwindigkeit auf Festplatten aus.

0.14 (0x1400)
Diese Versionsnummer trug das französische "Turbo-DOS", das für kurze Zeit mit Atari-Festplatten ausgeliefert wurde. Es war schnell und unsicher.

0.15 (0x1500)
Dieses GEMDOS befindet sich in TOS 1.04 und TOS 1.06. Es ist weitestgehend stabil und schnell. Ein bekannter Bug wird durch das offizielle Patchprogramm "POOLFIX3.PRG" behoben.

0.17 (0x1700)
Ist in TOS 1.62 zu finden und unterscheidet sich von 0.15 nur dadurch, daß das Programm "POOLFIX3.PRG" nicht mehr benötigt wird.

0.19 (0x1900)
Dieses GEMDOS tauchte erstmals im TOS 3.01 des TT auf. Es wurde in erster Linie um Funktionen zur Verwaltung der unterschiedlichen RAM-Arten erweitert.

Deklaration in C:

```
WORD Sversion (void);
```

Aufruf in Assembler:

```
move.w    #$30,-(sp)      ; Offset 0
trap      #1
addq.l    #2,sp
```

Parameter:

Sversion():	Low-Byte:	Haupt-Revision
	High-Byte:	Unter-Revision

Tgetdate (GEMDOS 42) – Get date

Das Systemdatum (GEMDOS-intern) wird abgefragt. Dabei handelt es sich um einen einfachen Zähler, der bei Systemstart initialisiert wird – ab TOS 1.02 bei Rechnern mit Hardware-Uhr mit der aktuellen Uhrzeit, sonst mit dem TOS-Erstellungsdatum (siehe "_sysbase").

Bei Rechnern mit Hardware-Uhr und ab TOS 1.02 wird zusätzlich bei jeder Beendigung eines GEMDOS-Prozesses die Zeit mit Hilfe der Hardware-Uhr aktualisiert.

Auf Rechnern ohne Hardware-Uhr kann man beim Reset die GEMDOS-Uhrzeit retten, indem man die Zeit mittels der XBIOS-Funktionen ausliest und in die GEMDOS-Uhr überträgt.

Deklaration in C:

```
UWORD Tgetdate (void);
```

Aufruf in Assembler:

```
move.w    #$2A,-(sp)      ; Offset 0
trap      #1
addq.l    #2,sp
```

Parameter:

Tgetdate(): Aktuelles Datum

Bit 0..4:	Tag (0..31)
Bit 5..8:	Monat (1..12)
Bit 9..15:	Jahre seit 1980 (0..119)

Tgettime (GEMDOS 44) – Get time

Fragt die Systemzeit der GEMDOS-internen Uhr ab (siehe auch "Tgetdate()").

Deklaration in C:

```
UWORD Tgettime (void);
```

Aufruf in Assembler:

```
move.w    #$2C,-(sp)      ; Offset 0
trap      #1
addq.l    #2,sp
```

Parameter:

Tgettime(): Aktuelle Uhrzeit

Bit	0..4:	Sekunden (0..29) – muß verdoppelt werden
Bit	5..10:	Minuten (0..59)
Bit	11..15:	Stunden (0..23)

Tsetdate (GEMDOS 43) – Set date

Setzt das Systemdatum in der GEMDOS-Uhr (ab TOS 1.02 auch in der jeweiligen Hardware-Uhr).

Deklaration in C:

```
WORD Tsetdate (UWORD date);
```

Aufruf in Assembler:

```
move.w    date,-(sp)      ; Offset 2
move.w    #$2B,-(sp)      ; Offset 0
trap      #1
addq.l    #4,sp
```

Parameter:

date:	aktuelles Datum (siehe unter "Tgetdate()")	
Tsetdate():	–1:	Das war kein gültiges Datum (0<=Tag<=31, 1<=Monat<=12, 0<=Jahr<=119)!

Tsettime (GEMDOS 45) – Set time

Setzt die aktuelle Systemzeit in der GEMDOS-Uhr (ab TOS 1.02 auch in der jeweiligen Hardware-Uhr).

Deklaration in C:

```
WORD Tsettime (UWORD time);
```

Aufruf in Assembler:

```
move.w    time,-(sp)      ; Offset 2
move.w    #$2D,-(sp)      ; Offset 0
trap      #1
addq.l    #4,sp
```

Parameter:

time:	Uhrzeit (siehe "Tgettime()")	
Tsettime():	1:	Das war keine gültige Uhrzeit (0<=Sekunden<=29, 0<=Minuten<=59, 0<=Stunden<=23)!

Kapitel 3: VDI-Betriebssystemroutinen

Einleitung

Das VDI ("Virtual Device Interface") bildet die "untere" Hälfte von GEM. Genauso wie das BIOS den Unterbau für alle Funktionen des GEMDOS bereitstellt, ist das VDI Grundlage für alle AES-Funktionen. Dazu bedient es sich aller tieferliegenden Betriebssystemschichten (BIOS, XBIOS bis GEMDOS) und greift innerhalb der gerätespezifischen Teile naturgemäß auch direkt auf die Hardware zu.

Auch wenn man mit dem VDI normalerweise primär Funktionen zur Grafikausgabe verbindet, erstreckt sich der Aufgabenbereich auch auf Eingaben (wie zum Beispiel per Maus oder Grafiktablett). Das VDI ist also eine Betriebssystemschicht zur Ansteuerung von im weitesten Sinn grafikorientierten Aus- *und* Eingabegeräten.

Ebensowenig sollte man das Wörtchen "Virtual" übersehen. Für das Anwenderprogramm bedeutet dies, daß (fast) alle Funktionsaufrufe auf jedes verfügbare Ausgabegerät – ob Bildschirm, Plotter, Drucker oder auch Diabelichter – angewendet werden können.

Zwei kurze Beispiele, um die Geräteunabhängigkeit des VDI zu verdeutlichen: die AES machen ausschließlich GEMDOS- und VDI-Aufrufe. Daher ist es möglich, die AES ohne jede Änderung auch auf solchen Grafikkarten einzusetzen, an deren Existenz 1984 ganz bestimmt noch niemand gedacht hat, zum Beispiel: 1280 * 960 Bildpunkte in 256 Farben.

Gleiches gilt natürlich für alle sauber programmierten GEM-Anwendungen, deren Zahl in den letzten Jahren glücklicherweise immer stärker gewachsen ist (man denke an "SciGraph", "Script 2", "LDW-Power", "Turbo-C" und "Gemini").

Bei Druckern sieht die Lage nicht anders aus: ob 9-Nadeldrucker, Tintenstrahldrucker oder Laserdrucker – immer kann man mit den gleichen Betriebssystemfunktionen Kreise malen oder Texte ausgeben.

Die Wurzeln des VDI liegen im CP/M-GSX-System, das Anfang der achtziger Jahre aus dem Bedarf für eine portable Grafikschnittstelle für die damals dominierenden CP/M-Systeme entstand (nicht vergessen: CP/M – das erste verbreitete Betriebssystem für Mikrocomputer, stammte ebenfalls von Digital Research).

Ähnlichkeiten in Funktionen und Bezeichnungen zum GKS ("Grafisches Kern-System") sind keineswegs zufällig, sondern von den Entwicklern bei Digital Research voll beabsichtigt. Das VDI erfüllt den ANSI-Standard X3H3.6 CG-VDI!

Grundlagen des VDI

Workstations

Fast alle VDI-Funktionen erwarten als Parameter das sogenannte "Handle" der zu benutzenden "Workstation". Die Parallele zu den vom GEMDOS vergebenen Datei-Handles kommt nicht von ungefähr, denn die dahinterstehende Idee ist eigentlich gleich.

Eine der vielen Aufgaben einer Betriebssystemschicht ist es schließlich, den Zugriff auf die Hardware-Ressourcen des Rechnersystems zu verwalten. Genauso, wie zwei Programme nicht gleichzeitig denselben Speicherbereich benutzen oder in dieselbe Datei schreiben dürfen, können auch die vom VDI verwalteten Ein- und Ausgabegeräte nie mehr als einem "Kunden" gleichzeitig zu Verfügung stehen. Niemand würde erwarten, daß etwas Sinnvolles dabei herauskommt, wenn zwei Applikationen gleichzeitig Daten an den Drucker schicken oder abwechselnd Tastendrücke auslesen.

Die Lösung: Wer auf ein bestimmtes VDI-Gerät zugreifen will, muß dazu eine Workstation öffnen. Je nach Sachlage erhält man als Resultat entweder eine Fehlermeldung oder die Workstation-Kennung (Handle) des betreffenden Geräts. Anschließend darf man das angesprochene Gerät nach Herzenslaune benutzen. Hauptsache, man schließt die Workstation am Ende, um sie wieder für andere Programme verfügbar zu machen.

Zu jeder Workstation gehört eine Menge von Merkmalen, die unveränderlich sind und über die Fähigkeiten des Geräts Auskunft geben. Dazu gehören neben vielen anderen Gerätetyp, Koordinatensystem und Farbfähigkeit. Neben den beim Öffnen der Workstation zurückgelieferten Daten kann man mit einer erweiterten Auskunftsfunktion noch mehr Daten abrufen. Dieses Informationsangebot macht es den Anwendungsprogrammen möglich, die Fähigkeiten des Geräts optimal zu nutzen – zum Beispiel dadurch, daß sie ihren Bildschirmaufbau an die Anzahl der darstellbaren Bildpunkte anpassen.

Daneben gibt es auch noch *veränderliche* Merkmale, die sogenannten "Attribute". Sie beschreiben den aktuellen *Zustand* des Geräts. Ein gutes Beispiel ist die Linienfarbe, die man nur ein einziges Mal – und nicht etwa bei jedem einzelnen Aufruf von Zeichenfunktionen – setzen muß. Die aktuellen Attribute werden vom VDI in internen Strukturen für jede einzelne Workstation getrennt gespeichert. So kann eine Applikation unbesorgt für Bildschirm und Plotter getrennt verschiedene Linienfarben einstellen – das VDI erkennt jeweils am Workstation-Handle, wer gemeint ist.

Der Bildschirm nimmt natürlich eine Sonderstellung ein – er *muß* schließlich für mehrere Programme gleichzeitig verfügbar sein. Wie sollten sonst das Fenstersystem der AES, mehrere

Accessories und ein Hauptprogramm gemeinsam auf dem gleichen Bildschirm ausgeben können?

Das VDI bietet dazu die sogenannten "virtuellen Workstations" an, die immer nur im Zusammenhang mit einer bereits geöffneten "physikalischen" Bildschirm-Workstation benutzt werden können. Jede virtuelle Workstation verfügt über einen eigenen Satz von Attributen. So können mehrere Programme Workstations öffnen, ohne sich bei den Bildschirmausgaben gegenseitig die Attribute zu verstellen.

Doch aufgepaßt: Nicht alle Eigenschaften der physikalischen Bildschirm-Workstation sind auf die virtuellen Workstations übertragbar. Tastatur und Maus können auch weiterhin immer nur von einem Programm gleichzeitig benutzt werden. Die Eingabefunktionen dürfen daher *nur* von dem Programm benutzt werden, das die physikalische Bildschirm-Workstation geöffnet hat.

Normalerweise "gehört" die Bildschirm-Workstation den AES. Daraus folgt, daß man in normalen GEM-Anwendungen *nie* die VDI-Eingabefunktionen benutzen darf – die Verwaltung von Maus und Tastatur wird vollständig von den AES (Screen-Manager) übernommen.

Es besteht übrigens keine Verbindung zwischen den virtuellen Workstations und dem Fenstersystem der AES. Jede virtuelle Workstation benutzt das gleiche Koordinatensystem wie die zugehörige physikalische Bildschirm-Workstation. Die Koordinaten auf Positionen relativ zum Fensterinhalt umzurechnen ist einzig und allein Aufgabe des Anwendungsprogramms.

Die Verwaltung der Workstation-Attribute erfordert natürlich Speicher. Nicht nur deshalb, weil der Bildschirmtreiber im ROM den Speicher dynamisch (per "Malloc()") anfordert, sollte man *nie* davon ausgehen, daß das Öffnen einer physikalischen oder virtuellen Workstation immer klappen muß! In vielen GDOS-Versionen (siehe unten) werden die Informationen in statischen Strukturen verwaltet, so daß die Maximalzahl von gleichzeitig geöffneten Workstations begrenzt ist.

Koordinatensysteme

Das VDI unterscheidet zwischen zwei verschiedenen Typen von Koordinatensystemen: dem "Rasterkoordinatensystem" (oder "RC-System") und dem "Normalisierten Gerätekoordinatensystem" (oder "NDC-System", steht für "Normalized Device Coordinates").

Bei Verwendung des RC-Systems kann man exakt das dem Ausgabegerät eigene Koordinatensystem nutzen. Bei einem Bildschirm entspricht das den horizontal und vertikal verfügbaren

Pixeln. Der Nullpunkt liegt in der linken oberen Ecke. Hauptvorteile der Rasterkoordinaten sind die absolut exakte Positionierung, die die genaue Arbeit mit Bildschirmrastern (Fenster etc.) erst möglich macht.

Das NDC-System hingegen hat immer eine maximale Auflösung von 32768 mal 32768 Punkten. Das VDI rechnet beim Aufruf eines Gerätetreibers automatisch auf dessen physikalisches Koordinatensystem um. Der Nullpunkt liegt in der linken *unteren* Ecke. Zur Arbeit mit dem NDC-System muß ein *GDOS* (siehe unten) installiert sein – der Bildschirmtreiber im ROM kann keine NDC-Koordinaten verarbeiten.

In der Praxis wird das NDC-System nur selten benutzt. Programme, die exakte Bemaßungen anbieten müssen (Desktop-Publishing, wissenschaftliche Grafik) müssen intern sowieso eine eigene, feinere Koordinatendarstellung benutzen.

Die Benutzung des NDC-Systems würde dazu führen, daß Koordinaten gleich zweimal konvertiert werden (einmal von der internen Darstellung in NDC-Koordinaten, anschließend in Rasterkoordinaten), was zu überflüssigen Geschwindigkeitsverlusten und Rundungsfehlern führt.

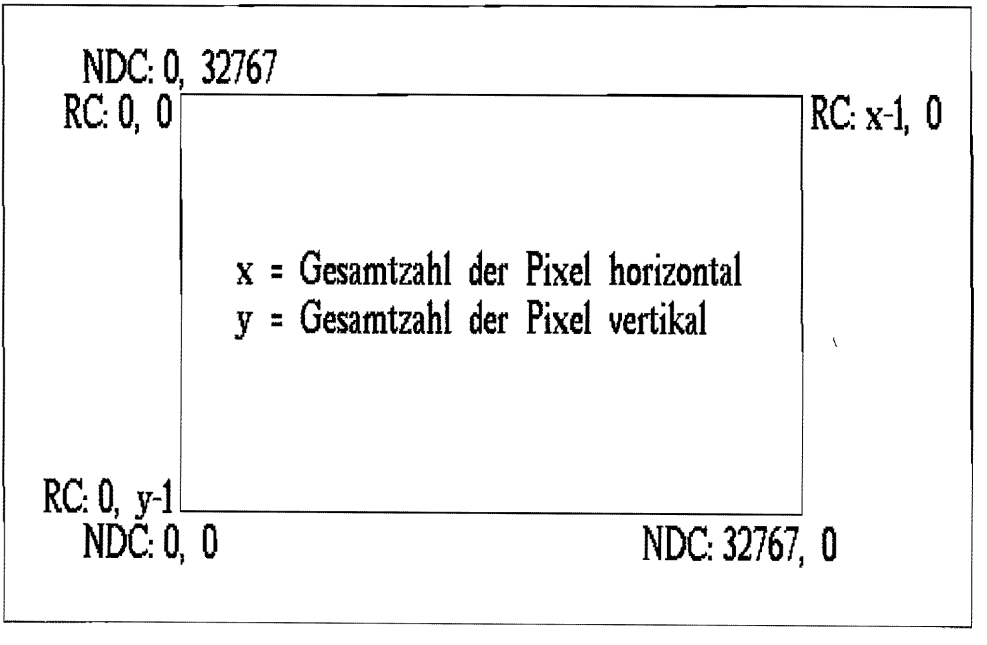


Abb. 3.1: Das RC- und das NDC-Koordinatensystem

Auflösung und Pixelgrößenverhältnis

Im Zusammenhang mit dem Begriff "Auflösung" kommt es nur zu häufig zu Mißverständnissen. Viele meinen, mit der Auflösung wäre die Anzahl der verfügbaren Bildpunkte gemeint. Weit gefehlt!

Die *Auflösung* gibt an, wie groß jeder einzelne Pixel ist. Das gebräuchliche Maß heißt "dpi", was für "dots per inch", also Pixel pro Zoll, steht (1 Zoll sind 2,54 cm). Gebräuchliche Auflösungen sind

Auflösung	Gerät
72 dpi	Macintosh-Bildschirm
300 dpi	Laserdrucker
360 dpi	24-Nadeldrucker

Rechenexempel: Bei einem handelsüblichen Laserdrucker passen

300 * 300 = 90000 Punkte

auf eine Fläche von

2,54 cm * 2,54 cm = 6,452 Quadratzentimeter.

Das sind etwa 14000 Punkte pro Quadratzentimer.

Ein anderes, im Druckgewerbe gebräuchliches Maß ist der "Punkt", der 1/72 Zoll groß ist. Schriftgrößen werden normalerweise in "Punkt" angegeben – der Text, den Sie gerade lesen, ist beispielsweise in einer 10-Punkt-Schrift gesetzt.

Wie aus der obigen Übersicht ersichtlich, ist bei einem Macintosh-Bildschirm ein Pixel genau einen "Punkt" groß. Eine 10-Punkt-Schrift nimmt auf dem Macintosh daher auch 10 Pixel ein. Diese Gleichheit vereinfacht natürlich viele Berechnungen, andererseits macht sie den Apple-Programmierern den Übergang zu neuen, höherauflösenden Bildschirmen sehr schwer.

Beim VDI sieht es etwas anders aus. Die Auflösung kann beim Öffnen der Workstation abgefragt werden und darf für Pixelbreite und Pixelhöhe verschieden sein.

Beim Zeichnen von Kreisen und ähnlicher geometrischer Figuren wird dieses *Pixelgrößenverhältnis* ("Aspect ratio") berücksichtigt – Kreise blieben also auch in der "mittleren" Auflösung des Atari ST Kreise. Als Referenzskala dient dabei die X-Achse.

Dieser Mechanismus erlaubt es prinzipiell, auch Geräte mit nicht-quadratischen Pixeln korrekt anzusteuern. Hier einige weitere Beispiele:

Auflösung	Gerät
360 dpi * 180 dpi	einige 24-Nadeldrucker
240 dpi * 216 dpi	manche 9-Nadeldrucker bei Mehrfachdruck
91 dpi * 91 dpi	"ST-Hoch" auf SM 124 (640 Pixel auf 7 Zoll)
91 dpi * 45 dpi	"ST-Mittel" auf einem 12-Zoll-Farbbildschirm

Wie man sieht, ist die Auflösung letztendlich auch von der Monitorgröße und der Bildeinstellung abhängig. Davon weiß natürlich das VDI nichts, und so sind die beim Öffnen der Workstation zu ermittelnden Werte nicht zwingend exakt. Wer es ganz präzise mag, müßte sich insofern ein Patchprogramm im AUTO-Ordner installieren, das dem VDI korrekten Pixelgrößen unterschiebt. Zumindest das Pixelgrößenverhältnis kann man jedoch den VDI-Rückgabewerten ziemlich verläßlich entnehmen.

Clipping

Clipping ist ein Verfahren, das die Benutzung eines Fenstersystems eigentlich erst möglich macht. Dabei gibt man mittels des Clipping-Rechtecks an, auf welchen Bildschirmbereich sich alle Grafikausgaben der betreffenden Workstation beziehen sollen. Alles, was aus dem so spezifizierten Bildausschnitt herausragt, wird bei der Bildschirmausgabe übergangen.

Eine zwar wenig effiziente, aber leicht zu programmierende Fensterausgaberoutine würde einfach für jedes zu zeichnende Teilrechteck des Fensters jeweis ein Clipping-Rechteck setzen und dann alle zum Aufbau des Fensterinhalts notwendigen Bildschirmausgaben wiederholen. Wer sich die Mühe macht, zunächst anhand einer Plausibilitätabfrage festzustellen, ob das zu zeichnende Objekt überhaupt *sichtbar* ist, kann natürlich deutlich Zeit einsparen.

Bleibt zu erwähnen, daß das Clipping natürlich nicht gratis zu haben ist. Wer optimale Geschwindigkeit beim Bildschirmaufbau erreichen will, sollte das Clipping-Rechteck so oft wie nur möglich ausschalten.

Rasterformate

Bei der Arbeit mit Bildschirmen spielen die *Rasterfunktionen* eine besonders große Rolle: sie sind für all das zuständig, was mit der Bewegung oder Veränderung von Bildschirmausschnitten zu tun hat – als Beispiele seien das Scrolling und das Darstellen von Icons genannt.

Die Rasterfunktionen können ihre Aufgabe natürlich nur dann effizient erledigen, wenn als interne Darstellung das gleiche Format verwendet wird, wie es auch im Bildspeicher benutzt wird. Anderenfalls müßten die Daten bei jedem einzelnen Aufruf einer Rasterfunktion konvertiert werden. Naturgemäß ist jedoch der interne Aufbau des Bildspeichers von der Hardware des benutzten Grafiksystems abhängig – mal davon abgesehen, daß der Bildspeicher gar nicht unbedingt direkt für den Prozessor zugänglich sein muß.

Wie viele Bits man pro Pixel braucht, hängt natürlich von der Anzahl der gleichzeitig darstellbaren Farben ab. Für monochrome Pixel braucht man jeweils nur ein Bit – man muß eben nur die Information "ein/aus" unterbringen. Bei 16 Farben sind es schon vier Bits (denn mit vier Bits kann man die Werte 0..15 darstellen). Diese Bits können je nach verwendeter Videohardware völlig verschieden angeordnet sein.

Bei einem *pixelorientierten* Format werden alle zu einem Pixel gehörenden Bits zusammen in ein oder mehrere Bytes kodiert. In einem 16-Farb-System würden also jeweils zwei Pixel gemeinsam in einem Byte abgespeichert; bei 256 Farben nimmt jeder Pixel ein Byte ein.

Bei *planeorientierten* Formaten verfolgt man einen anderen Ansatz. Man betrachtet den Bildspeicher einfach als eine Sammlung monochromer (einfarbiger) Bildebenen ("planes"). Um zu einem Pixel die Farbe zu ermitteln, kombiniert man die zuständigen Bits aus den einzelnen Ebenen zu einem Farbwert. Damit ist freilich noch nichts darüber ausgesagt, wie die einzelnen Ebenen im Bildspeicher angeordnet sind. Eine einfache (und im PC-Bereich auch übliche) Lösung ist es, eine Plane nach der anderen hintereinander im Bildspeicher anzuordnen.

Bei der in den bisherigen ST/TT-Modellen serienmäßig eingebauten Videohardware kommt eine spezielle Spielart der planeorientierten Formate zum Einsatz: die *Interleaved Bitplanes*. Die einzelnen Bildebenen sind dabei nicht jeweils am ganzen Stück nacheinander abgelegt, sondern wechseln sich wortweise ab (mehr dazu im Hardwareteil zum Thema "Grafiksystem"). Dieses Format verdankt seine Entstehung vermutlich technischen Zwängen bei der Entwicklung der ersten STs. Daß es künftig sicherlich nicht weiterverwendet werden wird, leuchtet ein, wenn man sich überlegt, auf wie viele Speicherzellen im Modus "TT-Niedrig" für jeden einzelnen Pixel zugegriffen werden muß (Antwort: 16 Bytes).

Doch selbst wenn es nur eine Möglichkeit zur Verwaltung der Planes gäbe, könnte daraus kaum ein Standardformat definiert werden. Orientierung der einzelnen Bits (steht Bit 15 für einen Pixel links oder rechts von Pixel 14?) und ähnliche Aspekte können prinzipbedingt bei jedem Grafiksystem anders gelöst sein.

Das VDI beseitigt dieses Problem auf salomonische Art und Weise: es unterscheidet zwischen dem *geräteabhängigen* Format (das von der Hardware abhängt) und dem *Standardformat*. Das

Standardformat ist für alle Rechner mit VDI gleich und kann daher immer dann benutzt werden, wenn Rastergrafiken von "außen" in das System eingebracht werden (Beispiel: Darstellung von Icons in einer Resource-Datei). Zur Umwandlung zwischen beiden Formaten steht die Funktion "vr_trnfm()" zu Verfügung. Und hier ist die Definition des VDI-Standardformats:

- Das Format ist planeorientiert. Jede Bildebene belegt ein zusammenhängendes Stück Speicher und hat die gleiche Anzahl von Bildpunkten.

- Das höchste Bit eines 16-Bit-Worts steht für den am weitesten links stehenden Pixel.

- Aufeinanderfolgende Worte im Bildspeicher bilden die einzelnen Zeilen. Das erste Wort einer solchen Zeile liegt am linken Bildrand. Die erste Zeile der Plane kodiert die oberste Pixelzeile.

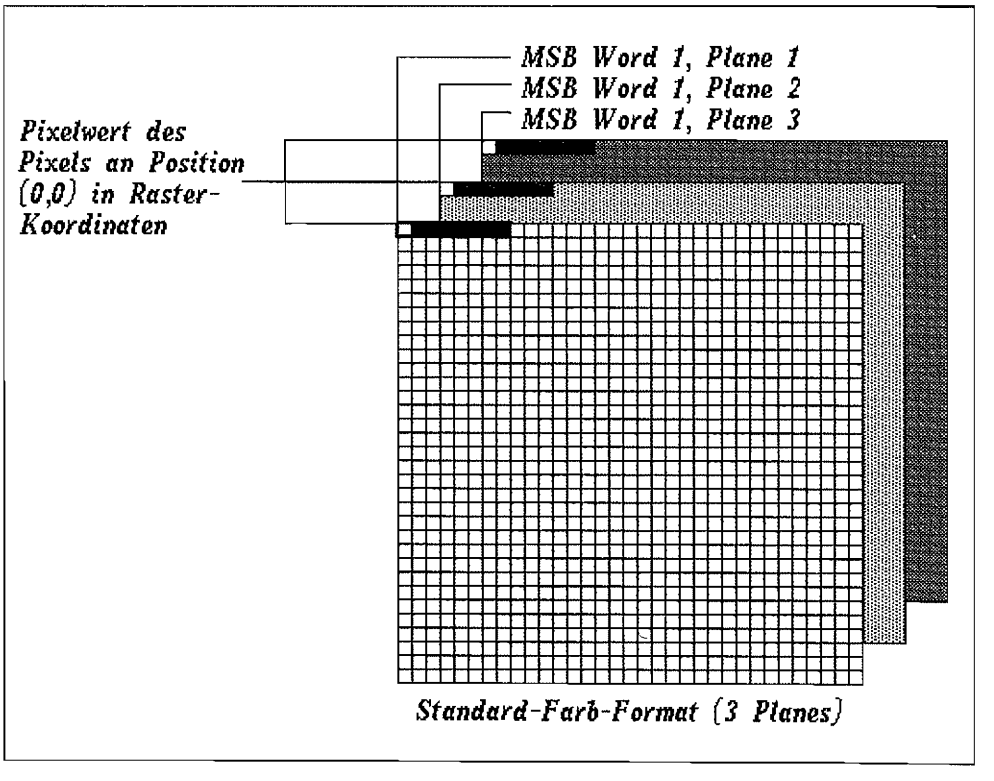


Abb. 3.2: Ein Beispiel für das geräteunabhängige Standardformat

In den monochromen Grafikmodi von ST und TT entspricht das geräteabhängige Format *zufällig* dem geräteunabhängigen Format. Das heißt natürlich nicht, daß man deshalb auf den Aufruf von "vr_trnfm()" verzichten darf!

Über das *geräteabhängige* Format hingegen hat man keinerlei Informationen. Schon deshalb verbietet sich jede direkte Manipulation am Bildspeicher von selbst.

Das VDI arbeitet prinzipiell Farbregister-orientiert ("Color-Lookup-Table"). Es geht also davon aus, daß es eine Maximalzahl von gleichzeitig darstellbaren Farben gibt und daß man diesen Farbregistern bestimmte – relativ frei wählbare – Farbtöne zuordnen kann.

Diese Eigenschaft ist jedoch nicht selbstverständlich und kann auch mit Hilfe von "vq_extnd()" erfragt werden. Zwei Beispiele für Grafikmodi ohne "Color-Lookup":

1. Der TT-Grafikmodus "TT-Hoch": es gibt nur zwei Farbregister und zwei Farbtöne (Schwarz und Weiß), die auch nicht vertauscht werden können.

2. Bei "True-Color"-Karten können "beliebig" viele Farben gleichzeitig dargestellt werden. Normalerweise werden 24 Bits pro Pixel benutzt – es stehen also etwa 16 Millionen Farben zu Verfügung.

   Bei einer derartigen Anzahl von Farbnummern hat sich die Idee von Farbregistern erledigt: die Anzahl der prinzipiell gleichzeitig darstellbaren Farbtöne übersteigt die typische Zahl von Bildpunkten bei weitem. Schon in Kürze soll das VDI solche Grafikkarten unterstützen.

Eine weitere unbeantwortete Frage bleibt: Auf welche Weise hängen die Pixelwerte im Bildspeicher mit VDI-Farbnummern zusammen?

1. Aus historischen Gründen darf man davon ausgehen, daß bei einem Pixel in VDI-Farbe 0 (normalerweise Weiß) alle Bits gelöscht sind. Analog dazu sind bei schwarzen Bildpunkten (VDI-Farbe 1) alle Pixel gesetzt.

   Diese Regel ergibt sich schon dadurch, daß durch ein vollständiges Invertieren aller Bits der Pixeldarstellung Weiß und Schwarz vertauscht werden müssen.

2. Mit "v_get_pixel()" kann man für eine gegebene Bildkoordinate sowohl VDI-Farbnummer als auch Pixelwert erfragen.

Zusätzlich haben Digital Research und Atari die Standardzuordnungen für 8- und 16-Farb-Grafikstufen dokumentiert (man achte speziell auf die Zuordnung von VDI-Farbe 1!).

Bei acht Farben:

Pixelwert	Farbindex	Farbe	Name
000	0	weiß (Hintergrund)	WHITE
001	2	rot	RED
010	3	grün	GREEN
011	6	gelb	YELLOW
100	4	blau	BLUE
101	7	magenta	MAGENTA
110	5	cyan	CYAN
111	1	schwarz	BLACK

Bei 16 Farben:

Pixelwert	Farbindex	Farbe	Name
0000	0	weiß (Hintergrund)	WHITE
0001	2	rot	RED
0010	3	grün	GREEN
0011	6	gelb	YELLOW
0100	4	blau	BLUE
0101	7	magenta	MAGENTA
0110	5	cyan	CYAN
0111	8	hellgrau	LWHITE
1000	9	dunkelgrau	LBLACK
1001	10	dunkelrot	LRED
1010	11	dunkelgrün	LGREEN
1011	14	dunkelgelb	LYELLOW
1100	12	dunkelblau	LBLUE
1101	15	dunkelmagenta	LMAGENTA
1110	13	dunkelcyan	LCYAN
1111	1	schwarz	BLACK

Als Datenstruktur zur einheitlichen Beschreibung von Rastern dient die MFDB-Struktur ("Memory Form Definition Block"):

```
typedef struct
{
     void *fd_addr;       /* Zeiger auf Speicherblock (muß auf gera-
                             der Adresse liegen). Für den Bildspei-
                             cher muß man 0 übergeben, alle anderen
                             Werte werden dann ignoriert */
     WORD fd_w;           /* Breite des Speicherblocks in Punkten */
     WORD fd_h;           /* Höhe des Speicherblocks in Punkten */
     WORD fd_wdwidth;     /* Breite des Speicherblocks in 16-Bit-
                             Worten */
     WORD fd_stand;       /* 0: geräteabhängiges Format,
                             1: Standardformat */
     WORD fd_nplanes;     /* Anzahl der Bildebenen */
     WORD fd_r1,fd_r2, fd_r3; /* reserviert */
} MFDB;
```

Eine MFDB-Struktur beschreibt einen "rechteckigen" Speicherblock – entweder im Arbeitsspeicher oder im Bildschirm. Wenn man den Bildspeicher meint, *muß* man "fd_addr" auf 0 setzen. Alle anderen Werte werden dann ignoriert. Die VDI-Rasterfunktionen arbeiten *ausschließlich* dann mit Clipping, wenn "fd_addr" 0 ist.

Anderenfalls beschreibt der MFDB einen für die CPU zugänglichen Speicherblock. Dann, und nur dann, muß man die restlichen Werte des MFDB mit sinnvollen Werten füllen.

Die folgende Beispielfunktion ermittelt den für einen rechteckigen Bildausschnitt benötigten Speicherplatz und besetzt schon die wichtigsten Felder des MFDB vor:

```
ULONG RastSize (WORD handle, WORD width, WORD height, MFDB *p)
{
     WORD work_out[57];
     WORD width_wd;

     vq_extnd (handle, 1, work_out);     /* erweiterte Attribute
                                            erfragen */
     width_wd = (width + 15) / 16;       /* Anzahl der benötigten
                                            WORDs */
     p->fd_w = width;
     p->fd_h = height;
     p->fd_wdwidth = width_wd;
     p->fd_nplanes = work_out[4];
     p->fd_stand = 0;
```

```
return (((ULONG) height) *          /* Bildschirmzeilen */
     ((ULONG) width_wd * 2) *       /* Bytes pro Zeile */
     ((ULONG) work_out[4]));        /* Planes */
}
```

GDOS

Das VDI – eine Illusion?

Bislang wurde immer nur ganz abstrakt von "dem" VDI gesprochen. Daß die eigentliche Arbeit von den *Gerätetreibern* erledigt wird, wurde auch schon erwähnt.

Bleibt die Frage: was bleibt vom VDI übrig, wenn man die Gerätetreiber wegnimmt? Antwort: es bleibt GDOS ("Graphics Device Operating System"), die Schaltzentrale, die für die Verwaltung der einzelnen Gerätetreiber, der nachladbaren Zeichensätze und der verschiedenen Koordinatensysteme zuständig ist.

Alle diese Fähigkeiten gehen dem im ROM verankerten VDI ab, weil es eben kein GDOS enthält. GDOS muß daher normalerweise nachträglich als residentes AUTO-Ordner-Programm installiert werden.

Die Gründe hierfür sind primär in den Platzproblemen zu suchen, die Atari bei der Zusammenstellung der ersten TOS-ROMs plagten. Was ursprünglich als Lücke galt, darf heute jedoch eher als Glücksfall eingestuft werden: kaum ein anderer Betriebssystemteil hat von Außenstehenden soviel Weiterentwicklung erfahren – gerade *deshalb*, weil es so leicht durch eine verbesserte Version zu ersetzen ist. (Ein gutes Beispiel ist das "AMC-GDOS" von VDI-Insider Arnd Beissner, das bis einschließlich Version 3.21 für den nichtkommerziellen Einsatz frei verfügbar und in fast allen Mailboxen zu finden ist.)

Und so ist es vermutlich auch zu erklären, warum GDOS auch in den neuesten TOS-Versionen nicht im ROM untergebracht ist.

Die Kehrseite der Medaille ist allerdings auch schnell beschrieben:

- für Nicht-Insider komplizierte Installation
- ursprünglich sehr lückenhafte Informationen
- sehr viele ältere Programme laufen bzw. liefen nicht korrekt, wenn GDOS installiert ist

Ein weiterer Grund für die ursprünglich schlechte Akzeptanz von GDOS war die unübersehbare Bremsung aller VDI-Funktionen, die Folge einer bemerkenswert ineffizienten Programmierung des Dispatchers ist. GDOS-Versionen von Fremdanbietern und künftige GDOS-Versionen von Atari haben dieses Problem allerdings nicht mehr (oder zumindest in einem erheblich geringerem Maße).

Ein anderes Problem ist natürlich die Verfügbarkeit. Wer GDOS zur Zeit in einem kommerziellen Programm einsetzen will, muß es mitliefern und dazu bei Atari eine Lizenz erwerben (gleiches gilt natürlich auch dann, wenn man das GDOS eines anderen Anbieters mitliefern möchte). Die von Atari geforderte Lizenzsumme ist allerdings so niedrig, daß auch kleine Softwarehäuser dadurch vor kein Problem gestellt werden. Genauere Informationen dazu erhalten Sie bei Ataris Software-Support.

Gerätetreiber

Jedem im System befindlichen Gerät ist ein eigener Treiber (mit einer speziellen Treiberdatei) zugeordnet. Der Bildschirmtreiber im ROM nimmt nur insofern eine Sonderrolle ein, als sein Programmcode eben im ROM und nicht in einer Datei vorliegt.

Jeder einzelne Gerätetreiber muß alle diejenigen VDI-Funktionen übernehmen, für die nicht das GDOS zuständig ist, und enthält also in etwa die gleiche Funktionalität wie der ROM-Bildschirmtreiber. Über Aufbau und Funktionen der verschiedenen Gerätetreiber gibt ein eigenes Kapitel Auskunft.

Zeichensätze

Das VDI kennt generell zwei verschiedene Klassen von Zeichensätzen. Damit ist allerdings noch *keine* Aussage darüber gemacht, welcher Art diese Zeichensätze sind – das ist vom jeweiligen Treiber bzw. dem verwendeten GDOS abhängig.

Zunächst einmal gibt es die sogenannten *Systemzeichensätze* – das sind solche Fonts, die das Gerät "von Haus aus" kennt. Den Systemzeichensatz des ROM-Bildschirmtreibers kennt jeder – man hat ihn permanent bei der Arbeit mit GEM vor Augen (auch, wenn es den Systemzeichensatz in mehreren Größen gibt, handelt es sich doch eben nur um eine *Schrift* mit einer gemeinsamen Zeichensatznummer). Die Anzahl der verfügbaren Systemzeichensätze wird beim Öffnen der Workstation zurückgeliefert und muß *nicht* gleich 1 sein!

Wenn GDOS installiert ist, kann man auf *zusätzliche* Zeichensätze zugreifen, die bei einem Aufruf von "vst_load_fonts()" geladen werden. Welcher Art diese Zeichensätze sind, ist nicht

dokumentiert und vom verwendeten GDOS abhängig. Aus der Sicht eines Anwendungsprogramms verhalten sich auf jeden Fall alle verfügbaren Zeichensätze gleich.

Koordinatensysteme

Die Unterstützung des NDC-Koordinatensystems ist eine weitere Aufgabe von GDOS. Dazu merkt es sich für jede geöffnete Workstation, ob RC- oder NDC-Koordinaten benutzt werden sollen, und kümmert sich bei jedem VDI-Aufruf um die Konvertierung von Koordinaten und Maßen.

Künftige Weiterentwicklungen

Im PC-Bereich sind am VDI diverse Erweiterungen vorgenommen worden. Einige Beispiele:

- Mehr Optionen beim Öffnen von Workstations (Stichwort: Papierformate bei Druckertreibern).
- Unterstützung von Grau- und Farbverläufen.
- Bézierkurven (realisiert durch einen "globalen" Schalter, der zwischen Polyline- und Bézierdarstellung von Linien und Polygonen umschaltet).
- Erweiterungen für "True-Color"-Grafik (also Farbgrafikkarten mit 24 Bit pro Pixel).

Ob und welche dieser Erweiterungen jemals Eingang in das Atari-GDOS bzw. die Gerätetreiber finden wird, ist heute schwer abzusehen. Sicher scheint, daß künftige GDOS-Versionen zumindest PC-GEM-kompatible Béziererweiterungen enthalten werden – einige Fremdhersteller bieten schon jetzt solche GDOS-Versionen an (zum Beispiel: AMCGDOS 5.0).

Von Atari seit Herbst 1990 angekündigt ist *FSMGDOS*, eine GDOS-Version, die Postscript-ähnliche Vektorschriften unterstützen soll. Hoffentlich ist die Entwicklung bald abgeschlossen, denn erst mit FSMGDOS wird das VDI die Flexibilität bei der Textausgabe haben, die man sich wünscht (identische Zeichensätze für alle Ausgabegeräte, freie Skalier- und Rotierbarkeit, ökonomischere Nutzung des Speicherplatzes).

Keine der beiden Entwicklungen war bei Redaktionsschluß fertiggestellt und veröffentlicht.

Daher müssen wir für weitere Informationen leider auf künftige Auflagen des Profibuchs, die Fachzeitschriften und den Entwickler-Support von Atari verweisen.

Technische Details

Informationen über GDOS erfragen

In einigen Fällen kann es wichtig sein, zu wissen, ob überhaupt ein vollständiges GDOS installiert ist (zum Beispiel, bevor man einen "vst_load_fonts()"-Aufruf macht). Glücklicherweise hat Atari beschrieben, wie man in dieser Frage vorzugehen hat. Die Funktion beruht auf der Tatsache, daß der Trap-Dispatcher den Eingabewert –2 für Register D0 genau dann verändert, wenn ein GDOS installiert ist (*Vorsicht*: Die GEM-Versionen des niederländischen Softwarehauses "ABC" (ABC-GEM 2.x) stürzen bei diesem Aufruf ab).

Bei den meisten Entwicklungssystemen ist diese Funktion bereits in die VDI-Bibliotheken aufgenommen worden. Hier ein Beispiel-Binding in Alcyon-C:

```
/* liefert 0, falls GDOS nicht installiert ist */
WORD vq_gdos()
{
     asm("   move.w  #-2,d0  ");
     asm("   trap    #2      ");
     asm("   cmp.w   #-2,d0  ");
     asm("   sne     d0      ");
     asm("   ext.w   d0      ");
}
```

Zeichensatzformat

Alle zur Zeit erhältlichen GDOS-Versionen können Zeichensätze im DR-Standardformat für Bitmap-Zeichensätze laden. Bei diesem Format wird der Zeichensatz einfach wie ein sehr breites monochromes Rasterbild organisiert – alle Zeichen stehen in Reih und Glied nebeneinander. Die Breite des Rasters ("form width") ist die Summe aller Zeichenbreiten, die Rasterhöhe ("form height") entspricht der Höhe eines einzelnen Zeichens.

Unmittelbare Konsequenz: Der linke Rand eines Zeichens fällt *nicht* unbedingt auf eine Bytegrenze! Nur am Ende jeder Rasterzeile wird mit Nullbits soweit aufgefüllt, bis der nächste Zeilenbeginn wieder auf eine Wortgrenze fällt.

Da das VDI ursprünglich auf dem PC entwickelt wurde, liegen normalerweise alle Daten im Intel-Format vor. Es müssen also in jedem einzelnen 16-Bit-Wort des Zeichensatzkopfes das obere und das untere Byte vertauscht werden.

Auch ein Motorola-Format (ohne Bytevertauschung) ist definiert und kann anhand eines Flags im Dateikopf unterschieden werden. Zeichensätze im Motorola-Format werden zwar ROM-intern für die Systemzeichensätze benutzt, können aber von den meisten GDOS-Versionen nicht gelesen werden. Der Dateiheader für einen solchen Zeichensatz sieht wie folgt aus:

```
typedef struct
{
     WORD      font_id;        /* Fontnummer */
     WORD      point;          /* Größe im Punktmaß */
     CHAR      name[32];       /* Name des Zeichensatzes */
     UWORD     first_ade;      /* Erstes Zeichen im Zeichensatz (nur
                                  die wenigsten Druckerschriften
                                  enthalten Control-Zeichen, also
                                  ASCII-Werte kleiner 32) */
     UWORD     last_ade;       /* Letztes Zeichen im Zeichensatz */
     UWORD     top;            /* Abstand Topline<->Baseline */
     UWORD     ascent;         /* Abstand Ascentline<->Baseline */
     UWORD     half;           /* Abstand Halfline<->Baseline */
     UWORD     descent;        /* Abstand Descentline<->Baseline */
     UWORD     bottom;         /* Abstand Bottomline<->Baseline */
     UWORD     max_char_width;
                               /* größte Zeichenbreite */
     UWORD     max_cell_width;
                               /* größte Zeichenzellenbreite */
     UWORD     left_offset;    /* linker Offset für Kursivschrift */
     UWORD     right_offset;   /* rechter Offset für Kursivschr. */
     UWORD     thicken;        /* Verbreiterungsfaktor für
                                  Fettschrift (bei Fettschrift wird
                                  die Breite um genausoviele Pixel
                                  erhöht) */
     UWORD     ul_size;        /* Dicke der Unterstreichung */
     UWORD     lighten;        /* Maske für helle Schrift (wird mit
                                  Raster undiert, normalerweise
                                  0 x 5555) */
     UWORD     skew;           /* Maske für Kursivschrift (be-
                                  schreibt, an welchen Stellen die
                                  Rasterzeilen verschoben werden
                                  sollen, normalerweise 0 x 5555 */
     UWORD     flags;          /* Bit 0: gesetzt, wenn es der
                                         "Default system font" ist
                                  Bit 1: gesetzt, wenn die "Hori-
```

```
                                        zontal offset table"
                                        benutzt wird
                                Bit 2: gesetzt, wenn Bytes nicht
                                        vertauscht werden müssen
                                        (Motorola-Format)
                                Bit 3: gesetzt, wenn der Font
                                        nicht proportional ist */
    UBYTE     *hor_table;     /* Zeiger auf "Horizontal Offset
                                Table"; in der Datei der Offset
                                zum Dateibeginn */
    UWORD     *off_table;     /* Zeiger auf "Character Offset
                                Table" */
    UWORD     *dat_table;     /* Zeiger auf Fontimage */
    UWORD     form_width;     /* Breite des Zeichensatzimage */
    UWORD     form_height;    /* Höhe des Zeichensatzimage */
    FONT_HDR  *next_font;     /* Zeiger auf nächsten Fontheader (in
                                Fontdatei zunächst unbenutzt) */
} FONT_HDR;
```

Bleibt ein Problem: Um das Motorola/Intel-Flag abfragen zu können, muß man eigentlich bereits wissen, welches Format der Zeichensatz hat (denn im Intel-Format liegen alle 16-Bit-Worte des Headers "vertauscht" vor!). Einzige Möglichkeit: Man geht davon aus, daß Bit 10 des Flags niemals benutzt sein wird und testet, ob Bit 2 im 67. Byte des Headers gesetzt ist – dann liegt der Font im Motorola-Format vor!

Die "Character Offset Table" ist eine Tabelle von 16-Bit-Werten, die den horizontalen Pixeloffset für jedes Zeichen innerhalb des Fontrasters angibt.

Als Index muß man also den ACSII-Code *minus* den ACSII-Code des ersten Zeichens im Zeichensatz benutzen ("first_ade"). Die Breite eines Zeichens ergibt sich aus der Differenz zum Offsetwert des nächsthöheren Zeichens (damit diese Formel auch für das letzte Zeichen funktioniert, enthält die Tabelle immer einen Eintrag mehr, als Zeichen da sind).

Die "Horizontal Offset Table" wird nur bei wenigen Zeichensätzen benutzt und enthält positve oder negative Offsetwerte, die vor der Ausgabe eines Zeichens auf die X-Position addiert werden.

Abschließend sei nochmals darauf hingewiesen, daß sich eine normale VDI-Anwendung *niemals* mit diesem Format befassen muß. Die Beschreibung ist nur für diejenigen wichtig, die Zeichensatzeditoren oder -generatoren entwickeln oder die Welt mit einer weiteren neuen GDOS-Version beglücken wollen.

Zeichensatznamen

Atari selbst liefert auf der GDOS-Systemdiskette folgende Zeichensätze mit:

Fontnummer	Name
02	Swiss (Helvetica-Schrift, Dateien "ATSS*.FNT")
14	Dutch (Times-Schrift, Dateien "ATTR*.FNT")
15	Typewriter (Courier-Schrift, Dateien "ATTP*.FNT")

Auch die Bedeutung der Dateinamen ist dokumentiert, wird aber leider nicht einheitlich benutzt:

Zeichen	Belegung
0..1	Hersteller des Zeichensatzes (bei Ataris Zeichensätzen "AT")
2..3	Kürzel für den Zeichensatztyp (z. B. "SS" für "Sans Serif", "TR" für "Times Roman", "TP" für "Typewriter")
4..5	Punktgröße des Zeichensatzes
6..7	Typ des Ausgabegeräts (z. B. "EP" für Epson-9-Nadeldrucker, "LS" für Laserdrucker, "CG" für die 640*200-Auflösung oder leer für Standard-Bildschirmfonts)

Die ASSIGN.SYS-Datei

Diese Textdatei enthält Informationen über die vorhandenen Gerätetreiber und die Zeichensätze, die ihnen zugeordnet sind. GDOS sucht sie bei Systemstart im Wurzelverzeichnis des Bootlaufwerks.

Hier zunächst eine Beispieldatei:

```
; Beispiel für ASSIGN.SYS-Datei
path = C:\gemsys

01p screen.sys ; Bildschirmtreiber (ROM)
ATSS10.FNT
ATSS12.FNT
ATSS18.FNT
ATSS24.FNT
ATTR10.FNT
```

```
ATTR12.FNT
ATTR18.FNT
ATTR24.FNT
ATTP10.FNT

02p screen.sys ; Bildschirmtreiber (ROM)

ATSS10.FNT
ATSS12.FNT
ATSS18.FNT
ATSS24.FNT
ATTR10.FNT
ATTR12.FNT
ATTR18.FNT
ATTR24.FNT
ATTP10.FNT

03p screen.sys ; Bildschirmtreiber (ROM)

ATSS10CG.FNT
ATSS12CG.FNT
ATSS18CG.FNT
ATSS24CG.FNT
ATTR10CG.FNT
ATTR12CG.FNT
ATTR18CG.FNT
ATTR24CG.FNT
ATTP10CG.FNT
ATTP10CG.FNT

04p screen.sys ; Bildschirmtreiber (ROM)

ATSS10.FNT
ATSS12.FNT
ATSS18.FNT
ATSS24.FNT
ATTR10.FNT
ATTR12.FNT
ATTR18.FNT
ATTR24.FNT
ATTP10.FNT
```

```
21r fx80.sys ; Epson FX80, resident

ATSS10EP.FNT
ATSS12EP.FNT
ATSS18EP.FNT
ATSS24EP.FNT
ATTR10EP.FNT
ATTR12EP.FNT
ATTR18EP.FNT
ATTR24EP.FNT
ATTP10EP.FNT

31 meta.sys ; Metafile-Treiber

ATSS10MF.FNT
ATSS12MF.FNT
ATSS18MF.FNT
ATSS24MF.FNT
ATTR10MF.FNT
ATTR12MF.FNT
ATTR18MF.FNT
ATTR24MF.FNT
ATTP10MF.FNT
```

Kommentare werden mit einem Semikolon eingeleitet und können an einer beliebigen Stelle in der Textzeile beginnen – auf einem Atari TT würden noch zusätzliche Einträge für die Geräte 5 bis 10 benötigt. Die erste Zeile, die keinen *Kommentar* enthält, darf einen Eintrag der Form

```
path = pfadname
```

enthalten.

GDOS sucht dann Treiber und Zeichensätze in dem dort angegebenen Verzeichnis (Maximallänge: 64 Zeichen!).

Es folgen die Einträge für die Gerätetreiber und die ihnen zugeordneten Zeichensätze.

Jeder Gerätetreiberformat hat das Format:

```
<Nummer><Flag> <Dateiname des Treibers>
```

Die Nummer entspricht der beim Öffnen der Workstation übergebenen Gerätenummer. Man beachte, daß für jede mögliche Gerätenummer auch ein Treiber angemeldet sein muß. Die AES benutzen zur Berechnung der zu benutzenden Bildschirmworkstation die Formel:

```
2 + Getrez ();
```

Als Flag darf "r" (Treiber gleich beim Booten resident laden) oder "p" (Treiber ist im ROM installiert) oder auch nichts (Treiber beim Öffnen der Workstation nachladen) angegeben werden. Bei gesetztem "p"-Flag ist der Treibername nur ein Platzhalter, den GDOS braucht, um die Zeile dekodieren zu können.

Es folgen die Einträge für die einzelnen verfügbaren Zeichensätze. Man beachte dabei, daß Fonts der gleichen Familie (also mit gleicher Fontnummer, aber verschiedener Punktgröße) direkt aufeinander folgen sollten.

Übersicht über die VDI-Bibliotheken

Die VDI-Funktionen sind in sieben Bereiche unterteilt. Diese Unterteilung ist nicht immer ganz konsequent – aus historischen Gründen wollen wir es aber dabei belassen. Der Meinung, die von Digital Research vergebenen Bezeichnungen seien allzu kryptisch, können wir uns nicht anschließen, man muß sich eben nur an sie gewöhnen.

Im allgemeinen kann man nämlich aus den Buchstaben vor dem ersten Unterstrich schon die Funktion gut einordnen. Beispiel: "vst_" steht für "VDI SET TEXT".

Kontrollfunktionen

Diese Funktionen dienen zur Kontrolle der Workstations – Öffnen und Schließen, Laden und Freigeben von Zeichensätzen, Setzen des Clippings und verwandte Informationen. Faustregel: Bei den meisten dieser Funktionen spielt GDOS eine Rolle.

Ausgabefunktionen

Mit Hilfe dieser Funktionen werden die Grafikgrundformen ausgegeben.

Attributfunktionen

Jede Grafikgrundform ist in verschiedenen Darstellungen verfügbar. Man kann mit den Attributfunktionen Farbe, Linienbreite und vieles mehr wählen. Mehrere Attributfunktionen können zu Füll-, Linien-, Marker- und Text-Attributen zusammengefaßt werden.

Raster-Operationen

Die Raster-Operationen bieten die Möglichkeit, Raster unter Anwendung verschiedener Verknüpfungen im Bildspeicher, im Arbeitsspeicher oder zwischen beiden Bereichen hin und her zu kopieren. Ferner können Raster konvertiert und die Belegung einzelner Pixel abgefragt werden.

Eingabefunktionen

Mit diesen Funktionen können die verschiedenen Eingabegeräte (wie Maus oder Tastatur) abgefragt werden. Die Eingabefunktionen sind zwar ein Bestandteil des Bildschirmtreibers, dürfen aber nur auf der *physikalischen* Workstation eingesetzt werden.

Eine Eigenheit der Eingabefunktionen sind die "doppelten" Bindings für viele Funktionen: Je nach eingeschalteter Betriebsart (Request oder Sample) werden unterschiedliche Funktionsnamen benutzt – auch wenn der benutzte Opcode gleich ist.

Auskunftsfunktionen

Mit diesen Funktionen kann man jederzeit gesetzte Attribute oder andere Parameter abfragen.

Escapes

Spezielle Fähigkeiten des Grafikgerätes können mit diesen Funktionen ausgenutzt werden. Auf dem Textbildschirm etwa kann man mit den Escape-Funktionen eine Hardcopy auslösen.

VDI-Bindings

Der VDI-Parameterblock

Das VDI benutzt zur Parameterübergabe nicht die Prozessorregister oder den Stack, sondern mehrere Ein- und Ausgabefelder. Im einzelnen sind das:

```
WORD contrl[12];
```

Dieses Feld enthält folgende Eingabeparameter:

contrl[0]: Opcode der Funktion (*Vorsicht*: kann sich über den ganzen Bereich von –32768 bis 32767 erstrecken!)

contrl[1]: Anzahl der Eingabekoordinatenpaare
contrl[3]: Anzahl der Werte im intin-Array
contrl[5]: Identifikationsnummer für Unter-Opcode
contrl[6]: Workstation-Handle
contrl[7..]: Mögliche weitere Werte, abhängig von der Funktion

Zurück erhält man folgende Rückgabewerte:

contrl[2]: Anzahl der Ausgabekoordinatenpaare
contrl[4]: Anzahl der Werte im intout-Array
contrl[6]: Workstation-Handle (beim Öffnen der Workstation)
contrl[7..]: Mögliche weitere Werte, abhängig von der Funktion

```
WORD intin[...];
WORD intout[...];
```

Diese beiden Felder enthalten Ein- bzw. Ausgabeparameter – und zwar alle die, die keine Maße oder Koordinaten beschreiben. Auch einzelne Zeichen und Zeichenketten werden hier übergeben (indem die einzelnen Zeichen auf 16-Bit-Werte expandiert werden). Wie viele Werte maximal übergeben werden können, hängt vom jeweiligen Treiber ab und kann mit "vq_extnd()" erfragt werden.

```
WORD ptsin[...];
WORD ptsout[...];
```

Diese beiden Felder enthalten alle Parameter, die Maß- oder Koordinatencharakter haben. Sie werden deshalb unabhängig von "normalen" Parametern geführt, weil bei ihnen gegebenfalls die Koordinatentransformation zwischen NDC- und RC-System durchgeführt werden muß. Die maximal zu übergebende Anzahl von Koordinatenpunkten ist nicht nur von wissenschaftlicher Bedeutung, sondern legt die maximale Punktzahl für Linienzüge und Polygone fest. Wie viele Koordinatenpunkte (also Wortpaare!) übergeben werden können, läßt sich mittels "vq_extnd()" erfragen. Die Adressen der fünf Parameterfelder werden als Zeiger im VDI-Parameterblock abgelegt:

Adresse	Inhalt
PB	Adresse des contrl-Arrays
PB+4	Adresse des intin-Arrays
PB+8	Adresse des ptsin-Arrays
PB+12	Adresse des intout-Arrays
PB+16	Adresse des ptsout-Arrays

Der VDI-Trap

Zum Aufruf einer VDI-Funktion lädt man Datenregister D0 mit der Konstante 115, D1 mit der Anfangsadresse des VDI-Parameterblocks und macht einen "TRAP #2"-Aufruf. Darüber, welche Register verändert werden, gibt es keine klaren Informationen. Tatsache aber ist, daß die entsprechenden Routinen im ROM *alle* Register retten.

Leider gibt es keinen dokumentierten Return-Code für "Unbekannte Funktionsnummer". Im Zweifelsfall muß man sich also folgendermaßen behelfen:

1. Die "Extended Inquire Function" bietet bereits etliche Informationen über den angesprochenen Treiber an.

2. Ansonsten muß man den fraglichen VDI-Aufruf ausprobieren und dann die Rückgabewerte analysieren. Viele Funktionen liefern als Resultat den eingestellten Wert zurück. Daneben kann man auch untersuchen, ob contrl[2] und contrl[4] die korrekten Zahlen enthalten.

Hier noch eine Übersicht über die möglichen Opcodes (also Werte für D0.W), die vom Dispatcher verstanden werden:

Opcode	Funktion
–2	Abfrage, ob GDOS vorhanden ist
–1	Nicht dokumentiert, aber vom GDOS benötigt: bei diesem Aufruf liefert der ROM-Bildschirmtreiber die Adresse einer Funktion zurück, die das GDOS dann – ganz wie bei einem "normalen" Bildschirmtreiber – per JSR aufrufen kann. Wer sich in den VDI-Trap einklinken will, *muß* auch diese Funktionsnummer abfangen, damit ein eventuell später installiertes GDOS wie beabsichtigt funktioniert.
115	VDI-Aufruf
200	AES-Aufruf

Die Beispiel-Bindings

Bei den hier vorliegenden Beispiel-Bindings des Alcyon-Compilers hat man es sich recht einfach gemacht und den Parameterblock kurzerhand folgendermaßen deklariert:

```
int *pooff,*iooff,*pioff,*iioff,*pblock;
```

Daß ein C-Compiler diese fünf Pointer direkt hintereinander im Speicher anlegt, ist nicht überraschend. Daß bei dieser Deklaration aber auch noch die richtige (nämlich umgekehrte) Reihenfolge herauskommt, dürfte ein implementationsabhängiger Zufall sein – dieser Programmierstil ist also nicht zur Nachahmung empfohlen!

Was noch fehlt, ist die Initialisierung des Parameter-Blocks. Die letzten vier Adressen werden von "OPEN WORKSTATION" bzw. "OPEN VIRTUAL WORKSTATION" eingesetzt, während die Funktion "vdi()" den ersten Wert setzt (nämlich "pblock", der den Zeiger auf das contrl-Array enthält).

Die Funktion "vdi ()" hat eine vergleichsweise einfache Aufgabe:

- Die Adreßregister A0 und A1 werden gerettet.
- Die Anfangsadresse des contrl-Arrays wird in "pblock" eingetragen.
- Die Adresse von "pblock" (also die Anfangsadresse des VDI-Parameterblocks) wird in Datenregister D1 geladen.
- Datenregister D0 wird mit der Zahl "115" geladen.
- Abschließend wird über "TRAP #2" GEM aufgerufen, und es werden noch die beiden zuvor geretteten Adreßregister wiederhergestellt.

Kein Teil des ST/TT-Betriebssystems ohne Ausnahmen: einige wenige Funktionen erwarten oder liefern 32-Bit-Werte. Da alle Ein- und Ausgabefelder jeweils nur 16-Bit-Wort-Felder sind, muß man mit einem Kunstgriff die Werte auf zwei 16-Bit-Variablen verteilen. Dazu werden hier die Funktionen "i_ptr" (schreibt den angegebenen Wert in contrl[7] und contrl[8]), "i_ptr2" (dasselbe mit contrl[9] und contrl[10]), sowie "m_lptr2" (überträgt contrl[9] und contrl[10] in die angegebene Adresse) benutzt.

VDI-Gerätetreiber

Einleitung

Ein Gerätetreiber ist eine völlig normale Programmdatei ohne STARTUP-Code (z.B. Speicherreservierung), die als erste Routine im Textsegment (also an der "Einsprungadresse") einen "Dispatcher" für die eingehenden VDI-Aufrufe enthält:

- Die Adresse des VDI-Parameterblocks wird in Register D1 übergeben.

- Die Funktion wird mit einer RTS-Instruktion abgeschlossen.

Folgendes muß man noch bei der Implementation eines Gerätetreibers beachten: Nicht alle Funktionen erreichen den eigentlichen Treiber genauso, wie sie abgesandt werden, sondern sie werden vom GDOS "vorverarbeitet". Im einzelnen sind dies:

- Das Laden der Zeichensätze. GDOS lädt alle in der ASSIGN.SYS-Datei angegebenen Zeichensätze (sofern der Speicher reicht), "reloziert" die einzelnen Zeiger und verkettet alle Header über das "next_font"-Feld zu einer linearen Liste. Der Treiber erhält diese vorverarbeiteten Zeichensätze im Rahmen eines "vst_load_fonts()"-Aufrufs mit der folgenden, vom normalen Gebrauch *abweichenden* Parameterbelegung:

  contrl[7/8] Adresse des "Scratch"-Puffers zur Berechnung von Texteffekten (wie kursiv oder fett).

  contrl[9] Länge dieses Puffers in 16-Bit-Worten.

  contrl[10/11] Anfangsadresse des ersten Fontheaders in der Zeichensatzliste.

  Dieses Interface zum Laden der Zeichensätze funktioniert also *ausschließlich* dann, wenn GDOS *nicht* installiert ist!

- Der Treiber arbeitet *immer* im Rasterkoordinatensystem. Die Konvertierung der Werte im ptsin- und ptsout-Feld wird vom GDOS übernommen.

Bildschirmtreiber

Besonderheiten

Bei den aktuellen TOS-Versionen findet sich der VDI-Bildschirmtreiber im ROM, kann aber durch einen vom GDOS nachzuladenden Treiber ersetzt werden.

Derartige Treiber existieren mittlerweile für diverse Grafikkarten, beispielsweise die des Herstellers "Matrix". Man beachte, daß bei einigen auf dem Markt befindlichen Treibern Funktionen fehlerhaft implementiert sind bzw. ganz fehlen.

(*Hinweis*: Erkundigen Sie sich vor einer eventuellen Anschaffung nach der vollen Unterstützung von GEM-Zeichensätzen.)

Der ROM-Bildschirmtreiber selbst weist folgende Merkmale auf:

- Die zu "CELL ARRAY" gehörigen Funktionen sind nicht implementiert (sind laut Digital Research aber auch optional).

- Einige der Eingabefunktionen arbeiten nicht fehlerfrei (was keine schlimme Einschränkung ist, da sie von den AES nicht benötigt werden).

- Das ROM-VDI basiert auf dem Kern der sogenannten "Line-A"-Funktionen, über die das VDI sämtliche Ausgaben abwickelt (mehr dazu später).

- Die Escape-Routinen zur Bildschirmausgabe sind mit den vom BIOS für "CON:" und "RAWCON:" benutzten Routinen identisch (können allerdings nicht wie ihre ihre BIOS-Gegenstücke "umgelenkt" werden).

Mindestfunktionsumfang

(Spezifikation nach Atari, "GEM Programmer's Guide")

Opcode	Funktion
1	Open workstation
2	Close workstation
3	Clear workstation
4	Update workstation
5	Escape-functions
1	Inquire addressable character cells
2	Exit alpha mode
3	Enter alpha mode
4	Cursor up
5	Cursor down
6	Cursor right
7	Cursor left
8	Home cursor
9	Erase to end of screen
10	Erase to end of line
11	Direct cursor address
12	Output cursor addressable text
15	Inquire current alpha cursor address
18	Place graphic cursor
19	Remove last graphic cursor

Opcode	Funktion
6	Polyline
7	Polymarker
8	Text
9	Filled area
11	Generalized Drawing Primitive (GDP)
1	Bar
2	Arc
3	Pie
4	Circle
5	Ellipse
6	Elliptical arc
7	Elliptical pie
8	Rounded rectangle
9	Filled rounded rectangle
10	Justified graphics text
12	Set character height, absolute mode
14	Set color representation
15	Set polyline linetype
17	Set polyline color index
18	Set polymarker type
20	Set polymarker color index
21	Set text face
22	Set graphic text color index
23	Set fill interior style
24	Set fill style index
25	Set fill color index
26	Inquire color representation
28	Input locator
31	Input string
32	Set writing mode
33	Set input mode
35	Inquire current polyline attributes
36	Inquire current polymarker attributes
37	Inquire current fill area attributes
38	Inquire current graphic text attributes
39	Set graphic text alignment
100	Open virtual screen workstation
101	Close virtual screen workstation
102	Extended inquire function

Opcode	Funktion
104	Set fill perimeter visibility
106	Set graphic text special effects
107	Set character cell height, points mode
108	Set polyline end styles
109	Copy raster, opaque
110	Transform form
111	Set mouse form
112	Set user-defined fill pattern
113	Set user-defined linestyle
114	Fill rectangle
115	Inquire input mode
116	Inquire text extent
117	Inquire character cell width
118	Exchange timer interrupt vector
121	Copy raster, transparent
122	Show cursor
123	Hide cursor
124	Sample mouse button state
125	Exchange button change vector
126	Exchange mouse movement vector
127	Exchange cursor change vector
128	Sample keyboard state information
129	Set clipping rectangle
130	Inquire face name and index
131	Inquire current face information

Im aktuellen ROM-Bildschirmtreiber zusätzlich implementiert:

Opcode	Funktion
13	Set character baseline vector
16	Set polyline width
19	Set polymarker height
30	Input choice (fehlerhaft)
103	Contour fill
105	Get pixel
119	Load fonts
120	Unload fonts

Line-A-Routinen

Der "harte Kern" des ROM-Bildschirmtreibers ist der sogenannte Line-A-Emulator. Diese Bezeichnung ist allerdings etwas irreführend, da es auf Anhieb nicht unbedingt klar ist, was denn überhaupt emuliert wird... Was macht die CPU des ST, wenn sie auf einen unbekannten Befehl trifft?

Nun, sie löst einen TRAP (Nummer 4, "illegaler Befehl") aus, den der verwirrte Benutzer auf dem Bildschirm durch vier Bömbchen erkennen kann. Eine Ausnahme bilden jedoch die Opcodes, deren oberste vier Bits ein "$A" oder ein "$F" darstellen. Läuft die CPU in einen solchen Opcode, wird wiederum eine spezielle TRAP-Routine aufgerufen. Die unteren 12 Bits des Befehlsworts kann man dann für eigene Informationen verwenden.

Bis TOS 1.04 einschließlich werden F-Opcodes intern durch das Betriebssystem für die verschiedensten Zwecke benutzt, und daher sollte man von ihnen tunlichst die Finger lassen. Die A-Opcodes dienen hingegen zum Aufruf der wichtigsten Grafikroutinen – man kann also sagen, daß die Line-A-Routinen auf Maschinenebene die Befehle eines imaginären Grafik-Chips emulieren.

Die Architektur des Betriebssystems spricht allerdings eindeutig *gegen* die Benutzung der Line-A-Routinen. Diese stellen nämlich die untere Ebene des VDI-Bildschirmtreibers im ROM dar. Mit ihrer Verwendung verbaut man sich also eine eventuelle Nutzung eines anderen (schnelleren) Bildschirmtreibers!

Auch ist eine Existenz der Line-A-Routinen nur für die ST-Modi (also 320 * 200, 640 * 200 und 640 * 400) garantiert. Schon bei 256-Farbgrafik (spezielle Grafikkarte bzw. TT in der "niedrigen" Auflösung) sind die Möglichkeiten der Line-A-Schnittstelle erschöpft (siehe "COLBIT0" bis "COLBIT3").

Wer meint, Line-A-Funktionen im Gegensatz zu VDI-Funktionen im Interrupt aufrufen zu können, hat sich übrigens geirrt: auch die Line-A-Funktionen sind *nicht* re-entrant (einer der Gründe: sie sprechen – falls vorhanden – direkt den Blitter an).

Auch Zeit sparen läßt sich entweder nur wenig oder gar nicht. Ein Beispiel: die Line-A-Funktion "Textblt" kann tatsächlich einzelne Zeichen schneller ausgeben als das VDI – kein Wunder, macht doch der Bildschirmtreiber im ROM auch nur den gleichen Line-A-Aufruf (allerdings per "JSR" statt per Line-A-Ausnahmebehandlung). Normalerweise möchte man jedoch mehrere Zeichen (Wörter oder ganze Zeilen) in einem Stück ausgeben.

Das geht mit "v_gtext()" mit *einem* VDI-Aufruf (eine Exception-Behandlung), während man per Line-A für jedes einzelne Zeichen eine (zeitfressende!) Exception auslösen muß.

Es spricht also alles gegen die Verwendung der Line-A-Routinen. Wer will schon Programme haben, die bei Hardwareänderungen oder neuen Rechnern dank der Verwendung von Line-A-Routinen nicht mehr funktionstüchtig sind?

Zitat aus den TT030 TOS Release Notes: "The Line-A graphics interface is maintained for backward compatibility with existing ST programs only. It should not be used for new programs. It will not keep pace with future hardware or software improvements. The VDI should be used."

Also: Finger weg von den Line-A-Routinen, wenn es sich nicht gerade um ein systemnahes Programm handelt, das sowieso nur auf den drei Standard-Grafikmodi des ST funktionieren muß.

Initialization ($A000)

Mit diesem Aufruf kann man die Anfangsadressen der Line-A-Variablen, der Line-A-Routinen und der Systemzeichensätze erfragen. Initialisiert wird sinnigerweise gar nichts! Zur Parameterübergabe an die Line-A-Routinen wird ein Parameterblock folgender Form verwendet (in Klammern jeweils der Offset zur Basisadresse):

```
typedef struct
{
     WORD PLANES;   /* Anzahl der Bildschirmebenen (+$00) */
     WORD WIDTH;    /* Bytes pro Bildschirmzeile (+$02) */
     LONG CONTRL;   /* Zeiger auf contrl[] (VDI) (+$04) */
     LONG INTIN;    /* Zeiger auf int_in[] (VDI) (+$08) */
     LONG PTSIN;    /* Zeiger auf pts_in[] (VDI) (+$0C) */
     LONG INTOUT;   /* Zeiger auf int_out[] (VDI) (+$10) */
     LONG PTSOUT;   /* Zeiger auf pts_out[] (VDI) (+$14) */
     WORD COLBIT0;  /* Farbwert für Plane 0 (+$18) */
     WORD COLBIT1;  /* Farbwert für Plane 1 (+$1A) */
     WORD COLBIT2;  /* Farbwert für Plane 2 (+$1C) */
     WORD COLBIT3;  /* Farbwert für Plane 3 (+$1E) */
     WORD LSTLIN;   /* letzten Pixel einer Linie zeichnen (1) oder
                       nicht zeichnen (0) +$20) */
     WORD LNMASK;   /* Muster für Linien (+$22) */
     WORD WMODE;    /* VDI-Schreibmodus (+$24) */
     WORD X1,Y1,X2,Y2;
                    /* Koordinatenangaben (+$26) */
     LONG PATPTR;   /* Zeiger auf Füllmuster (+$2E) */
```

```
WORD PATMSK;    /* dazugehörige Maske (+$32) */
WORD MFILL;     /* Füllmuster monochr./farbig (+$34)*/
WORD CLIP;      /* Clipping aus/an (+$36) */
WORD XMINCL,YMINCL;
                /* linke obere Ecke des Clip-rect. (+$38) */
WORD XMAXCL,YMAXCL;
                /* rechte untere Ecke des Clip-rect. (+$3C) */
WORD XDDA;      /* vor Textausgaben auf 0x8000 setzen (+$40) */
WORD DDAINC;    /* Vergrößerungsfaktor - bei Vergrößerung:
                   256*(Wunschgröße-Aktuelle)/Aktuelle; bei
                   Verkleinerung: 56*(Wunschgröße)/Aktuelle
                   (+$42) */
WORD SCALDIR;   /* Vergrößerungsrichtung (0: verkleinern,
                   1: vergrößern (+$44) */
WORD MONO;      /* Proportionalschrift ja/nein (+$46) */
WORD SOURCEX,SOURCEY;
                /* X,Y-Koordinate im Zeichensatz (+$48) */
WORD DESTX,DESTY;
                /* Koordinate des Zeichens auf dem Bildschirm
                   (+$4C) */
WORD DELX,DELY;
                /* Breite und Höhe des Zeichens (+$50) */
FONT_HDR  *FBASE;
                /* Zeiger auf Zeichensatzimage (+$54) */
WORD FWIDTH;    /* Breite des Zeichensatzimage (+$58) */
WORD STYLE;     /* Schreibstil (+$5A)
                   Bit 0:    Fettschrift
                   Bit 1:    Helle Schrift
                   Bit 2:    Kursivschrift
                   Bit 3:    Unterstrichene Schrift
                   Bit 4:    Umrißschrift */
WORD LITEMASK; /* Maske für das Schattieren (light) (+$5C) */
WORD SKEWMASK; /* Maske für Italics (+$5E) */
WORD WEIGHT;    /* zusätzliche Breite bei Bold (+$60) */
WORD ROFF;      /* Kursiv-Offset rechts (+$62) */
WORD LOFF;      /* Kursiv-Offset links (+$64) */
WORD SCALE;     /* Vergrößerung ja/nein (+$66) */
WORD CHUP;      /* Rotationswinkel * 10 (+$68) */
WORD TEXTFG;    /* Textfarbe (+$6A) */
LONG SCRTCHP;   /* Zeiger auf temp. Buffer für Texteffekte
                   (+$6C) */
```

```
    WORD SCRPT2;   /* Offset für denselben (+$70) */
    WORD TEXTBG;   /* Texthintergrundfarbe (+$72) */
    WORD COPYTRAN; /* Flag für "COPY RASTER FORM" (0: "opaque",
                      1: "transparent") (+$74) */
    LONG SEEDABORT;
                   /* Pointer auf Routine, die nach jeder
Bildzeile eines "SEEDFILL" aufgerufen wird (+$76) */
} LINEA;
```

In den Registern D0 und A0 erhält man einen Zeiger auf diesen Parameterblock. In der Literatur werden häufig auch noch "negative" Offsets zur Line-A-Basisadresse aufgeführt. Bei diesen Registern handelt es sich um die lokalen Variablen des Bildschirmtreibers.

Der *Lese-Zugriff* ist durch das Dokument "S.A.L.A.D." offiziell durch Atari freigegeben (dabei waren allerdings die Offsets für die Variablen bis einschließlich "MASK_FORM" falsch angegeben!) – aber eben *ausschließlich* für die drei Standard-Grafikmodi des Atari ST.

Es sei also noch einmal darauf hingewiesen, daß man solche Eigenschaften des *ROM*-Bildschirmtreibers nur dann auszunutzen sollte, wenn das Programm auch sonst ausschließlich über Line-A ausgibt. Auf keinen Fall anstelle der jeweiligen Auskunftsfunktionen des VDI benutzen!!!

```
typedef struct
{
    LONG RESERVED6;     /* (-$38E) */
    FONT_HDR  *CUR_FONT;
                        /* Zeiger auf den Header des aktuellen
                           Zeichensatzes(-$38A) */
    WORD RESERVED5[23]; /* (-$386) */
    WORD M_POS_HX;      /* X-Koordinate des Maus-"Hot spot"
                           (-$358) */
    WORD M_POS_HY;      /* Y-Koordinate des Maus-"Hot spot"
                           (-$356) */
    WORD M_PLANES;      /* Maus wird im Replace-Modus (1) oder im
                           XOR-Modus (-1) gezeichnet (-$354) */
    WORD M_CDB_BG;      /* Maus-Hintergrundfarbe (-$352) */
    WORD M_CDB_FG;      /* Maus-Vordergrundfarbe (-$350) */
    WORD MASK_FORM[32]; /* jeweils abwechselnd für Vordergrund und
                           Maske 16 Words (-$34E) */
    WORD INQ_TAB[45];   /* Informationen, die bei "vq_extnd()"
                           zurückgeliefert werden (-$30E) */
```

```
WORD DEV_TAB[45];   /* Informationen, die man bei "v_opnwk()"
                       erhält (-$2B4) */
WORD GCURX;         /* Aktuelle X-Position der Maus (-$25A) */
WORD GCURY;         /* Aktuelle Y-Position der Maus (-$258) */
WORD M_HID_CT;      /* Anzahl der erfolgten "Hide Mouse"-
                       Aufrufe (-$256) */
WORD MOUSE_BT;   /* Aktueller Status der Mausknöpfe (-$254) */
WORD REQ_COL[48]; /* Interne Daten für "vq_color()" (-$252) */
WORD SIZ_TAB[15];   /* Informationen, die bei "v_opnwk()"
                       zurückgeliefert werden (-$1F2) */
WORD RESERVED4[2];
LONG CUR_WORK;      /* Zeiger auf Attributdaten der aktuellen
                       virtuellen Workstation (-$1D0) */
FONT_HDR  *DEF_FONT;
                    /* Zeiger auf den Standardsystemzeichen-
                       satz (-$1CC) */
LONG FONT_RING[4];  /* Drei Pointer auf Zeichensatzlisten
                       (verkettete FONT_HDR-Strukturen. Das
                       letzte Element enthält eine 0 als
                       Endezeichen (-$1C8) */
WORD FONT_COUNT;    /* Anzahl der Zeichensätze in der
                        "FONT_RING"-Liste (-$1B8) */
WORD RESERVED3[45];
CHAR CUR_MS_STAT;   /* Mausstatus:
                    Bit 0:    linker Knopf
                    Bit 1:    rechter Knopf
                    Bit 2..4: reserviert
                    Bit 5:    Bewegungsflag (1: Maus wurde
                              bewegt)
                    Bit 6:    gesetzt: Status des rechten
                                       Knopfes hat sich
                                       geändert
                    Bit 7:    gesetzt: Status des linken
                                       Knopfes hat sich
                                       geändert (-$15C) */
CHAR RESERVED2;
WORD V_HID_CNT;     /* Anzahl der Hide-Cursor-Aufrufe */
WORD CUR_X;         /* X-Position der Maus (-$158) */
WORD CUR_Y;         /* Y-Position der Maus (-$156) */
CHAR CUR_FLAG;      /* <>0, wenn Mauszeiger beim nächsten
                   VBlank neu gezeichnet werden muß (-$154) */
```

```
CHAR MOUSE_FLAG;     /* <>0, wenn Maus-Interruptbehandlung
                           eingeschaltet ist (-$153) */
LONG RESERVED1;
WORD V_SAV_XY[2];    /* gerettete X- und Y-Koordinate des
                        Cursors (-$14E) */
WORD SAVE_LEN;       /* Anzahl der gebufferten Bildzeilen
                        (-$14A) */
LONG SAVE_ADDR;      /* Adresse des ersten gebufferten Bytes im
                        Bildspeicher (-$148) */
WORD SAVE_STAT;      /* Bit 0: Buffer ist gültig (1) oder
                              ungültig (0)
                        Bit 1: Longs (1) oder Words (0)
                              gebuffert (-$144) */
LONG SAVE_AREA[64]; /* Buffer für Bild unter Mauszeiger
                        ($-142) */
LONG USER_TIM;       /* aktueller Timer-Interrupt-Vektor;
                          sollte zur Beendigung zu "NEXT_TIM"
                        springen (-$42) */
LONG NEXT_TIM;       /* alter Timer-Interrupt-Vektor (-$3E) */
LONG USER_BUT;       /* Maustastenvektor (-$3A) */
LONG USER_CUR;       /* Mausvektor (-$36) */
LONG USER_MOT;       /* Mausbewegungsvektor (-$32) */
WORD V_CEL_HT;       /* Zeichenhöhe (-$2E) */
WORD V_CEL_MX;     /* maximale Cursor-Spaltenposition (-$2C) */
WORD V_CEL_MY;     /* maximale Cursor-Zeilenposition (-$2A) */
WORD V_CEL_WR;       /* Character-Zeilenbreite in Bytes (Höhe *
                        Breite einer Pixelzeile) (-$28) */
WORD V_COL_BG;       /* Hintergrundfarbe (-$26) */
WORD V_COL_FG;       /* Vordergrundfarbe (-$24) */
LONG V_CUR_AD;       /* Adresse der aktuellen Cursorposition
                        auf dem Bildschirm (-$22) */
WORD V_CUR_OF;       /* Vertikaler Offset vom physikalischen
                        Bildschirmanfang (kann mit VDI ESC 101
                        verändert werden) (-$1E) */
WORD V_CUR_XY[2];    /* X- und Y-Position des Cursors (-$1C) */
CHAR V_PERIOD;    /* Blinkgeschwindigkeit des Cursors (-$18) */
CHAR V_CUR_CT;       /* Zähler für Cursorblinken (-$17) */
LONG V_FNT_AD;       /* Zeiger auf Zeichensatzdaten des
                        Systemzeichensatzes (siehe auch VDI ESC
                        102). Die Zeichenbreiten müssen jeweils
                        8 Bytes betragen! (-$16) */
```

```
    WORD V_FNT_ND;      /* ASCII-Wert des letzten Zeichens im
                           Zeichensatz (-$12) */
    WORD V_FNT_ST;      /* ASCII-Wert des ersten Zeichens im
                           Zeichensatz (-$10) */
    WORD V_FNT_WD;      /* Breite der Fontdaten (des Fontimage) in
                           Bytes (-$E) */
    WORD V_REZ_HZ;      /* Bildschirmbreite in Pixel (-$0C) */
    LONG V_OFF_AD;      /* Zeiger auf Zeichensatz-Offset-Tabelle
                           (siehe VDI ESC 102) (-$0A) */
    WORD RESERVED;      /* TOS 1.00: Cursorflag (-$06)
                           Bit 0:  Blinken aus/ein
                           Bit 1:  momentaner Blinkstatus:
                                   normal/invertiert
                           Bit 2:  Cursor unsichtbar/sichtbar
                           Bit 3:  Wrapping am Zeilenende ein/aus
                           Bit 4:  Inverse Darstellung nein/ja
                           Bit 5:  Cursorposition gespeichert
                                   nein/ja
                           Hat sich seither geändert!!! */
    WORD V_REZ_VT;      /*  Bildschirmhöhe in Pixel (-$04) */
    WORD BYTES_LIN;     /*  Bytes pro Pixelzeile (-$02) */
} VDIESC;
```

In A1 erhält man einen Zeiger auf eine Tabelle mit Zeigern auf die Systemzeichensätze. Zusätzlich enthält A2 einen Zeiger auf eine Tabelle mit den Anfangsadressen der Line-A-Routinen.

Put pixel ($A001)

Setzt einen Punkt an den angegebenen Koordinaten. Die Parameter müssen in den entsprechenden Eingabefeldern des VDI gesetzt werden. Man beachte also die Variablen "INTIN" und "PTSIN".

Eingabewerte:

intin[0]: Farbe
ptsin[0]: X-Koordinate
ptsin[1]: Y-Koordinate

Get pixel ($A002)

Fragt die Farbe eines beliebigen Punktes auf dem Bildschirm ab.

Eingabewerte:

ptsin[0]: X-Koordinate
ptsin[1]: Y-Koordinate

Ausgabewerte:

D0: Farbe des gegebenen Punkts

Arbitrary Line ($A003)

Zeichnet eine Linie zwischen zwei Punkten. Die Linie wird dabei immer von links nach rechts gezogen. Für horizontale Linien steht die schnellere Funktion "Horizontal Line" ($A004) zur Verfügung, die auch von dieser Funktion genutzt wird.

Eingabewerte:

X1: X-Koordinate des ersten Punkts
Y1: Y-Koordinate des ersten Punkts
X2: X-Koordinate des zweiten Punkts
Y2: Y-Koordinate des zweiten Punkts
COLBIT0: Bitwert für die erste Farbplane
COLBIT1: Bitwert für die zweite Farbplane (nur bei Low- oder Medium-Res.)
COLBIT2: Bitwert für die dritte Farbplane (nur bei Low-Resolution)
COLBIT3: Bitwert für die vierte Farbplane (nur bei Low-Resolution)
LNMASK: Bitmuster (Punktmuster) für die Linie
WMODE: Schreibmodus:
- 0: Replace
- 1: Transparent (OR)
- 2: Inverse (XOR)
- 3: Inverse Transparent (XOR mit not(LN_MASK))

LSTLIN: Letzen Pixel der Linie zeichnen? (0: nein, 1: ja)

Ausgabewerte:

LNMASK: durch den Algorithmus geshiftet

Horizontal line ($A004)

Zeichnet eine horizontale Linie. Da diese Funktion intern auch für Flächenfüllroutinen benutzt wird, ergeben sich interessante weitere Funktionen.

So kann man statt eines Linienmusters auch mehrere Muster angeben, die dann alternierend in aufeinanderfolgenden Zeilen benutzt werden. So kann man sehr leicht mit "Mustern" zeichnen.

Eingabewerte:

X1:	X-Koordinate des ersten Punkts
Y1:	Y-Koordinate der Linie
X2:	X-Koordinate des zweiten Punkts
COLBIT0:	Bitwert für die erste Farbplane
COLBIT1:	Bitwert für die zweite Farbplane (nur bei Low- oder Medium-Res.)
COLBIT2:	Bitwert für die dritte Farbplane (nur bei Low-Resolution)
COLBIT3:	Bitwert für die vierte Farbplane (nur bei Low-Resolution)
WMODE:	Schreibmodus:
	0: Replace
	1: Transparent (OR)
	2: Invert (XOR)
	3: Inverse Transparent (XOR mit not(Eingabe))
PATPTR:	Zeiger auf ein Feld von Linienmustern (je 16 Bit)
PATMSK:	Anzahl der verschiedenen Linienmuster
MFILL:	0: nur in der ersten Farbebene
	>0: alle Farbebenen

Filled rectangle ($A005)

Zeichnet ein ausgefülltes Rechteck.

Eingabewerte:

X1:	X-Koordinate der oberen linken Ecke
Y1:	Y-Koordinate der oberen linken Ecke
X2:	X-Koordinate der unteren rechten Ecke
Y2:	Y-Koordinate der unteren rechten Ecke

COLBIT0: Bitwert für die erste Farbplane
COLBIT1: Bitwert für die zweite Farbplane (nur bei Low- oder Medium-Res.)
COLBIT2: Bitwert für die dritte Farbplane (nur bei Low-Resolution)
COLBIT3: Bitwert für die vierte Farbplane (nur bei Low-Resolution)
WMODE: Schreibmodus:
0: Replace
1: Transparent (OR)
2: Invert (XOR)
3: Inverse Transparent (XOR mit not(Eingabe))
PATPTR: Zeiger auf ein Feld von Linienmustern (je 16 Bit)
PATMSK: Anzahl der verschiedenen Linienmuster
MFILL: 0: nur in der ersten Farbebene
>0: alle Farbebenen
CLIP: Clipping: 0: aus, 1: an
XMINCL: Oberer Rand des Clip-Rechtecks
XMAXCL: Unterer Rand des Clip-Rechtecks
YMINCL: Linker Rand des Clip-Rechtecks
YMAXCL: Rechter Rand des Clip-Rechtecks

Filled polygon ($A006)

Die Bezeichnung dieser Funktion ist ein wenig irreführend, da lediglich eine Zeile des Polygons gefüllt wird.

Um das gesamte Polygon zu füllen, muß man also die Funktion mehrfach aufrufen.

Im ptsin[]-Feld werden die Koordinaten der Eckpunkte angegeben. Als letzter Punkt muß natürlich noch mal der Anfangspunkt angegeben werden, da man sonst keine geschlossene Figur erhält.

Eingabewerte:

ptsin[]: Enthält die Koordinaten der Eckpunkte des Polygonzugs
contrl[1]: Anzahl der Koordinaten in ptsin[]
Y1: Y-Koordinate der zu zeichnenden Linie
COLBIT0: Bitwert für die erste Farbplane
COLBIT1: Bitwert für die zweite Farbplane (nur bei Low- oder Medium-Res.)
COLBIT2: Bitwert für die dritte Farbplane (nur bei Low-Resolution)
COLBIT3: Bitwert für die vierte Farbplane (nur bei Low-Resolution)

WMODE:	Schreibmodus:
	0: Replace
	1: Transparent (OR)
	2: Invert (XOR)
	3: Inverse Transparent (XOR mit not(Eingabe))
PATPTR:	Zeiger auf ein Feld von Linienmustern (je 16 Bit)
PATMSK:	Anzahl der verschiedenen Linienmuster
MFILL:	0: nur in der ersten Farbebene
	>0: alle Farbebenen
CLIP:	Clipping (0: aus, 1: an)
XMINCL:	Oberer Rand des Clip-Rechtecks
XMAXCL:	Unterer Rand des Clip-Rechtecks
YMINCL:	Linker Rand des Clip-Rechtecks
YMAXCL:	Rechter Rand des Clip-Rechtecks

Ausgabewerte:

X1 und X2 werden verändert; A0 wird zerstört

Bitblt (Bit Block Transfer) ($A007)

Dies ist die Grundfunktion für "Text Block Transfer" ($A008) und "Copy raster form" ($A00E). Einziger Eingabewert für diese Funktion ist das Adreßregister A6, das einen Zeiger auf eine BITBLT-Struktur enthalten muß:

```
typedef struct {
     WORD B_WD;      /* Breite des Blocks in Pixeln */
     WORD B_HT;      /* Höhe des Blocks in Pixeln */
     WORD PLANE_CT; /* Anzahl der Farbplanes; wird durch "Bitblt"
                       zerstört */
     WORD FG_COL;    /* Vordergrundfarbe; wird durch "Bitblt"
                       zerstört */
     WORD BG_COL;    /* Hintergrundfarbe; wird durch "Bitblt"
                       zerstört */
     LONG OP_TAB;    /* Logische Verknüpfungen für alle vier
                       Kombinationen von Vordergrund- und
                       Hintergrundbits (siehe die 16 möglichen
                       Verknüpfungen für Blitter-Operationen) */
     WORD S_XMIN;    /* X-Koordinate des Quellrasters */
```

```
    WORD S_YMIN;    /* Y-Koordinate des Quellrasters */
    LONG S_FORM;    /* Anfangsadresse des Quellrasters */
    WORD S_NXWD;    /* Offset zum nächsten Wort in der gleichen
                       Plane (HighRez 2, MedRez 4, LowRez 8) */
    WORD S_NXLN;    /* Breite des Quellrasters in Bytes */
    WORD S_NXPL; /* Offset zur nächsten Plane (beim ST immer 2) */
    WORD D_XMIN;    /* X-Koordinate des Zielrasters */
    WORD D_YMIN;    /* Y-Koordinate des Zielrasters */
    LONG D_FORM;    /* Anfangsadresse des Zielrasters */
    WORD D_NXWD;    /* Offset zum nächsten Wort in der gleichen
                       Plane (Zielraster) */
    WORD D_NXLN;    /* Breite des Zielrasters in Bytes */
    WORD D_NXPL; /* Offset zur nächsten Plane (beim ST immer 2) */
    LONG P_ADDR;    /* Zeiger auf 16-Bit-Masken, mit denen jeweils
                       undiert wird. (0 = keine Maske) */
    WORD P_NXLN; /* Höhe der Maske in Bytes (eine Potenz von 2) */
    WORD P_NXPL;    /* Zeiger auf nächste Plane in der Maske */
    WORD P_MASK;    /* Höhe der Maske in Zeilen */
    CHAR SPACE[24];/* interner Arbeitsbereich für "Bitblt" */
} BITBLT;
```

Bemerkungen
Clipping findet nicht statt. Weiterhin wird nicht getestet, ob die angegebenen Ausschnitte tatsächlich innerhalb der benutzten Memory-Form liegen.

TextBlt (Text Block Transfer) ($A008)

Dies ist eine universelle Routine zur Ausgabe von einzelnen Zeichen auf dem Bildschirm. Zu ihren vielfältigen Funktionen zählen die Drehung von Zeichen (in 90-Grad-Schritten), verschiedene Textattribute und diverse Schriftgrößen. Bei "TextBlt" ergeht es einem nicht anders als bei anderen "Monster-Routinen" des Betriebssystems – je universeller, desto mehr Register muß man setzen und desto umständlicher ist die Handhabung.

Bevor wir uns aber auf die einzelnen Eingabeparameter stürzen, seien noch ein paar Worte zum Aufbau eines Zeichensatzes gesagt.

Von "konventionellen" Rechnern war man an einen sehr einfachen Aufbau von Zeichensätzen gewöhnt. Da hatten alle Zeichen die gleiche Breite und Höhe, besondere Schriftattribute wurden – wenn überhaupt verfügbar– von der Hardware erzeugt, und verschiedene, gleich-

zeitig verwendbare Zeichensätze waren überhaupt kein Thema. Ganz anders ist es unter TOS. Dadurch, daß alle Bildschirmausgaben im Grafikmodus erfolgen, kann das Betriebssystem verschiedene Textattribute (wie Fettschrift oder kursiv) direkt unterstützen. Zusätzlich wollen auch noch Dinge wie Proportionalschrift verwaltet sein. Womit wir schon beim Problem angelangt sind – wie legt man möglichst speicherplatzsparend einen Zeichensatz im Speicher ab? Atari hat folgende Lösung gewählt: der Zeichensatz wird einfach als großes "Bild" betrachtet, dessen Höhe die Zeichenhöhe und dessen Breite die Sum-me aller Zeichenbreiten ist.

Alle Zeichen stehen in diesem "Fontimage" direkt nebeneinander. Auf diese Weise geht zwar kein einziges Bit verloren, dafür muß aber die Position eines jeden Zeichens erst mittels einer Offset-Tabelle berechnet werden. Für GEM-Zeichensätze gibt es ein Standard-Dateiformat, wie es zum Beispiel auch von GDOS gelesen wird (siehe dazu Einführung ins VDI). Damit ist die Zeichenausgabe auf das Kopieren eines Ausschnitts aus einem imaginären Bildschirm (dem Zeichensatz) auf den tatsächlichen Bildschirm reduziert – und dafür gibt es ja andere Routinen.

Eingabewerte:

WMODE: Schreibmodus:
- 0: Replace
- 1: Transparent (OR)
- 2: Invert (XOR)
- 3: Inverse Transparent (XOR mit not(Eingabe))
- 4–19: Bitblt-Modi 0–15

CLIP: Clipping aus (0) oder an (1)

XMINCL: Oberer Rand des Clip-Rechtecks

XMAXCL: Unterer Rand des Clip-Rechtecks

YMINCL: Linker Rand des Clip-Rechtecks

YMAXCL: Rechter Rand des Clip-Rechtecks

XDDA: Vor jedem Aufruf auf $8000 setzen

DDAINC: Vergrößerungsfaktor

SCALDIR: Vergrößerungsrichtung

MONO:
- 0: Zeichen können verschiedene Breiten haben (wg. Texteffekten oder Proportionalschrift)
- 1: Alle Zeichen haben die gleiche Breite

SOURCEX: X-Position des Zeichens im Fontimage (Berechnung: zunächst vom ASCII-Wert "first_ade" abziehen und diesen Wert dann als Index in die Offset-Tabelle "off_table" verwenden)

SOURCEY: Y-Position des Zeichens im Fontimage. Gewöhnlich 0.

DESTX: X-Zielkoordinate

DESTY:	Y-Zielkoordinate
DELX:	Breite des Zeichens (dazu subtrahiert man die X-Position von der X-Position des folgenden Zeichens)
DELY:	Höhe des Zeichens (üblicherweise "form_height")
FBASE:	Zeiger auf Zeichensatzdaten ("dat_table")
FWIDTH:	Breite des Zeichensatzimage ("form_width")
STYLE:	Texteffekte: Bit 0: Fettschrift Bit 1: Helle Schrift Bit 2: Kursivschrift Bit 3: Unterstrichen (wird ignoriert) Bit 4: Umrissene Schrift ("outline")
LITEMASK:	Maske für helle Schrift ("lighten")
SKEWMASK:	Maske für Kursivschrift ("skew")
WEIGHT:	Zusätzliche Breite für Fettschrift ("thicken")
ROFF:	Rechter Offset des Zeichens bei Kursivschrift ("right_offset")
LOFF:	Linker Offset des Zeichens bei Kursivschrift ("left_offset")
SCALE:	Scaling aus (0) oder ein (1)
CHUP:	Schriftrichtung in 1/10 Grad (also 0, 900, 1800 oder 2700)
TEXTFG:	Textfarbe
SCRTCHP:	Zeiger auf einen Buffer, der zur Erzeugung der Texteffekte benötigt wird (2 * Maximalgröße eines Zeichens)
SCRPT2:	Offset zur Mitte des Buffers (=1/2 der Länge des Buffers)
TEXTBG:	Texthintergrundfarbe

Show mouse ($A009)

Diese Routine ist die direkte Entsprechung der "Show Cursor"-Funktion des VDI ("v_show_c()") und dient zum Einschalten des Mauscursors. Intern wird mitgezählt, wie oft die Maus vorher abgeschaltet worden ist, so daß man gegebenenfalls die Aufrufe verschachteln kann. Die Maus wird nur dann angezeigt, wenn dieser interne Zähler wieder auf 0 steht. Diesen Mechanismus kann man allerdings mittels des Parameters in intin[0] ausschalten.

Eingabeparameter:

intin[0]:	0:	Maus auf jeden Fall wieder einschalten
	>0:	normale Arbeitsweise

Bemerkungen

Die Line-A-Zeiger auf die VDI-Eingabefelder müssen natürlich auch korrekt gesetzt sein!

Hide mouse ($A00A)

Pendant zur vorhergehenden Funktion (schaltet den Mauscursor aus).

Transform mouse ($A00B)

Mit dieser Funktion kann man das Erscheinungsbild der Maus ändern. Im intin[]-Feld wird eine MFORM-Struktur erwartet.

Die einfachste Art der Parameterübergabe ist, sich zunächst den alten Zeiger auf das intin[]-Feld zu merken (INTIN) und mit einem Zeiger auf die MFORM-Struktur zu überschreiben. Nach Ausführung der Funktion stellt man dann den Vektor wieder her.

Die Mausform-Struktur sieht folgendermaßen aus:

```
typedef struct
{
     WORD mf_xhot;       /* Aktionspunkt X-Koordinate */
     WORD mf_yhot;       /* Aktionspunkt Y-Koordinate */
     WORD mf_nplanes;    /* auf 1 setzen */
     WORD mf_fg;         /* Maskenfarbe */
     WORD mf_bg;         /* Cursorfarbe */
     WORD mf_mask[16];   /* Maskendaten (16*16 Punkte) */
     WORD mf_data[16];   /* Cursordaten (16*16 Punkte) */
} MFORM;
```

Der Aktionspunkt ist dabei die Stelle innerhalb des Mauscursors, auf die sich alle Maus-Koordinatenangaben beziehen. Beim einfachen Pfeil liegt der Aktionspunkt beispielsweise in der linken oberen Ecke (0,0).

Undraw sprite ($A00C)

Löscht ein mit "DRAW SPRITE" gezeichnetes Sprite vom Bildschirm und stellt den Hintergrund wieder her. Die Funktion erwartet in A2 einen Zeiger auf den Sprite-Save-Block (siehe "DRAW SPRITE"). Register A6 wird zerstört.

Draw sprite ($A00D)

Zeichnet ein bis zu 16*16 Punkte großes Sprite auf dem Bildschirm. Register A6 wird zerstört.

Eingabewerte:

D0: X-Koordinate des Aktionspunktes des Sprites auf dem Bildschirm
D1: Y-Koordinate
A0: Zeiger auf SDB-Struktur
A2: Zeiger auf Sprite-Save-Buffer (Größe in Bytes: 10+64 * Anzahl der Farbebenen). Die Adresse dieses Buffers muß auch an "UNDRAW SPRITE" übergeben werden.

```
typedef struct
{
     WORD xhot;          /* X-Koordinate des Aktionspunkts */
     WORD yhot;          /* Y-Koordinate des Aktionspunkts */
     WORD form;          /* 1: VDI-Format, -1: XOR-Format */
     WORD bgcol;         /* Hintergrundfarbe */
     WORD fgcol;         /* Vordergrundfarbe */
     WORD image[32];     /* Sprite-Image */
} SDB;
```

In den Sprite-Daten wechseln sich jeweils ein Wort für den Vordergrund und ein Wort für den Hintergrund ab. Es ergeben sich folgende Verknüpfungen (*old* sei der bisherige Pixelwert des Bildpunkts):

Vordergr.-Bit	Hintergr.-Bit	VDI-Format	XOR-Format
0	0	new=old	new=old
0	1	new=bgcol	new=bgcol
1	0	new=fgcol	new=fgcol xor old
1	1	new=fgcol	new=fgcol

Copy raster form ($A00E)

Diese Funktion entspricht exakt der VDI-Funktion "COPY RASTER, OPAQUE", mit der Ausnahme, daß das contrl[]-Feld nicht vollständig gesetzt und keine Workstation geöffnet zu werden braucht. Man muß lediglich die Elemente im ptsin[]-Feld und contrl[7-10] ausfüllen.

Seedfill ($A00F)

Entspricht exakt der VDI-Funktion, mit folgenden Ausnahmen:

- Man braucht keine Workstation zu öffnen.
- Die Clipping-Variablen müssen gesetzt werden.
- SEEDABORT ist ein Zeiger auf eine Routine, die am Ende jeder Bildschirmzeile einmal aufgerufen wird. Liefert sie in D0 einen Wert ungleich 0 zurück, wird der Füllvorgang abgebrochen.

Plottertreiber

Einleitung

GDOS-Plottertreiber für den Atari gibt es meines Wissens bislang nicht. Ich würde mich allerdings gerne eines Besseren belehren lassen!

Mindestfunktionsumfang

(Spezifikation nach Atari, "GEM Programmer's Guide")

Opcode		Funktion
1		Open workstation
2		Close workstation
3		Clear workstation
4		Update workstation
5		Escape-functions
	1	Inquire addressable character cells
6		Polyline
7		Polymarker
8		Text
9		Filled area
11		Generalized Drawing Primitive (GDP)
	1	Bar
	2	Arc
	3	Pie

Opcode	Funktion
4	Circle
5	Ellipse
6	Elliptical arc
7	Elliptical pie
8	Rounded rectangle
9	Filled rounded rectangle
10	Justified graphics text
12	Set character height, absolute mode
15	Set polyline linetype
17	Set polyline color index
18	Set polymarker type
20	Set polymarker color index
21	Set text face
22	Set graphic text color index
23	Set fill interior style
24	Set fill style index
25	Set fill color index
35	Inquire current polyline attributes
36	Inquire current polymarker attributes
37	Inquire current fill area attributes
38	Inquire current graphic text attributes
39	Set graphic text alignment
102	Extended inquire function
104	Set fill perimeter visibility
107	Set character cell height, points mode
108	Set polyline end styles
116	Inquire text extent
117	Inquire character cell width
130	Inquire face name and index
131	Inquire current face information

Druckertreiber

Besonderheiten

GDOS-Druckertreiber gibt es mittlerweise in allen Formen und Farben – angefangen beim Epson-kompatiblen bis zum Atari-Laserdrucker. Selbst Treiber für HP-Laserdrucker und Postscript-Ausgabegeräte wurden schon gesichtet.

Eingetragene Entwickler können bei Atari das sogenannte "Treiber-Entwicklungs-Paket" bekommen. Es besteht aus einer kompletten Bibliothek, die nur noch um einige wenige druckerspezifische Funktionen erweitert werden muß, um einen Treiber für einen monochromen, grafikfähigen Drucker zu erzeugen. Bei der Arbeit mit VDI-Druckertreibern beachte man, daß nicht alle Treiber funktional völlig identisch sind. Bei Verwendung des Lasertreibers gibt es nicht nur zusätzliche Funktionen, sondern einige bestehende wurden auch teilweise erweitert. Auf Unterschiede wird bei den einzelnen Funktionen jeweils getrennt hingewiesen.

Schließlich sei noch einmal auf die erst ab GEM 2.0 dokumentierten Escape-Funktionen "v_q_scan()" und "v_alpha_text()" hingewiesen, die eine sehr einfache Möglichkeit zur standardisierten Textausgabe liefern.

Mindestfunktionsumfang

(Spezifikation nach Atari, "GEM Programmer's Guide")

Opcode		Funktion
1		Open workstation
2		Close workstation
3		Clear workstation
4		Update workstation
5		Escape-functions
	1	Inquire addressable character cells
	20	Form advance
	21	Output window
	22	Clear display list
	23	Output bit image file
6		Polyline
7		Polymarker
8		Text
9		Filled area
11		Generalized Drawing Primitive (GDP)
	1	Bar
	2	Arc
	3	Pie
	4	Circle
	5	Ellipse
	6	Elliptical arc
	7	Elliptical pie
	8	Rounded rectangle

Opcode	Funktion
9	Filled rounded rectangle
10	Justified graphics text
12	Set character height, absolute mode
15	Set polyline linetype
17	Set polyline color index
18	Set polymarker type
20	Set polymarker color index
21	Set text face
22	Set graphic text color index
23	Set fill interior style
24	Set fill style index
25	Set fill color index
26	Inquire color representation
32	Set writing mode
35	Inquire current polyline attributes
36	Inquire current polymarker attributes
37	Inquire current fill area attributes
38	Inquire current graphic text attributes
39	Set graphic text alignment
102	Extended inquire function
104	Set fill perimeter visibility
106	Set graphic text special effects
107	Set character cell height, points mode
108	Set polyline end styles
112	Set user-defined fill pattern
116	Inquire text extent
117	Inquire character cell width
129	Set clipping rectangle
130	Inquire face name and index
131	Inquire current face information

Metafile-Treiber

Einleitung

Der Metafile-Treiber ermöglicht es, VDI-Befehle in geräteunabhängiger Form in eine Datei zu schreiben. Der so erzeugte Metafile enthält eine "Aufzeichnung" aller getätigten VDI-Aufrufe inklusive aller Parameter.

Format von Metafiles

Wer Metafiles lesen können will, muß das Format des Metafile-Headers kennen:

```
typedef struct
{
     WORD   mf_header;      /* -1 (Metafile-Kennung) */
     WORD   mf_hlength;     /* Länge des Headers in Integers (24) */
     WORD   mf_version;     /* bei der aktuellen Version 101
                               (Version 1.01), Formel:
                               100*Hauptnummer+Unternummer */
     WORD   mf_ndcrcfl;     /* NDC/RC-Flag (0 oder 2) */
     WORD   mf_extents[4]; /* optional - maximale Ausmaße der
                               Grafik - können mit
                               "v_meta_extents()" gesetzt werden
                               (sonst mit 0 gefüllt) */
     WORD   mf_pagesz[2];   /* optional - Seitengröße in 1/10 mm -
                               kann mit "vm_pagesize()" gesetzt
                               werden (sonst mit Nullen gefüllt) */
     WORD   mf_coords[4];   /* optional - Koordinatensystem - kann
                               mit "vm_coords()" gesetzt werden
                               (sonst mit Nullen gefüllt) */
     WORD   mf_imgflag;     /* falls Bit 0 = 1, so enthält die Datei
                               Bitimages, sonst nicht. Die anderen
                               Bits sind immer 0. */
     WORD   mf_resvd[9];    /* zur Zeit unbenutzt */
} METAHDR;
```

Beim Lesen der Metafiles wird keine feste Länge benutzt. Vielmehr liest eine Applikation erst die ersten beiden Wörter und damit die Länge des Headers. Anschließend wird der Rest des Headers gelesen. Es folgen dann "beliebig" viele Einträge folgender Form :

Wort	Inhalt	Belegung
0	contrl[0]	VDI-Befehlsnummer
1	contrl[1]	Anzahl der Werte im ptsin[]-Feld
2	contrl[3]	Anzahl der Werte im intin[]-Feld
3	contrl[5]	Unterfunktionsnummer
ab 4	ptsin[]	Alle belegten Felder des ptsin[]-Feldes
es folgt	intin[]	Alle belegten Felder des intin[]-Feldes

Speziell behandelt werden die Funktionen "v_opnwk()", die die Datei öffnet und alle Standard-Attribute als Aufrufe von Attributfunktionen in die Datei schreibt, "v_clswk()", die als Endekennung eine "–1" in die Datei schreibt, und einige Escapefunktionen, die Werte in den Header der Metadatei eintragen (oder sie umbenennen).

Fehler bei der Ausgabe in Dateien werden auf ausgesprochen humorvolle Weise beanstandet – in "ptsout[0]" und "ptsout[1]" werden die Ziffern "987" bzw. "654" geschrieben (*nicht* offiziell dokumentiert!). Was Digital Research sich dabei gedacht hat, wird wohl für immer ein Rätsel bleiben. Unbedingt zu beachten ist ferner, daß alle Wörter im Intel-Format (also High- und Low-Byte vertauscht) abgelegt werden!

Erweiterte Opcodes

Der Hauptzweck von Metafiles ist der standardisierte Datenaustausch zwischen GEM-Applikationen. Die PC-Version von GEM wird von deutschen Softwarehaus CCP gepflegt, das auch das bekannte Anwendungsprogramm "GEM Artline" entwickelt hat.

Viele Atari-Programme bemühen sich, GEM/3-kompatible Metafiles lesen und schreiben zu können. Die Unterschiede zum oben beschriebenen Format ergeben sich ausschließlich durch das mögliche Auftreten von Opcodes neuer VDI-Funktionen (wie etwa Bézierkurven oder Grauraster). GEM/4 (in Artline 2) kennt weitere Funktionen wie etwa zum Setzen von Farbverläufen.

Daher ist es leider *nicht* möglich, an dieser Stelle exakt und vollständig zu beschreiben, was alles in einem Metafile stehen kann – das hängt ausschließlich davon ab, welches Programm die Datei erzeugt und welches sie einlesen können soll. Im Zweifelsfall muß man bei Digital Research die aktuellen Entwicklerunterlagen zu PC-GEM erwerben.

Vielleicht ringt sich Atari eines Tages selbst zu einer überarbeiteten Spezifikation durch.

Mindestfunktionsumfang

(Spezifikation nach Atari, "GEM Programmer's Guide")

Opcode	Funktion
1	Open workstation
2	Close workstation
3	Clear workstation
4	Update workstation
5	Escape-functions

Opcode		Funktion
	1	Inquire addressable character cells
	2	Exit alpha mode
	3	Enter alpha mode
	20	Form advance
	21	Output window
	22	Clear display list
	23	Output bit image file
	98	Update metafile extents
	99	Write metafile item
	100	Change GEM VDI filename
6		Polyline
7		Polymarker
8		Text
9		Filled area
11		Generalized Drawing Primitive (GDP)
	1	Bar
	2	Arc
	3	Pie
	4	Circle
	5	Ellipse
	6	Elliptical arc
	7	Elliptical pie
	8	Rounded rectangle
	9	Filled rounded rectangle
	10	Justified graphics text
12		Set character height, absolute mode
13		Set character baseline vector
14		Set color representation
15		Set polyline linetype
16		Set polyline line width
17		Set polyline color index
18		Set polymarker type
19		Set polymarker height
20		Set polymarker color index
21		Set text face
22		Set graphic text color index
23		Set fill interior style
24		Set fill style index
25		Set fill color index

Opcode	Funktion
26	Inquire color representation
32	Set writing mode
35	Inquire current polyline attributes
36	Inquire current polymarker attributes
37	Inquire current fill area attributes
38	Inquire current graphic text attributes
39	Set graphic text alignment
102	Extended inquire function
103	Contour fill
104	Set fill perimeter visibility
106	Set graphic text special effects
107	Set character cell height, points mode
108	Set polyline end styles
112	Set user-defined fill pattern
113	Set user-defined line style pattern
114	Fill rectangle
117	Inquire character cell width
129	Set clipping rectangle
131	Inquire current face information

Kameratreiber

Einleitung

Treiber für die "Polaroid Palette" sind nicht erhältlich und werden wohl auch nicht mehr implementiert werden (aber wer weiß...). In der Dokumentation zu GEM 2.0 hat Digital Research sämtliche Funktionen geändert – da es sowieso bislang keine Treiber gibt, haben wir hier alle Änderungen vollständig übernommen!

Mindestfunktionsumfang

(Spezifikation nach Digital Research zu GEM 2.0)

Opcode	Funktion
1	Open workstation
2	Close workstation
3	Clear workstation

Opcode		Funktion
4		Update workstation
5		Escape-functions
	1	Inquire addressable character cells
	23	Output bit image file
	91	Select camera film type and exposure time
	92	Inquire camera film name
6		Polyline
11		Generalized Drawing Primitive (GDP)
	1	Bar
	2	Arc
	3	Pie
	4	Circle
	5	Ellipse
	6	Elliptical arc
	7	Elliptical pie
	8	Rounded rectangle
	9	Filled rounded rectangle
	10	Justified graphics text
12		Set character height, absolute mode
13		Set character baseline vector
14		Set color representation
15		Set polyline linetype
16		Set polyline line width
17		Set polyline color index
18		Set polymarker type
19		Set polymarker height
20		Set polymarker color index
21		Set text face
22		Set graphic text color index
23		Set fill interior style
24		Set fill style index
25		Set fill color index
26		Inquire color representation
32		Set writing mode
35		Inquire current polyline attributes
36		Inquire current polymarker attributes
37		Inquire current fill area attributes
38		Inquire current graphic text attributes
39		Set graphic text alignment

Opcode	Funktion
102	Extended inquire function
104	Set fill perimeter visibility
106	Set graphic text special effects
107	Set character cell height, points mode
108	Set polyline end styles
112	Set user-defined fill pattern
113	Set user-defined line style pattern
116	Inquire text extent
117	Inquire character cell width
119	Load fonts
120	Unload fonts
129	Set clipping rectangle
130	Inquire font name and index
131	Inquire current face information

VDI-Referenz

Kontrollfunktionen

OPEN WORKSTATION (VDI 1)

"OPEN WORKSTATION" initialisiert einen Grafik-Gerätetreiber für ein bestimmtes Ein-/Ausgabegerät. Ein Eingabefeld teilt die dazu notwendigen Parameter mit, Informationen über das Gerät erhält man in einem speziellen Ausgabefeld. Gerätetreiber müssen in der ASSIGN.SYS-Datei, die beim Booten von GDOS gelesen wird, angemeldet sein (siehe Erläuterungen zum Aufbau der ASSIGN.SYS-Datei).

Sollte es sich um einen Bildschirmtreiber handeln, so wird der Grafik-Modus gestartet. Im Normalfall sind es die AES, die die Bildschirm-Workstation für sich öffnen. Anwendungsprogramme müssen daher *virtuelle* Workstations öffnen. Nur wenn die AES noch nicht aktiv sind (während der Abarbeitung des AUTO-Ordners), ist es möglich, Bildschirm-Workstations zu öffnen. Wird OPEN WORKSTATION dazu benutzt, einen Metafile zu öffnen, so schreibt diese Funktion den Metafileheader und initialisiert den Dateipuffer. Wenn die Funktion erfolgreich war, wird in contrl[6] eine Gerätekennung übergeben, sonst ist contrl[6] gleich 0 (*Vorsicht*: Die allermeisten GDOS-Versionen können *nicht* beliebig viele Workstations öffnen!).

Deklaration in C:

```
void v_opnwk (WORD *work_in, WORD *handle, WORD *work_out)
{
     iioff = work_in;
     iooff = work_out;
     pooff = work_out+45;
     contrl[0] = 1;
     contrl[1] = 0;
     contrl[3] = 11;
     vdi ();
     *handle = contrl[6];
     iioff = intin;
     iooff = intout;
     pooff = ptsout;
     pioff = ptsin;
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung
contrl	contrl[0]	1 Opcode für V_OPNWK
contrl+2	contrl[1]	0 # Einträge in ptsin
contrl+4	contrl[2]	6 # Einträge in ptsout
contrl+6	contrl[3]	11 # Einträge in intin
contrl+8	contrl[4]	45 # Einträge in intout
contrl+12	contrl[6]	handle Gerätekennung
intin	intin[0..10]	work_in[0..10]
intout	intout[0..44]	work_out[0..44]
ptsout	ptsout[0..11]	work_out[45..56]

Parameter:

work_in[0]: Gerätidentifikationsnummer, bestimmt den zu ladenden Gerätetreiber (siehe auch ASSIGN.SYS-Datei):
- 1: Bildschirmtreiber (aktuelle Auflösung)
- 2+Getrez(): Bildschirmtreiber (das VDI schaltet selbsttätig in die entsprechende Auflösung; nur für die Standard-Auflösungen von ST und TT definiert).
- ab 11: Plottertreiber
- ab 21: Druckertreiber
- ab 31: Metafile-Treiber
- ab 41: Kamera
- ab 51: Grafiktablett

work_in[1]: Linientyp

work_in[2]: Linienfarbe (wird auf 1 gesetzt, falls die Werte bei work_in[2], work_in[4], work_in[6] bzw. work_in[9] falsch gesetzt werden)

work_in[3]: Markertyp

work_in[4]: Markerfarbe

work_in[5]: Zeichensatznummer

work_in[6]: Textfarbe

work_in[7]: Fülltyp

work_in[8]: Füllmuster-Index (Bemerkung: bei diesem Parameter existiert ein Fehler im VDI. Er wird um den Wert 1 zu hoch übernommen, d. h. bei Angabe von n wird Füllmuster n + 1 gewählt.)

work_in[9]: Füllmuster-Farbe

work_in[10]: Koordinaten-Flag
0: NDC-Koordinaten (nur mit GDOS)
1: reserviert
2: RC-Koordinaten
Vorsicht: Je nach verwendetem GDOS wird bei einem ungültigen Wert entweder NDC oder RC gewählt!

work_out[0]: Maximale horizontale Pixelposition
(beispielsweise im Grafikmodus "ST-Hoch": 639)

work_out[1]: Maximale vertikale Pixelposition
(beispielsweise im Grafikmodus "ST-Hoch": 399)

work_out[2]: Gerätekoordinaten-Flag
0: Bild kann genau skaliert werden (wie auf dem Bildschirm)
1: Bild kann nicht genau skaliert werden (wie auf Film-Recordern)

work_out[3]: Breite eines Pixels (oder Plotterschrittweite) in Mikrometern (=mm/1000). Für Bildschirme sind work_out[3] und work_out[4] prinzipbedingt nicht exakt (das Betriebssystem kennt schließlich nicht die benutzte Bilddiagonale). Immerhin kann man den Quotienten aus beiden Werten sinnvoll auswerten!

work_out[4]: Höhe eines Pixels (oder Plotterschrittweite) in Mikrometern (=mm/1000)

work_out[5]: Anzahl der Schriftzeichenhöhen (0: beliebig veränderbar)

work_out[6]: Anzahl der Linientypen

work_out[7]: Anzahl der Linienbreiten (0: beliebig veränderbar)

work_out[8]: Anzahl der Markertypen

work_out[9]: Anzahl der Markergrößen (0: beliebig veränderbar)

work_out[10]: Anzahl der verfügbaren Zeichensätze (kann mehr als einer sein!)

work_out[11]: Anzahl der Muster

work_out[12]: Anzahl der Schraffuren

work_out[13]: Anzahl der vordefinierten Farben

work_out[14]: Anzahl der Grafikgrundfunktionen (GDP)

work_out[15] bis
work_out[24]: Liste der unterstützten Grafikgrundfunktionen (GDP).
Das Ende ist durch –1 gekennzeichnet.
GEM VDI unterstützt die folgenden 10 Grundfunktionen:
1: Balken
2: Bogen
3: Kreissegment (-ausschnitt)
4: Kreis
5: Ellipse
6: elliptische Bogen
7: Ellipsensegment (-ausschnitt)

8: Rechteck mit abgerundeten Ecken
9: ausgefülltes, abgerundetes Rechteck
10: justierter Grafiktext

work_out[25] bis
work_out[34]: Liste möglicher Attribute der Grafikgrundfunktionen:
0: Linie
1: Marker
2: Text
3: ausgefüllter Bereich
4: kein Attribut

work_out[35]: Farbdarstellungs-Flag (0 nicht verfügbar, 1 verfügbar)
work_out[36]: Textrotations-Flag (0 nicht verfügbar, 1 verfügbar)
work_out[37]: Flächenfüllung (0 nicht verfügbar, 1 verfügbar)
work_out[38]: Flag für die Funktion CELLARRAY (0 nicht verfügbar, 1 verfügbar)
work_out[39]: Anzahl aller verfügbaren Farben oder
0: mehr als 32767 Farben (kontinuierliche Verteilung)
2: monochrom

work_out[40]: Kennzeichnung für Grafik-Cursor-Kontrolle
0: keine
1: Tastatur (allein)
2: Tastatur und anderes Gerät (Maus)

work_out[41]: Eingabegerät für variierende Eingaben
0: keine
1: Tastatur
2: anderes Gerät

work_out[42]: Auswahltasten
0: keine
1: Funktionstasten auf der Tastatur
2: anderes Tastenfeld

work_out[43]: alphanumerische Eingabe (String)
0: keine
1: Tastatur

work_out[44]: Ein-/Ausgabegerät-Typ
0: nur Ausgabe
1: nur Eingabe
2: Ein-/Ausgabe
3: reserviert
4: Metafile-Ausgabe

work_out[45]: geringste Zeichenbreite
work_out[46]: geringste Zeichenhöhe bzgl. der Y-Achse des verwendeten Koordinatensystems (Abstand zwischen Baseline und Topline)

work_out[47]: größte Zeichenbreite
work_out[48]: größte Zeichenhöhe bzgl. der Y-Achse des verwendeten Koordinatensystems (Abstand zwischen Baseline und Topline)
work_out[49]: geringste Linienbreite bzgl. der X-Achse des verwendeten Koordinatensystems
work_out[50]: immer 0
work_out[51]: größte Linienbreite bzgl. der X-Achse des verwendeten Koordinatensystems
work_out[52]: immer 0
work_out[53]: geringste Markerbreite bzgl. der X-Achse
work_out[54]: geringste Markerhöhe bzgl. der X-Achse
work_out[55]: größte Markerbreite bzgl. der X-Achse
work_out[56]: größte Markerhöhe bzgl. der X-Achse

Bemerkungen
contrl[6] ist hier ein Ausgabeparameter, der für die anderen Funktionen als Eingabeparameter benötigt wird. Die geringste Zeichenbreite und -höhe bezeichnet die effektive Zeichengröße, nicht die der Box mit Platz etwa für Unterstreichung.

Vergleiche auch Anhang, dort sind die speziellen Werte des ST aufgeführt sowie "EXTENDED INQUIRE FUNCTION" (VDI 102).

Zusätzliche Funktion für Matrixdrucker – es kann die maximale Auflösung angegeben werden:

ptsin[0]: maximale X-Auflösung
ptsin[1]: maximale Y-Auflösung
contrl[1]: auf 1 setzen

Zusätzliche Funktion für den Atari-Page-Printer – die Adresse des internen Buffers wird zurückgeliefert:

contrl[0]: High-Word der Adresse
contrl[1]: Low-Word der Adresse

Standardattribute:

Attribut	Standardwert	VDI-Funktion
Zeichenhöhe	nominelle Zeichenhöhe	vst_height
Basislinienwinkel	0 Grad	vst_rotation
Textausrichtung	links, Basislinie	vst_alignment

Attribut	Standardwert	VDI-Funktion
Texteffekt	normal	vst_effects
Linienstärke	nominelle Linienstärke	vst_width
Markergröße	nominelle Markergröße	vsm_height
Aussehen der Linienenden	rechteckig	vsl_ends
Schreibmodus	Replace	vswr_mode
Eingabemodus	REQUEST (alle Eingabegeräte)	vsin_mode
Umrahmung	sichtbar	vsf_perimeter
frei definierbares Linienmuster	durchgehend	vsl_udsty
frei definierbares Füllmuster	Logo des Herstellers	vsf_udpat
Cursor	versteckt	v_show_c und v_hide_c
Clipping	abgeschaltet	vs_clip

Standardfarben:

Index	Farbe	Bezeichnung
0	weiß	WHITE
1	schwarz	BLACK
2	rot	RED
3	grün	GREEN
4	blau	BLUE
5	cyan	CYAN
6	gelb	YELLOW
7	magenta	MAGENTA
8	hellgrau	LWHITE
9	dunkelgrau	LBLACK
10	dunkelrot	LRED
11	dunkelgrün	LGREEN
12	dunkelblau	LBLUE
13	dunkelcyan	LCYAN
14	dunkelgelb	LYELLOW
15	dunkelmagenta	LMAGENTA
darüber:	geräteabhängig	

CLOSE WORKSTATION (VDI 2)

Mit "CLOSE WORKSTATION" wird das mit "OPEN WORKSTATION" geöffnete Ein-/Ausgabegerät geschlossen und jede weitere Ausgabe zu diesem Gerät unterbunden. Es muß unbedingt darauf geachtet werden, daß vor Benutzung von "CLOSE WORKSTATION" alle virtuellen Workstations geschlossen werden.

CLOSE WORKSTATION bewirkt auf den genannten Gerätetypen folgende Ereignisse:

Bildschirm	Umschaltung in den Alphamodus
Plotter	Update
Drucker	Update (falls nicht unmittelbar vorher einer stattgefunden hat)
Metafile	Datei wird ordnungsgemäß geschlossen
Kamera	Update

Deklaration in C:

```
void v_clswk (WORD handle)
{
     contrl[0] = 2;
     contrl[1] = contrl[3] = 0;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	2	Opcode für V_CLSWK
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung

OPEN VIRTUAL SCREEN WORKSTATION (VDI 100)

Es wird ein *virtuelles* Ausgabegerät auf einem bereits geöffneten *physikalischen* Ausgabegerät geöffnet. Das entsprechende Gerät muß zuvor mit "OPEN WORKSTATION" geöffnet worden sein. Sinn und Zweck des Ganzen ist, daß mehrere Programme gleichzeitig das Gerät benutzen können, ohne einander in die Quere zu kommen (zum Beispiel das eigene Programm und die AES). Auch innerhalb eines Programmes kann es sinnvoll sein, mit mehreren Workstations gleichzeitig zu arbeiten – beispielsweise wenn man zwischen zwei häufig benutzten Mengen von Attributen hin- und herschalten möchte.

Diese Funktion ist nur für den Bildschirmtreiber definiert. Die Gerätekennung der aktuellen physikalischen Bildschirm-Workstation *muß* man beim AES mit "graf_handle()" erfragen!

Das VDI besitzt keinen Mechanismus, um die Eingabegeräte bei mehreren virtuellen Workstations zu verwalten. Diese Aufgabe muß die Applikation übernehmen, die die physikalische Bildschirm-Workstation öffnet.

In den TOS-Versionen 1.00, 1.04, 1.06, 1.62 und 2.05 gibt es einen Fehler bei der Verwaltung der virtuellen Workstations: Folge kann sein, daß Handles mehrfach vergeben werden, so daß unvermittelt Workstation-Attribute verstellt werden. Dies passiert genau dann, wenn in der Bildschirmtreiber-internen Workstation-Liste "Lücken" entstehen. Einzige Abhilfe: das Programm "VDIFIX" von Karsten Isakovic im AUTO-Ordner installieren (man bekommt es direkt vom Autor, in den meisten Mailboxen und unter Umständen auch beim Fachhändler).

Deklaration in C:

```
void v_opnvwk (WORD *work_in, WORD *handle, WORD *work_out)
{
     iioff = work_in;
     iooff = work_out;
     pooff = work_out+45;
     contrl[0] = 100;
     contrl[1] = 0;
     contrl[3] = 11;       contrl[6] = *handle;
     vdi ();
     *handle = contrl[6];
     iioff = intin;
     iooff = intout;
     pooff = ptsout;
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung
contrl	contrl[0]	100 Opcode für V_OPNVWK
contrl+2	contrl[1]	0 # Einträge in ptsin
contrl+4	contrl[2]	6 # Einträge in ptsout
contrl+6	contrl[3]	11 # Einträge in intin
contrl+8	contrl[4]	45 # Einträge in intout
contrl+12	contrl[6]	handle Gerätekennung
intin	intin[0..10]	work_in[0..10]
intout	intout[0..44]	work_out[0..44]
ptsout	ptsout[0..11]	work_out[45..56]

Bemerkungen
Die Ein- und Ausgabeparameter sind mit denen von OPEN WORKSTATION (VDI 1) identisch.

contrl[6] ist beim Aufruf die Nummer der physikalischen Workstation, auf der die virtuelle Workstation geöffnet werden soll. Bei der Rückkehr enthält contrl[6] die Nummer der geöffneten Workstation oder den Fehlercode 0.

"v_opnvwk()" alloziert Speicher, weshalb bei Accessories Vorsicht geboten ist. Um Problemen vorzubeugen, sollte eine virtuelle Workstation von einem Accessory möglichst sofort geöffnet und nie geschlossen werden.

CLOSE VIRTUAL SCREEN WORKSTATION (VDI 101)

Mit "CLOSE VIRTUAL SCREEN WORKSTATION" wird das mit "OPEN VIRTUAL SCREEN WORKSTATION" geöffnete virtuelle Ein-/Ausgabegerät geschlossen und jede weitere Ausgabe zu diesem Gerät unterbrochen.

Vorsicht! Wenn beim Öffnen der Workstation ein Fehler aufgetreten ist, darf man diese Funktion *nicht* aufrufen!

Deklaration in C:

```
void v_clsvwk (WORD handle)
{
     contrl[0] = 101;
     contrl[1] = contrl[3] = 0;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	101	Opcode für V_CLSVWK
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung

CLEAR WORKSTATION (VDI 3)

Mit "CLEAR WORKSTATION" wird der Bildschirm gelöscht und auf die ausgewählte Hintergrundfarbe (Index 0 in der Farbtabelle) gesetzt. Analog wird bei einem Plotter der Puffer gelöscht und bei einem Drucker zusätzlich ein Seitenvorschub durchgeführt. Bei einem Metafile wird lediglich der Opcode gespeichert, bei einer Kamera wird die Ausgabe gelöscht und die Hintergrundfarbe gesetzt. Ein Aufruf von "OPEN WORKSTATION" löscht das Display (Bildschirm o. ä.) automatisch.

Deklaration in C:

```
void v_clrwk (WORD handle)
{
     contrl[0] = 3;
     contrl[1] = contrl[3] = 0;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	3	Opcode für V_CLRWK
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung

UPDATE WORKSTATION (VDI 4)

"UPDATE WORKSTATION" wird benötigt, um einem Ein-/Ausgabegerät mitzuteilen, alle gepufferten Grafikkommandos auszuführen.

Auf einem Bildschirm wird diese Funktion nicht benötigt, da die Grafikkommandos hier umgehend ausgeführt werden (sie ist im ROM auch nicht implementiert). So speichern beispielsweise Druckertreiber die Kommandos in einem Puffer (wird beim Druckertreiber auch "Display-List" genannt).

Die Ausgabe über diese Geräte erfolgt erst nach Aufruf der "UPDATE WORKSTATION"-Funktion. Ist der Puffer für das Ausgabegerät geleert, kehrt die "UPDATE WORKSTATION"-Funktion zurück. Zu beachten ist hierbei, daß bei einem Drucker kein Seitenvorschub gegeben wird. Bei einem Metafile wird lediglich der Opcode in den Metafile-Puffer geschrieben.

Deklaration in C:

```
void v_updwk (WORD handle)
{
     contrl[0] = 4;
     contrl[1] = contrl[3] = 0;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	4	Opcode für V_UPDWK
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung

Bemerkungen
Zusätzliche Funktion für Druckertreiber: Statt der aktuellen Seite kann ein eigener Buffer auf dem Drucker ausgegeben werden.

```
contrl[3] = 2; /* zwei Werte in intin[] */
intin[0]       /* High-Word der Anfangsadresse */
intin[1]       /* Low-Word der Anfangsadresse */
contrl[1] = 1; /* Buffer nicht löschen */
contrl[0] = 4;
contrl[6] = handle;
vdi (); ...
```

Zusätzliche Funktion für die Atari-SLM-Laserdrucker: In intout[0] wird der Status des Druckers zurückgeliefert.

intout[0]	Fehler
0	NO ERROR
2	PRINTER NOT READY
3	TONER EMPTY
5	PAPER EMPTY

LOAD FONTS (VDI 119)

Mit dieser Funktion kann man zusätzlich zu den fest installierten Systemzeichensätzen weitere Zeichensätze laden. Als Ergebnis erhält man die Zahl der hinzugekommenen Zeichensätze (wenn man also "vst_load_fonts()" für eine Workstation mehrfach aufruft, erhält man keine weiteren Zeichensätze und daher als Ergebnis eine 0).

Für diese Funktion muß GDOS installiert sein ("vq_gdos()" benutzen!), sonst gibt es bedauerlicherweise einen Systemabsturz. Die zusätzlichen Zeichensätze müssen in der ASSIGN.SYS-Datei vermerkt sein. Daß "vst_load_fonts()" ohne GDOS nicht funktioniert, hat einen guten Grund. GDOS selbst benutzt nämlich diesen Funktionsaufruf, um die Zeichensatzheader an den entsprechenden Gerätetreiber weiterzuleiten. *Ohne* GDOS dient diese Funktion also zur Installation bereits geladener GEM-Zeichensätze. Nur *mit* GDOS ist sie für Anwenderprogramme interessant.

Und so lädt man auf korrekte Art und Weise Zeichensätze:

1. Nach "v_opnvwk()" die Anzahl der Systemzeichensätze (aus work_out[10]) merken.
2. Mit "vq_gdos()" sicherstellen, daß "vst_load_fonts()" aufgerufen werden darf.
3. Den Return-Wert zur Zahl der Systemzeichensätze addieren. Damit hat man die Gesamtzahl der Zeichensätze.
4. Für alle Zeichensatznummern von 1 bis zur Gesamtzahl "vqt_name()" aufrufen. Damit erfährt man den Zeichensatznamen und den Zeichensatzindex, unter dem man den Zeichensatz später abrufen kann.
5. Am Ende des Programms "vst_unload_fonts()" nicht vergessen!

Deklaration in C:

```
WORD vst_load_fonts (WORD handle, WORD select)
{
     contrl[0] = 119;
     contrl[1] = 0;
     contrl[3] = 1;
     contrl[6] = handle;
     intin[0] = select;
     vdi ();
     return intout[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	119	Opcode für VST_LOAD_FONTS
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	1	# Einträge in intin
contrl+8	contrl[4]	1	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	select	
intout	intout[0]	Return-Wert	

Parameter:

select:	0 (reserviert für zukünftige Benutzung)
vst_load_fonts():	Anzahl der zusätzlich verfügbaren Zeichensätze

Bemerkungen

"vst_load_fonts()" kann Speicher allozieren, daher ist bei Accessories Vorsicht geboten. Um Problemen vorzubeugen, sollte die Funktion entweder nicht oder sofort nach dem Start aufgerufen werden.

UNLOAD FONTS (VDI 120)

Durch "UNLOAD FONTS" werden die durch "vst_load_fonts()" geladenen Zeichensätze wieder entfernt. Da die Zeichensätze nur einmal pro physikalische Workstation geladen werden, werden sie natürlich erst dann wieder aus dem Speicher entfernt, wenn alle "benutzenden" Workstations (beim Bildschirm also die verschiedenen virtuellen) "vst_unload_fonts()" aufgerufen haben.

Die Systemzeichensätze bleiben von dieser Funktion unberührt.

Deklaration in C:

```
void vst_unload_fonts (WORD handle, WORD select)
{
     intin[0] = select;
     contrl[0] = 120;
     contrl[1] = 0;
     contrl[3] = 1;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	120	Opcode für VST_UNLOAD_FONTS
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	1	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	select	

Parameter:

select: 0 (reserviert für zukünftige Benutzung)

SET CLIPPING RECTANGLE (VDI 129)

Mit dieser Funktion wird ein Arbeitsbereich festgelegt oder freigegeben. Wird ein Arbeitsbereich festgelegt, so werden alle Grafikoperationen auf diesen Bereich beschränkt. Überstehende Bereiche werden abgeschnitten. Wenn man kein Clip-Rechteck setzt, wird auch an den Bildschirmgrenzen *nicht* geclippt!

Deklaration in C:

```
void vs_clip (WORD handle, WORD clip_flag, WORD *pxyarray)
{
     pioff = pxyarray;
     intin[0] = clip_flag;
     contrl[0] = 129;
     contrl[1] = 2;
     contrl[3] = 1;
     contrl[6] = handle;
     vdi ();
     pioff = ptsin;
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	129	Opcode für VS_CLIP
contrl+2	contrl[1]	2	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	1	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	clip_flag	
ptsin	ptsin[0..3]	pxyarray[0..3]	

Parameter:

clip_flag: 0: Clippen des Arbeitsbereichs aus
1: Clippen des Arbeitsbereichs ein
pxyarray[0]: X-Koordinate des Eckpunktes
pxyarray[1]: Y-Koordinate des Eckpunktes
pxyarray[2]: X-Koordinate der diagonal gegenüberliegenden Ecke
pxyarray[3]: Y-Koordinate der diagonal gegenüberliegenden Ecke

Ausgabefunktionen

POLYLINE (VDI 6)

Diese Funktion dient dazu, einen Polygonzug zu zeichnen.

Die Koordinaten des ersten Punktes bestimmen den Startpunkt des Linienzuges. Nacheinander werden dann alle Punkte durch eine Linie verbunden.

Einzelne Punkte können durch ein einzelnes Koordinatenpaar nicht gezeichnet werden, jedoch durch eine Linie der Länge Null, also zwei gleiche Koordinatenpaare.

Die Linienattribute und die Einstellung des Schreibmodus werden berücksichtigt.

Deklaration in C:

```
void v_pline (WORD handle, WORD count, WORD *pxyarray)
{
     pioff = pxyarray;
     contrl[0] = 6;
     contrl[1] = count;
     contrl[3] = 0;
     contrl[6] = handle;
     vdi ();
     pioff = ptsin;
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	6	Opcode für V_PLINE
contrl+2	contrl[1]	count (n)	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
ptsin	ptsin[0..2n-1]	pxyarray[0..2n-1]	

Parameter:

pxyarray[0]:	X-Koordinate des ersten Punktes
pxyarray[1]:	Y-Koordinate des ersten Punktes
pxyarray[2]:	X-Koordinate des zweiten Punktes
pxyarray[3]:	Y-Koordinate des zweiten Punktes
.	.
.	.
.	.
pxyarray[2n-2]:	X-Koordinate des letzten (n-ten) Punktes
pxyarray[2n-1]:	Y-Koordinate des letzten (n-ten) Punktes

POLYMARKER (VDI 7)

Diese Funktion zeichnet Marker an vorgegebene Stellen. Die Markerattribute und der eingestellte Schreibmodus werden benutzt.

Anwendung finden die Marker etwa bei grafischer Darstellung von Statistiken.

Deklaration in C:

```
void v_pmarker (WORD handle, WORD count, WORD *pxyarray)
{
     pioff = pxyarray;
     contrl[0] = 7;
     contrl[1] = count;
     contrl[3] = 0;
     contrl[6] = handle;
     vdi ();
     pioff = ptsin;
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	7	Opcode für V_PMARKER
contrl+2	contrl[1]	count (n)	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
ptsin	ptsin[0..2n-1]	pxyarray[0..2n-1]	

Parameter:

count:	Anzahl der Marker
pxyarray[0]:	X-Koordinate des ersten Markers
pxyarray[1]:	Y-Koordinate des ersten Markers
pxyarray[2]:	X-Koordinate des zweiten Markers

pxyarray[3]:	Y-Koordinate des zweiten Markers
.	.
.	.
.	.
pxyarray[2n-2]:	X-Koordinate des letzten (n-ten) Markers
pxyarray[2n-1]:	Y-Koordinate des letzten (n-ten) Markers

TEXT (VDI 8)

Diese Funktion schreibt einen Text auf den Grafikbildschirm.

Das Koordinatenpaar bestimmt die Position des Textes. Der Text wird linksjustiert ausgegeben. Die Basislinie liegt in Richtung der X-Achse, wird also in der Höhe durch die Y-Koordinate bestimmt. Der Text kann durch diverse Attribute auch anders ausgerichtet werden. Jedes einzelne Zeichen (Character) wird durch die Bits 0...7 bestimmt. Die Textattribute und der Schreibmodus werden berücksichtigt.

Man kann die Ausgabegeschwindigkeit beschleunigen, wenn man folgende Punkte beachtet:

- auf Bytegrenzen justiert ausgeben (das nützt natürlich nur bei acht Pixel breiten, nichtproportionalen Zeichensätzen etwas)
- möglichst ohne Clipping ausgeben
- beim Systemzeichensatz (allgemein: bei nichtproportionalen, acht Pixel breiten Zeichen) im Replace-Modus und ohne Clipping ausgeben, ansonsten den TRANSPARENT-Modus wählen.

Ein C-Binding benötigt einen Byte-orientierten und Null-terminierten, ein Assembler-Binding einen Word-orientierten String. Im Zeichensatz fehlende Zeichen werden durch ein Ersatzzeichen (im allgemeinen das Fragezeichen) ersetzt.

Deklaration in C:

```
void v_gtext (WORD handle, WORD x, WORD y, const char *string)
{
     WORD i;
     ptsin[0] = x;
     ptsin[1] = y;
     i = 0;
     while (intin[i++] = *string++);
     contrl[0] = 8;
     contrl[1] = 1;
     contrl[3] = -i;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	8	Opcode für V_GTEXT
contrl+2	contrl[1]	1	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	n	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0..n-1]	string[0..n-1]	
ptsin	ptsin[0]	x	
ptsin+2	ptsin[1]	y	

Parameter:

x: X-Koordinate des Textes
y: Y-Koordinate des Textes
string: Zeiger auf auszugebende Zeichenkette

FILLED AREA (VDI 9)

Diese Funktion füllt eine von einem Polygonzug umrahmte Fläche aus. Dabei werden die Füllattribute beachtet. Die ausgefüllte Fläche wird von einer Linie in der Füllfarbe umgeben. Änderungen der Attribute bzw. des Schreibmodus sind über die Attributfunktionen möglich.

Sollte das Ausgabegerät keine Ausfüllmöglichkeit haben, so wird die Fläche nur mit der aktuellen Füllfarbe umgeben. Es müssen mindestens drei Koordinatenpaare übergeben werden. Eine Nullfläche (nur ein Koordinatenpaar) wird nur dann als Punkt gezeichnet, wenn die automatische Umrahmung der Fläche (siehe SET FILL PERIMETER VISIBILITY) eingeschaltet ist.

Eine Linie wird nicht ausgegeben. Offene Polygonzüge werden selbsttätig geschlossen.

Deklaration in C:

```
void v_fillarea (WORD handle, WORD count, WORD *pxyarray)
{
     pioff = pxyarray;
     contrl[0] = 9;
     contrl[1] = count;
     contrl[3] = 0;
     contrl[6] = handle;
     vdi ();
     pioff = ptsin;
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	9	Opcode für V_FILLAREA
contrl+2	contrl[1]	count (n)	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
ptsin	ptsin[0..2n-1]	pxyarray[0..2n-1]	

Parameter:

count:	Anzahl der Punkte
pxyarray[0]:	X-Koordinate des ersten Punktes
pxyarray[1]:	Y-Koordinate des ersten Punktes
pxyarray[2]:	X-Koordinate des zweiten Punktes
pxyarray[3]:	Y-Koordinate des zweiten Punktes
.	.
.	.
.	.
pxyarray[2n-2]:	X-Koordinate des letzten (n-ten) Punktes
pxyarray[2n-1]:	Y-Koordinate des letzten (n-ten) Punktes

CELL ARRAY (VDI 10)

Diese Funktion erlaubt einen komplexeren Farbaufbau des Bildschirms. Die Farbe der gesetzten Pixel (nicht ganzer Objekte wie Linien) ist ortsabhängig und nicht, wie sonst üblich, objektabhängig. Ein frei definierbares Rechteck wird in einzelne Teile – Zellen – durch Zeilen und Spalten unterteilt. Jeder dieser Zellen wird eine Farbe zugeordnet, die für die Pixelfarbe zuständig ist. Werden nun die Zellen beschrieben, so erscheinen die gesetzten Pixel in der Farbe, die für die jeweilige Zelle bestimmt wurde. Diese Funktion ist im ROM-Bildschirmtreiber nicht implementiert.

Deklaration in C:

```
void v_cellarray (WORD handle, WORD *pxyarray, WORD row_length,
                  WORD el_used, WORD num_rows, WORD wrt_mode,
                  WORD *colarray)
{
     iioff = colarray;
     pioff = pxyarray;
     contrl[0] = 10;
     contrl[1] = 2;
     contrl[3] = row_lenght*num_rows;
     contrl[6] = handle;
     contrl[7] = row_length;
     contrl[8] = el_used;
     contrl[9] = num_rows;
     contrl[10] = wrt_mode;
     vdi ();
     iioff = intin;
     pioff = ptsin;
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	10	Opcode für V_CELLARRAY
contrl+2	contrl[1]	2	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	n	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout

Adresse	Feldelement	Belegung
contrl+12	contrl[6]	handle Gerätekennung
contrl+14	contrl[7]	row_length
contrl+16	contrl[8]	el_used
contrl+18	contrl[9]	num_rows
contrl+20	contrl[10]	wrt_mode
intin	intin[0..n-1]	colarray[0..n-1]
ptsin	ptsin[0..3]	pxyarray[0..3]

Parameter:

row_length: Zeilenlänge im Farbindexarray
el_used: Zonenanzahl pro Zeile (also Spaltenanzahl)
num_rows: Anzahl der Zeilen
wrt_mode: Schreibmodus
colarray[0] bis
colarray[n-1]: Farbindexarray, enthält die Farbinformation zeilenweise
pxyarray[0]: X-Koordinate der unteren linken Ecke des Begrenzungsrechteckes
pxyarray[1]: Y-Koordinate der unteren linken Ecke des Begrenzungsrechteckes
pxyarray[2]: X-Koordinate der oberen rechten Ecke des Begrenzungsrechteckes
pxyarray[3]: Y-Koordinate der oberen rechten Ecke des Begrenzungsrechteckes

Bemerkungen
n ist hier die # Zeilen * # Spalten der Zellenaufteilung. Hat man etwa ein Feld von horizontal zehn und vertikal vier Zellen, so sind dies zehn Spalten und vier Zeilen.

Somit ist n = 4 * 10 = 40.

CONTOUR FILL (VDI 103)

Diese Funktion füllt eine Fläche abhängig vom Startpunkt aus. Die Füllfläche wird durch den Bildschirmrand oder durch eine definierte Farbe begrenzt. Die aktuellen Füllattribute werden beachtet. Einige Ausgabegeräte unterstützen diese Funktion nicht (kann mit "vq_extnd()" erfragt werden).

Deklaration in C:

```
void v_contourfill (WORD handle, WORD x, WORD y, WORD index)
{
     intin[0] = index;
     ptsin[0] = x;
     ptsin[1] = y;
     contrl[0] = 103;
     contrl[1] = contrl[3] = 1;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	103	Opcode für V_CONTOURFILL
contrl+2	contrl[1]	1	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	1	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	index	
ptsin	ptsin[0]	x	
ptsin+2	ptsin[1]	y	

Parameter:

index: Farbindex (bei negativem Parameter wird gefüllt, bis ein andersfarbiges Pixel angetroffen wird)
x: X-Koordinate des Startpunktes
y: Y-Koordinate des Startpunktes

FILL RECTANGLE (VDI 114)

Diese Funktion füllt ein Rechteck unter Beachtung der Attribute (ähnlich "FILLED AREA" (VDI 9)) aus, ohne "Perimeter" (Umrahmungseinstellung) zu berücksichtigen.

Deklaration in C:

```
void vr_recfl (WORD handle, WORD *pxyarray)
{
     pioff = pxyarray;
     contrl[0] = 114;
     contrl[1] = 2;
     contrl[3] = 0;
     contrl[6] = handle;
     vdi ();
     pioff = ptsin;
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	114	Opcode für VR_RECFL
contrl+2	contrl[1]	2	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
ptsin	ptsin[0..3]	pxyarray[0..3]	

Parameter:

pxyarray[0]: X-Koordinate des Eckpunktes
pxyarray[1]: Y-Koordinate des Eckpunktes
pxyarray[2]: X-Koordinate der diagonal gegenüberliegenden Ecke
pxyarray[3]: Y-Koordinate der diagonal gegenüberliegenden Ecke

Bemerkungen
Vorsicht: Nicht auf allen Ausgabegeräten verfügbar! Im Zweifel besser "v_bar()" benutzen!

GENERALIZED DRAWING PRIMITIVE (VDI 11) (GDP)

Die GDP-Funktion erlaubt eine einfache Handhabung einer Reihe von Grafikgrundfunktionen. Die zehn möglichen Grafikgrundfunktionen sind in der Reihenfolge ihrer Identifikationsnummer aufgeführt.

Einige der Funktionen sind nicht auf allen Ausgabegeräten verfügbar.

BAR (VDI 11, GDP 1)

Die Funktion BAR zeichnet ein ausgefülltes Rechteck. Benutzt wird die Grundfunktion etwa für Balkendiagramme. Die Füllattribute sowie der Schreibmodus werden berücksichtigt.

Deklaration in C:

```
void v_bar (WORD handle, WORD *pxyarray)
{
     pioff = pxyarray;
     contrl[0] = 11;
     contrl[1] = 2;
     contrl[3] = 0;
     contrl[5] = 1;
     contrl[6] = handle;
     vdi ();
     pioff = ptsin;
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	11	Opcode für GDP
contrl+2	contrl[1]	2	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	1	V_BAR
contrl+12	contrl[6]	handle	Gerätekennung
ptsin	ptsin[0..3]	pxyarray[0..3]	

Parameter:

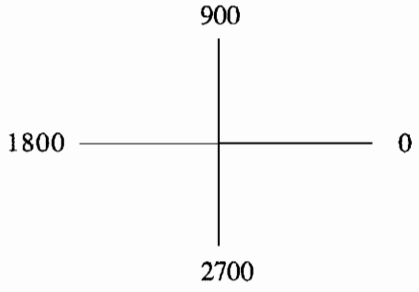
pxyarray[0]: X-Koordinate des Eckpunktes
pxyarray[1]: Y-Koordinate des Eckpunktes
pxyarray[2]: X-Koordinate der diagonal gegenüberliegenden Ecke
pxyarray[3]: Y-Koordinate der diagonal gegenüberliegenden Ecke

ARC (VDI 11, GDP 2)

Die Funktion ARC zeichnet einen Kreisbogen. Es werden die Linienattribute, "Perimeter" (Umrahmung), der Schreibmodus und die tatsächliche Pixelgröße berücksichtigt.

Der Bogen wird entgegen dem Uhrzeigersinn (positiv) gezeichnet. Nulldurchgänge sind durchaus möglich. Die Angabe des Radius bezieht sich auf die X-Achse.

Die Winkelmessung erfolgt in 1/10 Grad:



Deklaration in C:

```
void v_arc (WORD handle, WORD x, WORD y, WORD radius, WORD begang,
            WORD endang)
{
     ptsin[0] = x;
     ptsin[1] = y;
     ptsin[2] = ptsin[3] = ptsin[4] = ptsin[5] = 0;
     ptsin[6] = radius;
     ptsin[7] = 0;
     intin[0] = begang;
     intin[1] = endang;
     contrl[0] = 11;
     contrl[1] = 4;
     contrl[3] = contrl[5] = 2;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	11	Opcode für GDP
contrl+2	contrl[1]	4	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	2	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	2	V_ARC
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	begang	
intin+2	intin[1]	endang	
ptsin	ptsin[0]	x	
ptsin+2	ptsin[1]	y	
ptsin+4	ptsin[2..5]	0	
ptsin+12	ptsin[6]	radius	
ptsin+14	ptsin[7]	0	

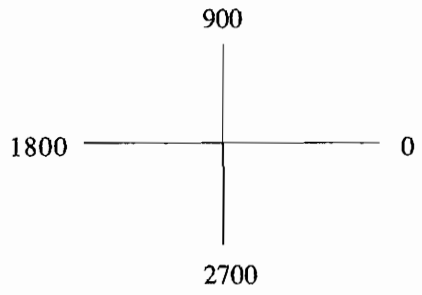
Parameter:

begang:	Startwinkel für den Bogen (0..3600)
endang:	Endwinkel für den Bogen (0..3600)
x:	X-Koordinate für den Mittelpunkt
y:	Y-Koordinate für den Mittelpunkt
radius:	Radius (bzgl. der X-Achse)

PIESLICE (VDI 11, GDP 3)

Die Funktion PIESLICE zeichnet einen Kreisflächenausschnitt, wie er auch für Tortengrafiken benutzt wird. Es werden die Füllattribute, der Schreibmodus und das Pixelgrößenverhältnis berücksichtigt. Der Kreisflächenausschnitt wird entgegen dem Uhrzeigersinn beginnend beim Startwinkel und endend beim Endwinkel gezeichnet, wobei der Radius an der X-Achse gemessen wird. Nulldurchgänge sind durchaus möglich.

Die Winkel werden in 1/10 Grad gemessen:



Deklaration in C:

```
void v_pieslice (WORD handle, WORD x, WORD y, WORD radius,
                 WORD begang, WORD endang)
{
     ptsin[0] = x;
     ptsin[1] = y;
     ptsin[2] = ptsin[3] = ptsin[4] = ptsin[5] = 0;
     ptsin[6] = radius;
     ptsin[7] = 0;
     intin[0] = begang;
     intin[1] = endang;
     contrl[0] = 11;
     contrl[1] = 4;
     contrl[3] = 2;
     contrl[5] = 3;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	11	Opcode für GDP
contrl+2	contrl[1]	4	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	2	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	3	V_PIESLICE
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	begang	
intin+2	intin[1]	endang	
ptsin	ptsin[0]	x	
ptsin+2	ptsin[1]	y	
ptsin+4	ptsin[2..5]	0	
ptsin+12	ptsin[6]	radius	
ptsin+14	ptsin[7]	0	

Parameter:

begang:	Startwinkel für den Kreisausschnitt (0..3600)
endang:	Endwinkel für den Kreisausschnitt (0..3600)
x:	X-Koordinate für den Mittelpunkt
y:	Y-Koordinate für den Mittelpunkt
radius:	Radius (bzgl. der X-Achse)

CIRCLE (VDI 11, GDP 4)

Die Funktion CIRCLE zeichnet eine Kreisfläche. Es werden die Füllattribute, der Schreibmodus und das Pixelgrößenverhältnis berücksichtigt. Der Radius wird anhand der X-Achse bestimmt.

Deklaration in C:

```
void v_circle (WORD handle, WORD x, WORD y, WORD radius)
{
     ptsin[0] = x;
     ptsin[1] = y;
     ptsin[2] = ptsin[3] = 0;
     ptsin[4] = radius;
     ptsin[5] = 0;
     contrl[0] = 11;
     contrl[1] = 3;
     contrl[3] = 0;
     contrl[5] = 4;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	11	Opcode für GDP
contrl+2	contrl[1]	3	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	4	V_CIRCLE
contrl+12	contrl[6]	handle	Gerätekennung
ptsin	ptsin[0]	x	
ptsin+2	ptsin[1]	y	
ptsin+4	ptsin[2..3]	0	
ptsin+8	ptsin[4]	radius	
ptsin+10	ptsin[5]	0	

Parameter:

x: X-Koordinate für den Mittelpunkt
y: Y-Koordinate für den Mittelpunkt
radius: Radius (bzgl. der X-Achse)

ELLIPSE (VDI 11, GDP 5)

Die Funktion ELLIPSE zeichnet eine Ellipsenfläche.

Die Füllattribute und der Schreibmodus werden berücksichtigt. Das Pixelgrößenverhältnis wird ignoriert; der Radius für die X- und Y-Achse ist getrennt zu übergeben.

Deklaration in C:

```
void v_ellipse (WORD handle, WORD x, WORD y, WORD xradius,
                WORD yradius)
{
     ptsin[0] = x;
     ptsin[1] = y;
     ptsin[2] = xradius;
     ptsin[3] = yradius;
     contrl[0] = 11;
     contrl[1] = 2;
     contrl[3] = 0;
     contrl[5] = 5;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	11	Opcode für GDP
contrl+2	contrl[1]	2	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	5	V_ELLIPSE
contrl+12	contrl[6]	handle	Gerätekennung
ptsin	ptsin[0]	x	
ptsin+2	ptsin[1]	y	
ptsin+4	ptsin[2]	xradius	
ptsin+6	ptsin[3]	yradius	

Parameter:

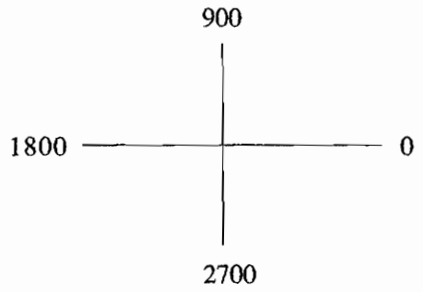
x: X-Koordinate für den Mittelpunkt
y: Y-Koordinate für den Mittelpunkt
xradius: Radius der Ellipse in X-Richtung
yradius: Radius der Ellipse in Y-Richtung

ELLIPTICAL ARC (VDI 11, GDP 6)

Die Funktion ELLIPTICAL ARC zeichnet einen Ellipsenbogenausschnitt. Die Linienattribute und der Schreibmodus werden berücksichtigt.

Das Pixelgrößenverhältnis wird ignoriert, die Radien in X- und Y-Richtung sind also getrennt zu übergeben. Die Ausgabe der Ausschnitts erfolgt entgegen dem Uhrzeigersinn vom Anfangs- bis zum Endwinkel.

Die Winkel werden in 1/10 Grad gemessen:



Deklaration in C:

```
void v_ellarc (WORD handle, WORD x, WORD y, WORD xradius,
               WORD yradius, WORD begang, WORD endang)
{
     ptsin[0] = x;
     ptsin[1] = y;
     ptsin[2] = xradius;
     ptsin[3] = yradius;
     intin[0] = begang;
     intin[1] = endang;
     contrl[0] = 11;
     contrl[1] = contrl[3] = 2;
     contrl[5] = 6;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	11	Opcode für GDP
contrl+2	contrl[1]	2	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	2	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	6	V_ELLARC
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	begang	
intin+2	intin[1]	endang	
ptsin	ptsin[0]	x	
ptsin+2	ptsin[1]	y	
ptsin+4	ptsin[2]	xradius	
ptsin+6	ptsin[3]	yradius	

Parameter:

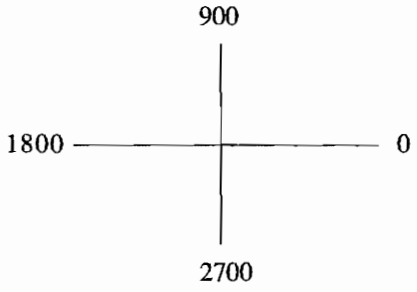
begang:	Startwinkel für den Ellipsenbogen (0..3600)
endang:	Endwinkel für den Ellipsenbogen (0..3600)
x:	X-Koordinate für den Mittelpunkt
y:	Y-Koordinate für den Mittelpunkt
xradius:	Radius der Ellipse in X-Richtung
yradius:	Radius der Ellipse in Y-Richtung

ELLIPTICAL PIE (VDI 11, GDP 7)

Die Funktion ELLIPTICAL PIE zeichnet einen Ellipsenflächenausschnitt, wie er auch für Tortengrafiken benutzt werden kann. Es werden die Füllattribute und der Schreibmodus berücksichtigt.

Das Pixelgrößenverhältnis wird ignoriert, die Radien in X- und Y-Richtung sind folglich getrennt zu betrachten. Der Ausschnitt wird vom Anfangs- bis zum Endwinkel entgegen dem Uhrzeigersinn ausgegeben.

Die Winkel werden in 1/10 Grad gemessen:



Deklaration in C:

```
void v_ellpie (WORD handle, WORD x, WORD y, WORD xradius,
               WORD yradius, WORD begang, WORD endang)
{
     ptsin[0] = x;
     ptsin[1] = y;
     ptsin[2] = xradius;
     ptsin[3] = yradius;
     intin[0] = begang;
     intin[1] = endang;
     contrl[0] = 11;
     contrl[1] = contrl[3] = 2;
     contrl[5] = 7;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	11	Opcode für GDP
contrl+2	contrl[1]	2	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	2	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	7	V_ELLPIE
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	begang	
intin+2	intin[1]	endang	
ptsin	ptsin[0]	x	
ptsin+2	ptsin[1]	y	
ptsin+4	ptsin[2]	xradius	
ptsin+6	ptsin[3]	yradius	

Parameter:

begang:	Startwinkel für den Ellipsenausschnitt (0..3600)
endang:	Endwinkel für den Ellipsenausschnitt (0..3600)
x:	X-Koordinate für den Mittelpunkt
y:	Y-Koordinate für den Mittelpunkt
xradius:	Radius der Ellipse in X-Richtung
yradius:	Radius der Ellipse in Y-Richtung

ROUNDED RECTANGLE (VDI 11, GDP 8)

Die Funktion ROUNDED RECTANGLE zeichnet ein Rechteck mit abgerundeten Ecken. Es werden die Linienattribute, der Schreibmodus und das Pixelgrößenverhältnis (bezüglich der Bögen) berücksichtigt.

Deklaration in C:

```
void v_rbox (WORD handle, WORD *xyarray)
{
     pioff = pxyarray;
     contrl[0] = 11;
     contrl[1] = 2;
     contrl[3] = 0;
     contrl[5] = 8;
     contrl[6] = handle;
     vdi ();
     pioff = ptsin;
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	11	Opcode für GDP
contrl+2	contrl[1]	2	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	8	V_RBOX
contrl+12	contrl[6]	handle	Gerätekennung
ptsin	ptsin[0..3]	xyarray[0..3]	

Parameter:

xyarray[0]: X-Koordinate des Eckpunktes
xyarray[1]: Y-Koordinate des Eckpunktes
xyarray[2]: X-Koordinate der diagonal gegenüberliegenden Ecke
xyarray[3]: Y-Koordinate der diagonal gegenüberliegenden Ecke

FILLED ROUNDED RECTANGLE (VDI 11, GDP 9)

Die Funktion FILLED ROUNDED RECTANGLE zeichnet ein ausgefülltes Rechteck mit abgerundeten Ecken. Es werden Füllattribute, der Schreibmodus und das Pixelgrößenverhältnis (bezüglich der Bögen) berücksichtigt.

Deklaration in C:

```
void v_rfbox (WORD handle, WORD *xyarray)
{
     pioff = xyarray;
     contrl[0] = 11;
     contrl[1] = 2;
     contrl[3] = 0;
     contrl[5] = 9;
     contrl[6] = handle;
     vdi ();
     pioff = ptsin;
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	11	Opcode für GDP
contrl+2	contrl[1]	2	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	9	V_RFBOX
contrl+12	contrl[6]	handle	Gerätekennung
ptsin	ptsin[0..3]	xyarray[0..3]	

Parameter:

xyarray[0]: X-Koordinate des Eckpunktes
xyarray[1]: Y-Koordinate des Eckpunktes
xyarray[2]: X-Koordinate der diagonal gegenüberliegenden Ecke
xyarray[3]: Y-Koordinate der diagonal gegenüberliegenden Ecke

JUSTIFIED GRAPHICS TEXT (VDI 11, GDP 10)

JUSTIFIED GRAPHICS TEXT erlaubt die wenig aufwendige Ausgabe eines Textes, genauer einer Zeichenkette.

Der Text wird links und rechts bezogen auf den Startpunkt und die Länge justiert. Um den Text auf die erforderliche Länge zu bringen, hat man die Wahl zwischen Dehnung der Wort- und/oder Zeichenzwischenräume.

Die Ausdehnung kann auch unterbleiben. Es werden die Textattribute und der Schreibmodus berücksichtigt.

Bei einem C-Binding ist der String Byte-orientiert und Null-terminiert, bei einem Assembler-Binding Word-orientiert.

Im Zeichensatz fehlende Zeichen werden durch ein Ersatzsymbol (meist ein Fragezeichen) ersetzt.

Deklaration in C:

```
void v_justified (WORD handle, WORD x, WORD y, const CHAR *string,
                  WORD length, WORD word_space, WORD char_space)
{
     WORD *tmp;

     ptsin[0] = x;
     ptsin[1] = y;
     ptsin[2] = length;
     ptsin[3] = 0;
     intin[0] = word_space;
     intin[1] = char_space;
     tmp = &(intin[2]);
     while (*tmp++=*string++) ;
     contrl[0] = 11;
     contrl[1] = 2;
     contrl[3] = (int)(tmp-&(intin))-1;
     contrl[5] = 10;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung
contrl	contrl[0]	11 Opcode für GDP
contrl+2	contrl[1]	2 # Einträge in ptsin
contrl+4	contrl[2]	0 # Einträge in ptsout
contrl+6	contrl[3]	n+2 # Einträge in intin
contrl+8	contrl[4]	0 # Einträge in intout
contrl+10	contrl[5]	10 V_JUSTIFIED
contrl+12	contrl[6]	handle Gerätekennung
intin	intin[0]	word_space
intin+2	intin[1]	char_space
intin+4	intin[2..n+1]	string[0..n-1]
ptsin	ptsin[0]	x
ptsin+2	ptsin[1]	y
ptsin+4	ptsin[2]	length
ptsin+6	ptsin[3]	0

Parameter:

word_space: Flag für Wortzwischenräume
0: keine Dehnung der Wortzwischenräume
nicht 0: Dehnung der Wortzwischenräume
char_space: Flag für Zeichenzwischenräume
0: keine Dehnung der Zeichenzwischenräume
nicht 0: Dehnung der Zeichenzwischenräume
string: Zeiger auf Zeichenkette
x: X-Koordinate des Textausrichtungspunktes
y: Y-Koordinate des Textausrichtungspunktes
length: Textlänge (bzgl. der X-Achse)

Attributfunktionen

SET WRITING MODE (VDI 32)

SET WRITING MODE bestimmt, auf welche Art und Weise Quellpixel (also zum Beipiel die aktuelle Linienfarbe) und Zielpixel (also der aktuelle Hintergrund) verknüpft werden, um die resultierenden Farbwerte zu ermitteln. Normaler Schreibmodus ist der REPLACE-Modus.

Die Anzahl der verfügbaren Schreibmodi erhält man über die Funktion EXTENDED INQUIRE FUNCTION.

Deklaration in C:

```
WORD vswr_mode (WORD handle, WORD mode)
{
     intin[0] = mode;
     contrl[0] = 32;
     contrl[1] = 0;
     contrl[3] = 1;
     contrl[6] = handle;
     vdi ();
     return intout[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	32	Opcode für VSWR_MODE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	1	# Einträge in intin
contrl+8	contrl[4]	1	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	mode	
intout	intout[0]	Return-Wert	

Parameter:

mode:	Schreibmodus	
	MD_REPLACE (1):	Replace
	MD_TRANS (2):	Transparent
	MD_XOR (3):	XOR
	MD_ERASE (4):	Reverse Transparent
vswr_mode():	ausgewählter Schreibmodus (bei Wahl eines ungültigen Parameters wird der Replace-Modus gesetzt)	

Die Verknüpfungen der Schreibmodi:

Folgende Bezeichnungen werden benutzt:

Maske:	Grafikobjekt (Linie, Füllmuster, Grafiktext, Marker)
Vorn:	Vordergrundfarbe des neuen Grafikobjektes
Hinten:	Hintergrundfarbe des neuen Grafikobjektes
Alt:	gegenwärtige Farbe
Neu:	resultierende Farbe

Die Erfahrung zeigt, daß man die Unterschiede zwischen den verschiedenen Modi am besten anhand eines Beispiels versteht. Daher lehnen wir uns in den folgenden Ausführungen an das

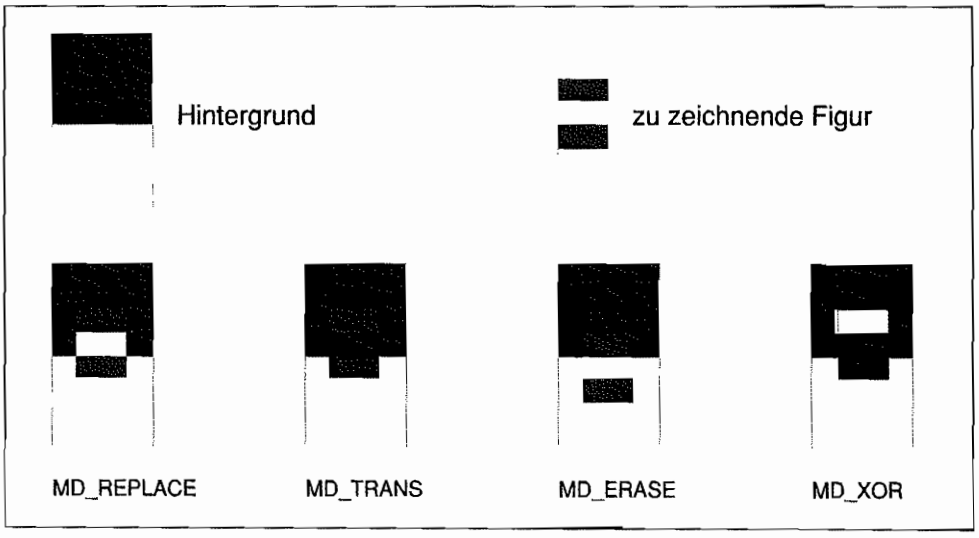


Abb. 3.3: Die Verknüpfungen der Schreibmodi

"GEM Programmier-Handbuch" (ebenfalls bei SYBEX erschienen, mittlerweile leider aber nicht mehr erhältlich) an.

In der folgenden Beschreibung der Verknüpfungen gehen wir von der in Abbildung 3.3 gezeigten Sachlage aus: der Hintergrund ist in eine schwarze und eine weiße Hälfte geteilt. Auf diesem Hintergrund wird nun die rechts gezeigte Figur ausgegeben. Die gemusterten Flächen stehen für gesetzte Bits, die weißen für nicht gesetzte. Die folgenden vier Beispiele zeigen die Auswirkungen der verschiedenen Verknüpfungen:

MD_REPLACE
Diese Darstellungsart ist einfach zu verstehen. Wenn das VDI ein Bild ausgibt, überschreibt es den Hintergrund mit der Vordergrundfarbe, wenn das entsprechende Bit gesetzt ist, oder es setzt ein weißes Pixel, wenn das Bit 0 ist. Die logische Verknüpfung lautet:

```
Neu := (Vorn AND Maske) OR (Hinten AND NOT maske).
```

Um das Ganze auf Papier und Bleistift zu übertragen: zunächst malt man die Figur auf weißes Papier, schneidet sie dann aus und klebt den Schnipsel dann auf die Grafik.

MD_TRANS
Bei der transparenten Darstellung ignoriert das VDI alle Bits der zu zeichnenden Form, die auf 0 gesetzt sind, und gibt lediglich die Bits aus, die gesetzt sind. Die logische Verknüpfung lautet:

```
Neu := (Vorn AND Maske) OR (ALT AND NOT Maske)
```

Und wieder die Papieranalogie: dieser Modus entspricht dem Ersetzen, nur daß man auf durchsichtige Folie zeichnet.

MD_XOR
Diese Darstellungsart eignet sich vorzüglich für das Zeichnen von Figuren, die ohne Änderung des Hintergrunds wieder gelöscht oder bewegt werden sollen. Die Bits der Hintergrunds und der zu zeichnenden Figur werden mit einem Exklusiv-ODER (XOR) verknüpft. Diese Verknüpfung hat eine sehr interessante Eigenschaft.

Wenn Sie die Figur eine zweites Mal in der Darstellungsart XOR ausgeben, ist die Figur plötzlich wieder verschwunden. Der Grund hierfür liegt in der Tatsache, daß die XOR-Verknüpfung eines Bits mit sich selbst den inversen Wert ergibt.

Sie können diese Darstellungsart für einfache Animationen wie eine Gummibox einsetzen. Auch bewegte Objekte können auf diese Art und Weise gezeichnet werden:

1. Das Objekt einmal ausgeben.
2. Das gleiche Objekt erneut ausgeben (nun ist es wieder verschwunden).
3. Objekt verändern (Größe, Position).
4. Zurück zu Schritt 1.

Die logische Verküpfung für MD_XOR:

```
Neu:= (Maske XOR Alt)
```

Man beachte, daß das Resultat völlig unabhängig von der eingestellen Ausgabefarbe ist!

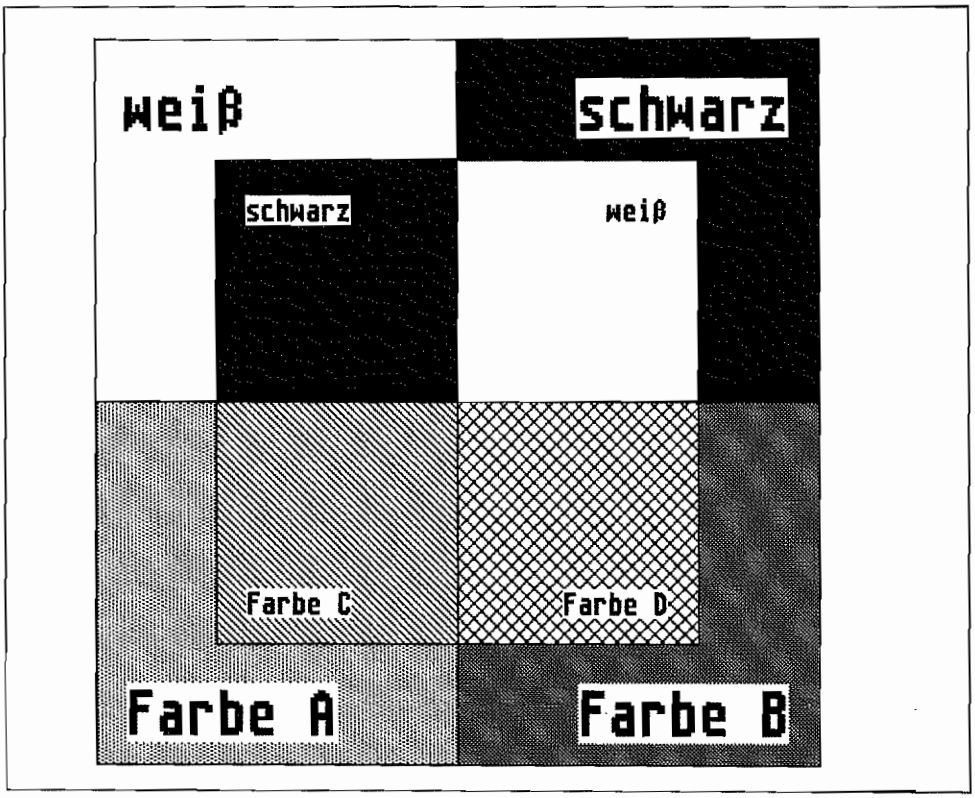


Abb. 3.4: Der XOR-Modus

In Farbauflösungen werden, wie auch im monochromen Modus, die Pixelwerte exklusiv-oderiert, jedoch ist der Effekt schwerer vorherzusehen. Im XOR-Modus werden die Bits gemäß der gemalten Figur mit XOR logisch verknüpft, unabhängig von der Farbe des Musters oder der gemalten Figur!

Beispielsweise wird aus einem Pixelwert (bei vier Farbebenen) 0x1011 in Verbindung mit XOR 0x0100. Vor allem ist damit klar, daß aus Schwarz Weiß wird und umgekehrt. Alles andere ist nicht fest definiert. Stehen vier Farben zur Verfügung (beispielsweise niedrige ST-Auflösung), so wird aus Rot durch XOR Grün und umgekehrt. Stehen jedoch 16 Farben zur Verfügung (beispielsweise mittlere TT-Auflösung), so wird aus Rot Dunkelcyan und aus Grün Dunkelmagenta! Der monochrome Modus ist folglich nur ein (sehr einfacher) Spezialfall!

MD_ERASE ("Reverse Transparent")
Bei der invers transparenten Darstellung werden nur die auf 0 gesetzten Pixel des zu zeichnenden Bildes berücksichtigt (im Gegensatz zur transparenten Darstellung). Alle nicht gesetzten Bits werden beim Zeichnen in der Vordergrundfarbe ausgegeben.

Diese Darstellungsart bietet einige interessante Einsatzmöglichkeiten. Sie können die transparente und die invers transparente Darstellung gemeinsam benutzen, um beispielsweise eine zweifarbige unterbrochene Linie oder Text mit einer zusätzlichen Hintergrundfarbe erscheinen zu lassen.

In beiden Fällen geben Sie die Figur erst in der transparenten Darstellung mit einer Vordergrundfarbe aus, ändern dann die Vordergrundfarbe und geben die Figur ein zweites Mal mit invers transparenter Darstellung aus.

Die logische Verknüpfung für MD_ERASE:

```
Neu := (Alt AND Maske) OR (Vorn AND NOT Maske)
```

SET COLOR REPRESENTATION (VDI 14)

Diese Funktion wählt die Farbintensität zu den einzelnen Farbregistern. Gewählt wird mit einer Intensität der RGB-Farben (Rot, Grün, Blau) zwischen 0 und 1000 (Promille). Sollte eine ungültige Farbnummer übergeben werden, so werden die erreichbaren Werte angenommen, also bei negativen Werten 0 und bei zu großen Werten 1000.

Üblicherweise verfügt nicht jedes Ausgabegerät über 1000 mögliche Farbabstufungen. Folglich sind für verschiedene Eingabe-Intensitäten der RGB-Farben die tatsächlich eingestellten Intensitäten gleich. Die Anzahl der Farbindizes ist geräteabhängig und kann beim Öffnen der (virtuellen) Workstation abgefragt werden.

SET COLOR REPRESENTATION kann nur benutzt werden, wenn "Lookup-table"-Unterstützung vorhanden ist (anderenfalls gibt es ja keine Farbregister, die man setzen könnte).

Deklaration in C:

```
void vs_color (WORD handle, WORD index, WORD *rgb_in)
{
     WORD i;
     intin[0] = index;
     for (i=1; i<4; i++)
          intin[i] = *rgb_in++;
     contrl[0] = 14;
     contrl[1] = 0;
     contrl[3] = 4;
     contrl[6] = handle:
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	14	Opcode für VS_COLOR
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	4	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung

Adresse	Feldelement	Belegung
intin	intin[0]	index
intin+2	intin[1..3]	rgb_in[0..2]

Parameter:

index: Farbnummer (die ersten 16 Farbnummern werden vom AES benutzt und sollten deshalb nicht verändert werden!)
rgb_in[0]: Farbintensität von Rot
rgb_in[1]: Farbintensität von Grün
rgb_in[2]: Farbintensität von Blau

Tabelle der VDI-Standardfarben:

Index	Farbe	Bezeichnung
0	weiß	WHITE
1	schwarz	BLACK
2	rot	RED
3	grün	GREEN
4	blau	BLUE
5	cyan	CYAN
6	gelb	YELLOW
7	magenta	MAGENTA
8	hellgrau	LWHITE
9	dunkelgrau	LBLACK
10	dunkelrot	LRED
11	dunkelgrün	LGREEN
12	dunkelblau	LBLUE
13	dunkelcyan	LCYAN
14	dunkelgelb	LYELLOW
15	dunkelmagenta	LMAGENTA
darüber:	geräteabhängig	

SET POLYLINE LINE TYPE (VDI 15)

Diese Funktion wählt die (Polyline-)Liniendarstellung aus. Die Anzahl der verfügbaren Darstellungen ist quasi unbegrenzt. Sechs Linienformen sind für alle Geräte fest vorgegeben. Weitere sind geräteabhängig bzw. können selbst definiert werden.

Ist ein gewählter Linientyp nicht verfügbar, so wird der Linientyp "durchgezogen" (oder genauer: LT_SOLID) gewählt. Bei der Benutzung dickerer Linien wird unter Umständen auf den Defaulttyp gewechselt und auch der Schreibmodus verändert.

Die entsprechenden Verfügbarkeiten bei dem Ausgabegerät können mit EXTENDED INQUIRE FUNCTION abgefragt werden.

Deklaration in C:

```
WORD vsl_type (WORD handle, WORD style)
{
     intin[0] = style;
     contrl[0] = 15;
     contrl[1] = 0;
     contrl[3] = 1;
     contrl[6] = handle;
     vdi ();
     return intout[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	15	Opcode für VSL_TYPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	1	# Einträge in intin
contrl+8	contrl[4]	1	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	style	
intout	intout[0]	Return-Wert	

Parameter:

style: Linienstil (1 = gesetztes Pixel, 0 = nicht gesetztes Pixel), 16 Bits

style	MSB LSB	
LT_SOLID (1)	1111111111111111	(durchgehend)
LT_LONGDASH (2)	1111111111110000	(langer Strich)
LT_DOTTED (3)	1110000011100000	(Punkte)
LT_DASHDOT (4)	1111111000111000	(Strich, Punkt)
LT_DASHED (5)	1111111100000000	(Strich)
LT_DASHDOTDOT (6)	1111000110011000	(Strich, Punkt, Punkt)
LT_USERDEF (7)	frei definierbar	(Default: LT_SOLID)
8 und mehr	abhängig vom Ausgabegerät	

vsl_type(): ausgewählter Linienstil

SET USER-DEFINED LINE STYLE PATTERN (VDI 113)

Mit dieser Funktion kann der siebte (frei definierbare) Linientyp von SET POLYLINE LINE TYPE (VDI 15) festgelegt werden. Als erster Punkt der Linie wird das höchstwertige Bit des 16-Bit-Wortes gewählt. Standardwert ist eine durchgezogene Linie.

Deklaration in C:

```
void vsl_udsty (WORD handle, WORD pattern)
{
     intin[0] = pattern;
     contrl[0] = 113;
     contrl[1] = 0;
     contrl[3] = 1;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	113	Opcode für VSL_UDSTY
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	1	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	pattern	

Parameter:

pattern: Linienmuster als 16-Bit-Wort

SET POLYLINE LINE WIDTH (VDI 16)

Mit dieser Funktion wählt man die Linienbreite. Die Funktion ist nicht auf allen Ausgabegeräten einsetzbar. Die gesetzte Breite ist kleiner oder gleich der gewählten Breite. Eine Korrektur der Breite seitens der Funktion kann nötig sein, wenn kein passender Wert (eine ungerade Zahlen) übergeben wurde. Die Breite der Linie bezieht sich immer auf Koordinaten in X-Richtung.

Deklaration in C:

```
WORD vsl_width (WORD handle, WORD width)
{
     ptsin[0] = width;
     ptsin[1] = 0;
     contrl[0] = 16;
     contrl[1] = 1;
     contrl[3] = 0;
     contrl[6] = handle;
     vdi ();
     return ptsout[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	16	Opcode für VSL_WIDTH
contrl+2	contrl[1]	1	# Einträge in ptsin
contrl+4	contrl[2]	1	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
ptsin	ptsin[0]	width	
ptsin+2	ptsin[1]	0	
ptsout	ptsout[0]	Return-Wert	
ptsout+2	ptsout[1]	0	

Parameter:

width:	Linienbreite (nur ungerade Werte ab 1; bei falscher Wahl wird die nächstkleinere mögliche Linienbreite gewählt)
vsl_width():	ausgewählte Linienbreite (bezüglich der X-Achse)

SET POLYLINE COLOR INDEX (VDI 17)

Mit dieser Funktion kann die Linienfarbe gewählt werden. Die zwei Standardindizes 0 und 1 sind auf allen Geräten verfügbar. Weitere hängen vom Ausgabegerät ab.

Deklaration in C:

```
WORD vsl_color (WORD handle, WORD color_index)
{
     intin[0] = color_index;
     contrl[0] = 17;
     contrl[1] = 0;
     contrl[3] = 1;
     contrl[6] = handle;
     return intout[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	17	Opcode für VSL_COLOR
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	1	# Einträge in intin
contrl+8	contrl[4]	1	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	color_index	
intout	intout[0]	Return-Wert	

Parameter:

color_index: Linienfarbe
vsl_color(): ausgewählte Linienfarbe (bei ungültigem Farbindex: 1)

SET POLYLINE END STYLES (VDI 108)

Mit dieser Funktion wird das Aussehen der Linienenden gewählt. Das normale Aussehen der Enden ist ein eckiger Linienabschluß.

Zusätzlich gibt es noch abgerundete Enden oder Enden mit Pfeilspitzen. Auch ein ganzer Polygonzug wird als Linie betrachtet. Das Ende der Linie ist bei der Pfeilspitze in der Spitze, bei dem abgerundeten Ende jedoch im Zentrum des (Halb-)Kreises, der das Ende darstellt.

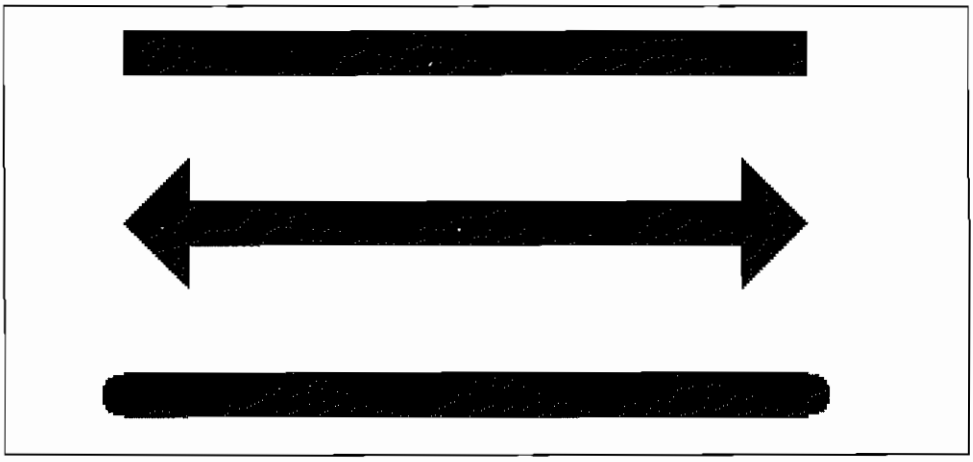


Abb. 3.5: Die verschiedenen Linien-Endstile

Deklaration in C:

```
void vsl_ends (WORD handle, WORD beg_style, WORD end_style)
{
     intin[0] = beg_style;
     intin[1] = end_style;
     contrl[0] = 108;
     contrl[1] = 0;
     contrl[3] = 2;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	108	Opcode für VSL_ENDS
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	2	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	beg_style	
intin+2	intin[1]	end_style	

Parameter:

beg_style: Aussehen des Linienendes am Anfangspunkt der Linie.
LE_SQUARED (0): eckig
LE_ARROWED (1): Pfeilform
LE_ROUNDED (2): abgerundet

end_style: analog mit dem Endpunkt

SET POLYMARKER TYPE (VDI 18)

Mit dieser Funktion kann das Aussehen der Marker ausgewählt werden, von denen es mindestens die aufgeführten sechs gibt. Bei Übergabe einer ungültigen Nummer wird der dritte Typ gewählt. Der erste Typ (der Punkt) kann in der Größe nicht verändert werden.

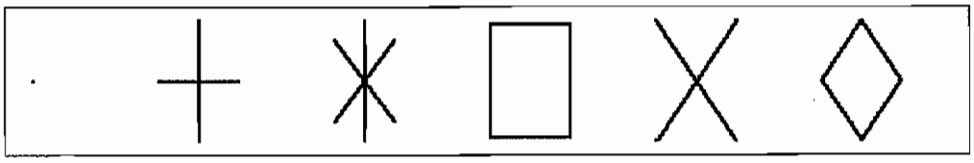


Abb. 3.6: Die Markertypen

Deklaration in C:

```
WORD vsm_type (WORD handle, WORD symbol)
{
     intin[0] = symbol;
     contrl[0] = 18;
     contrl[1] = 0;
     contrl[3] = 1;
     contrl[6] = handle;
     vdi ();
     return intout[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	18	Opcode für VSM_TYPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	1	# Einträge in intin
contrl+8	contrl[4]	1	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	symbol	
intout	intout[0]	Return-Wert	

Parameter:

symbol:	Markertyp	
	MT_DOT (1):	Punkt
	MT_PLUS (2):	Plus
	MT_ASTERISK (3):	Sternchen
	MT_SQUARE (4):	Quadrat
	MT_DCROSS (5):	Kreuz
	MT_DIAMOND (6):	Raute
	über 7:	geräteabhängig
vsm_type():	ausgewählter Markertyp (bei falschen Eingabewerten wird MT_ASTERISK gewählt)	

SET POLYMARKER HEIGHT (VDI 19)

Mit dieser Funktion kann die Markerhöhe (außer bei Typ 1) bezüglich der Y-Achse gewählt werden, wobei die Breite angepaßt wird.

Bei Angabe einer ungültigen (zu großen) Höhe wird die nächstkleinere gewählt. Die Anzahl der verfügbaren Höhen wird beim Öffnen der (virtuellen) Workstations zurückgegeben.

Deklaration in C:

```
WORD vsm_height (WORD handle, WORD height)
{
     ptsin[0] = 0;
     ptsin[1] = height;
     contrl[0] = 19;
     contrl[1] = 1;
     contrl[3] = 0;
     contrl[6] = handle;
     vdi ();
     return ptsout[1];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	19	Opcode für VSM_HEIGHT
contrl+2	contrl[1]	1	# Einträge in ptsin
contrl+4	contrl[2]	1	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
ptsin	ptsin[0]	0	
ptsin+2	ptsin[1]	height	
ptsout	ptsout[0]	set_height_x	
ptsout+2	ptsout[1]	Return-Wert	

Parameter:

height:	Markerhöhe bzgl. Y-Richtung (bei falscher Wahl wird die nächstkleinere mögliche Größe gewählt)
set_height_x:	ausgewählte Markerhöhe bzgl. X-Richtung (das Pixelgrößenverhältnis wird berücksichtigt); bei PC-GEM erhält man lediglich eine 0 als Rückgabewert
vsm_height():	ausgewählte Markerhöhe bzgl. Y-Richtung

SET POLYMARKER COLOR INDEX (VDI 20)

Mit dieser Funktion wählt man die Farbe der Marker. Zwei Farbindizes (0 und 1) sind auf jeden Fall verfügbar, weitere hängen von dem jeweiligen Ausgabegerät ab. Ein ungültiger Farbindex wird durch den Index 1 ersetzt.

Deklaration in C:

```
WORD vsm_color (WORD handle, WORD color_index)
{
     intin[0] = color_index;
     contrl[0] = 20;
     contrl[1] = 0;
     contrl[3] = 1;
     contrl[6] = handle;
     vdi ();
     return intout[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	20	Opcode für VSM_COLOR
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	1	# Einträge in intin
contrl+8	contrl[4]	1	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	color_index	
intout	intout[0]	Return-Wert	

Parameter:

color_index: gewünschte Markerfarbe
vsm_color(): ausgewählte Markerfarbe (1 bei ungültigem Farbindex)

SET CHARACTER HEIGHT, ABSOLUTE MODE (VDI 12)

Mit dieser Funktion kann die absolute Zeichenhöhe, von der Basislinie bis zur Zeichenzellenobergrenze, bestimmt werden. Ist die gewünschte Zeichenhöhe nicht verfügbar, so wählt SET CHARACTER HEIGHT die nächstkleinere verfügbare.

Es wird die größte Zeichen- und Zellenbreite zurückgegeben. Sie ist somit insbesondere bei Proportionalfonts gleich der Breite des breitesten Zeichens.

Bei nicht proportionalen Schriften sind logischerweise alle Zeichen gleich breit.

Deklaration in C:

```
void vst_height (WORD handle, WORD height, WORD *char_width,
                 WORD *char_height, WORD *cell_width,
                 WORD *cell_height)
{
     ptsin[0] = 0;
     ptsin[1] = height;
     contrl[0] = 12;
     contrl[1] = 1;
     contrl[3] = 0;
     contrl[6] = handle;
     vdi ();
     *char_width = ptsout[0];
     *char_height = ptsout[1];
     *cell_width = ptsout[2];
     *cell_height = ptsout[3];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	12	Opcode für VST_HEIGHT
contrl+2	contrl[1]	1	# Einträge in ptsin
contrl+4	contrl[2]	2	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung

Adresse	Feldelement	Belegung
ptsin	ptsin[0]	0
ptsin+2	ptsin[1]	height
ptsout	ptsout[0]	char_width
ptsout+2	ptsout[1]	char_height
ptsout+4	ptsout[2]	cell_width
ptsout+6	ptsout[3]	cell_height

Parameter:

height: Zeichenhöhe (bei falscher Wahl wird die nächstkleinere mögliche Zeichensatzgröße eingesetzt)
char_width: ausgewählte Zeichenbreite in NDC/RC-Einheiten bezüglich der X-Achse
char_height: ausgewählte Zeichenhöhe in NDC/RC-Einheiten bezüglich der Y-Achse
cell_width: ausgewählte Zeichenzellenbreite in NDC/RC-Koordinaten bezüglich der X-Achse
cell_height: ausgewählte Zeichenzellenhöhe in NDC/RC-Koordinaten bezüglich der Y-Achse

Bemerkungen
Die meisten Bildschirmtreiber (auch der im ROM) können bei "vst_height()" vorhandene Fonts beliebig verkleinern oder auf das Doppelte vergrößern.

SET CHARACTER HEIGHT, POINTS MODE (VDI 107)

Mit dieser Funktion kann die Zeichenzellengröße – Abstand zweier Basislinien – bestimmt werden. Gemessen wird der Abstand in (Drucker-)Point (1/72 Zoll, ungefähr 0,353 mm; dies entspricht nicht dem typografischen Punkt mit einer Höhe von ca. 0,38 mm!).

Ist die gewünschte Zeichenhöhe nicht verfügbar, so wird die nächstkleinere gewählt. Bei proportionalen Zeichensätzen werden für Breite und Höhe die maximal möglichen Werte zurückgegeben.

Deklaration in C:

```
WORD vst_point (WORD handle, WORD point, WORD *char_width,
                WORD *char_height, WORD *cell_width,
                WORD *cell_height)
{
     intin[0] = point;
     contrl[0] = 107;
     contrl[1] = 0;
     contrl[3] = 1;
     contrl[6] = handle;
     vdi ();
     *char_width = ptsout[0];
     *char_height = ptsout[1];
     *cell_width = ptsout[2];
     *cell_height = ptsout[3];
     return intout[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	107	Opcode für VST_POINT
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	2	# Einträge in ptsout
contrl+6	contrl[3]	1	# Einträge in intin
contrl+8	contrl[4]	1	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung

Adresse	Feldelement	Belegung
intin	intin[0]	point
intout	intout[0]	Return-Wert
ptsout	ptsout[0]	char_width
ptsout+2	ptsout[1]	char_height
ptsout+4	ptsout[2]	cell_width
ptsout+6	ptsout[3]	cell_height

Parameter:

point: Zeichenzellenhöhe in Point (bei falscher Wahl wird die nächstkleinere mögliche Zeichensatzgröße eingesetzt)
vst_point(): Ausgewählte Zeichenzellenhöhe in Point
char_width: Ausgewählte Zeichenbreite in NDC/RC-Einheiten bezüglich der X-Achse
char_height: Ausgewählte Zeichenhöhe in NDC/RC-Einheiten bezüglich der Y-Achse
cell_width: Ausgewählte Zeichenzellenbreite in NDC/RC-Einheiten bezüglich der X-Achse
cell_height: Ausgewählte Zeichenzellenhöhe in NDC/RC-Einheiten bezüglich der Y-Achse

Bemerkungen

Der ROM-Bildschirmtreiber kann vorhandene Zeichensätze jeweils auf die doppelte Größe skalieren. Daher ist es nicht ohne weiteres möglich, zwischen "skalierten" und tatsächlich vorhandenen Fonts zu unterscheiden. Die folgende Routinen ermittelt alle vorhandenen Punktgrößen für den aktuell eingestellten Zeichensatz. Sie basiert darauf, daß "vst_point()" im Zweifel die nächstkleinere vorhandene Größe wählt:

```
/* Anfangsgröße: 999 Punkt */
WORD asked_for = 999, got_size;
got_size = asked_for;

/* Solange die zuletzt gefundene Größe tatsächlich kleiner oder
gleich der verlangten ist */
while (got_size <= asked_for)
{
     /* Versuch: ein Punkt kleiner als der zuletzt gefundene */
        got_size = vst_point (handle, asked_for, &dummy, &dummy,
                              &dummy, &dummy);
       printf ("available size: %d\n", got_size);
}
```

Wenn man feststellt, daß *alle* Punktgrößen vorhanden sind, liegt offensichtlich ein freiskalierbarer Zeichensatz vor (wie etwa bei einem VDI-Postscript-Treiber). In einem solchen Fall sollte man dem Anwender nur eine Liste von Standardgrößen anzeigen und die exakte Größenangabe über ein Edit-Feld erlauben.

SET CHARACTER BASELINE VECTOR (VDI 13)

Mit dieser Funktion kann die Ausrichtung der Basislinie (für Textausgabe) gewählt werden. Gemessen wird in 1/10 Grad. Auf einigen Geräten ist die Funktion nicht oder nur in Teilen verfügbar. Der Bildschirmtreiber im ROM unterstützt nur die Rotation in 90-Grad-Schritten. SET CHARACTER BASELINE VECTOR nimmt den Winkel, der dem gewünschten am nächsten liegt.

Deklaration in C:

```
WORD vst_rotation (WORD handle, WORD angle)
{
     intin[0] = angle;
     contrl[0] = 13;
     contrl[1] = 0;
     contrl[3] = 1;
     contrl[6] = handle;
     vdi ();
     return intout[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	13	Opcode für VST_ROTATION
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	1	# Einträge in intin
contrl+8	contrl[4]	1	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	angle	
intout	intout[0]	Return-Wert	

Parameter:

angle: gewünschte Basislinienausrichtung (0..3600)
vst_rotation(): ausgewählte Basislinienausrichtung

SET TEXT FACE (VDI 21)

Diese Funktion wählt – sofern verfügbar – einen Zeichensatz für die nachfolgenden Grafiktextausgaben aus. Die Unterstützung der Zeichensätze hängt vom jeweiligen Gerät ab. Die Namen und Indizes werden über INQUIRE FACE NAME AND INDEX bestimmt.

Deklaration in C:

```
WORD vst_font (WORD handle, WORD font)
{
     intin[0] = font;
     contrl[0] = 21;
     contrl[1] = 0;
     contrl[3] = 1;
     contrl[6] = handle;
     vdi ();
     return intout[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	21	Opcode für VST_FONT
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	1	# Einträge in intin
contrl+8	contrl[4]	1	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	font	
intout	intout[0]	Return-Wert	

Parameter:

font: Zeichensatzindex (die konkreten Nummern muß man mittels "INQUIRE FACE NAME AND INDEX" abfragen!)

vst_font(): ausgewählter Zeichensatz

SET GRAPHIC TEXT COLOR INDEX (VDI 22)

Mit dieser Funktion kann die Farbe des Textes gewählt werden. Verfügbar sind mindestens die Indizes 0 und 1, weitere sind vom jeweiligen Ausgabegerät abhängig. Bei Übergabe eines ungültigen Farbindexes wählt SET GRAPHIC TEXT COLOR INDEX den Index 1.

Deklaration in C:

```
WORD vst_color (WORD handle, WORD color_index)
{
     intin[0] = color_index;
     contrl[0] = 22;
     contrl[1] = 0;
     contrl[3] = 1;
     contrl[6] = handle;
     vdi ();
     return intout[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	22	Opcode für VST_COLOR
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	1	# Einträge in intin
contrl+8	contrl[4]	1	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	color_index	
intout	intout[0]	Return-Wert	

Parameter:

color_index: gewünschte Textfarbe
vst_color(): ausgewählte Textfarbe (1 bei Übergabe eines ungültigen Indexes)

SET GRAPHIC TEXT SPECIAL EFFECTS (VDI 106)

Mit dieser Funktion können verschiedene spezielle Texteffekte ausgewählt werden. Es besteht die Möglichkeit, diese Effekte untereinander zu vermischen.

Als Effekte stehen zur Verfügung:

- fett
- hell
- kursiv
- unterstrichen
- umrandet, ausgehöhlt
- schattiert
- jede Kombination der obengenannten Effekte

Nicht verfügbare Texteffekte werden nicht gesetzt. Die möglichen Grundeffekte:

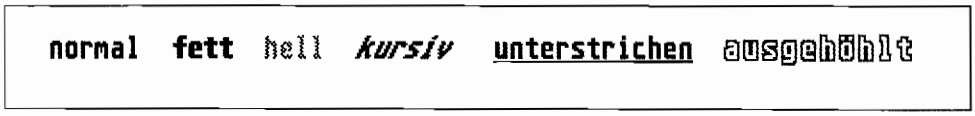


Abb. 3.7: Die möglichen Texteffekte

Deklaration in C:

```
WORD vst_effects (WORD handle, WORD effect)
{
     intin[0] = effect;
     contrl[0] = 106;
     contrl[1] = 0;
     contrl[3] = 1;
     contrl[6] = handle;
     vdi ();
     return intout[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	106	Opcode für VST_EFFECTS
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	1	# Einträge in intin
contrl+8	contrl[4]	1	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	effect	
intout	intout[0]	Return-Wert	

Parameter:

effect: gewünschter Texteffekt in Bitdarstellung

TF_NORMAL	(0x00):	normaler Text
TF_THICKENED	(0x01):	Fettschrift
TF_LIGHTENED	(0x02):	helle Schrift
TF_SLANTED	(0x04):	kursive Schrift
TF_UNDERLINED	(0x08):	unterstrichene Schrift
TF_OUTLINED	(0x10):	ausgehöhlte, umrandete Schrift
TF_SHADOWED	(0x20):	schattierte Schrift (im ROM-Treiber nicht implementiert)

vst_effects(): ausgewählter Texteffekt

SET GRAPHIC TEXT ALIGNMENT (VDI 39)

Mit dieser Funktion kann die horizontale (bezüglich der Basislinie) und vertikale Ausrichtung (senkrecht zur Basislinie) eines Textes bestimmt werden. Horizontal sind drei und vertikal sechs Möglichkeiten verfügbar.

Defaulteinstellung ist der linke Rand der Basislinie (und nicht die untere linke Ecke der linken Zeichenzelle).

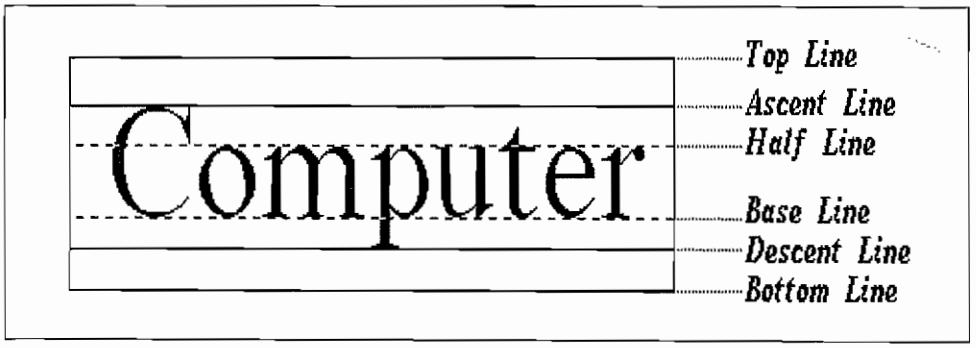


Abb. 3.8: Die unterschiedlichen Textausrichtungen

Deklaration in C:

```
void vst_alignment (WORD handle, WORD hor_in, WORD vert_in,
                    WORD *hor_out, WORD *vert_out)
{
     intin[0] = hor_in;
     intin[1] = vert_in;
     contrl[0] = 39;
     contrl[1] = 0;
     contrl[3] = 2;
     contrl[6] = handle;
     vdi ();
     *hor_out = intout[0];
     *vert_out = intout[1];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	39	Opcode für VST_ALIGNMENT
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	2	# Einträge in intin
contrl+8	contrl[4]	2	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	hor_in	
intin+2	intin[1]	vert_in	
intout	intout[0]	hor_out	
intout+2	intout[1]	vert_out	

Parameter:

hor_in:	horizontale Ausrichtung	
	TA_LEFT (0):	linksjustiert (default)
	TA_CENTER (1):	zentriert
	TA_RIGHT (2):	rechtsjustiert
vert_in:	vertikale Ausrichtung	
	TA_BASELINE (0):	Basislinie (default)
	TA_HALF (1):	Halblinie (Oberkante Kleinbuchstaben)
	TA_ASCENT (2):	Zeichenoberkante
	TA_BOTTOM (3):	Zeichenzellenunterkante
	TA_DESCENT (4):	Zeichenunterkante
	TA_TOP (5):	Zeichenzellenoberkante
hor_out:	ausgewählte horizontale Ausrichtung	
vert_out:	ausgewählte vertikale Ausrichtung	

SET FILL INTERIOR INDEX (VDI 23)

Mit dieser Funktion wird der Fülltyp ausgewählt. Man hat die Wahl zwischen fünf Typen: leer (Hintergrundfarbe), voll (monochrom), Muster, schraffiert, frei definiert. Bei Übergabe eines ungültigen Typs wählt SET FILL INTERIOR INDEX den Typ "leer".

Deklaration in C:

```
WORD vsf_interior (WORD handle, WORD style)
{
     intin[0] = style;
     contrl[0] = 23;
     contrl[1] = 0;
     contrl[3] = 1;
     contrl[6] = handle;
     vdi ();
     return intout[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	23	Opcode für VSF_INTERIOR
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	1	# Einträge in intin
contrl+8	contrl[4]	1	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	style	
intout	intout[0]	Return-Wert	

Parameter:

style: Gewünschter Fülltyp
FIS_HOLLOW (0): leer
FIS_SOLID (1): voll (einfarbig)
FIS_PATTERN (2): Muster
FIS_HATCH (3): Schraffur
FIS_USER (4): frei definierbar

vsf_interior(): ausgewählter Fülltyp

SET FILL STYLE INDEX (VDI 24)

Mit dieser Funktion wird der Musterindex zum Fülltyp ausgewählt. Diese Funktion hat nur dann einen Sinn, wenn als Fülltyp nicht *leer*, *einfarbig* oder *frei definierbar* gewählt wurde. Nicht verfügbare Indizes werden von SET FILL STYLE INDEX durch den Musterindex 1 ersetzt. Fülltyp 1, gefolgt von einem beliebigen Musterindex, entspricht immer dem Fülltyp 2 mit Musterindex 8. Der Index 1 bei Mustern (Typ 2) ist immer das Muster mit der geringsten Intensität auf dem Ausgabegerät. Es ist immer monochrom. Hier die Auswahl der Füllmuster mit Fülltyp und Musterindex:

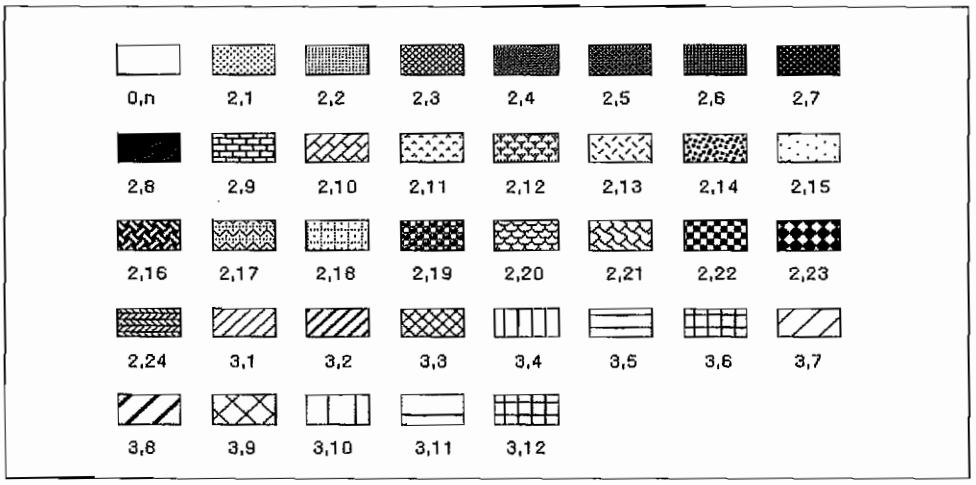


Abb. 3.9: Mögliche Kombinationen von Fülltyp und Musterindex

Deklaration in C:

```
WORD vsf_style (WORD handle, WORD style_index)
{
     intin[0] = style_index;
     contrl[0] = 24;
     contrl[1] = 0;
     contrl[3] = 1;
     contrl[6] = handle;
     vdi ();
     return intout[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	24	Opcode für VSF_STYLE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	1	# Einträge in intin
contrl+8	contrl[4]	1	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	style_index	
intout	intout[0]	Return-Wert	

Parameter:

style_index: gewünschter Musterindex (bei Mustern oder Schraffuren)
vsf_style(): ausgewählter Fülltyp (bei Mustern oder Schraffuren)

SET FILL COLOR INDEX (VDI 25)

Mit dieser Funktion wird die Füllfarbe gewählt. Die Farbindizes 0 und 1 sind immer verfügbar, weitere hängen von dem Ausgabegerät ab. Ein ungültiger Farbindex wird durch 1 ersetzt.

Deklaration in C:

```
WORD vsf_color (WORD handle, WORD color_index)
{
     intin[0] = color_index;
     contrl[0] = 25;
     contrl[1] = 0;
     contrl[3] = 1;
     contrl[6] = handle;
     vdi ();
     return intout[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	25	Opcode für VSF_COLOR
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	1	# Einträge in intin
contrl+8	contrl[4]	1	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	color_index	
intout	intout[0]	Return-Wert	

Parameter:

color_index: gewünschte Farbe
vsf_color(): ausgewählte Farbe

SET FILL PERIMETER VISIBILITY (VDI 104)

Diese Funktion schaltet die automatische Umrahmung der Füllfläche ein oder aus. Der Rand wird bei eingeschalteter Umrahmung (default) in der gegenwärtigen Füllfarbe (also im allgemeinen *nicht* in Schwarz!) als durchgehende Linie gezeichnet.

Deklaration in C:

```
WORD vsf_perimeter (WORD handle, WORD per_vis)
{
     intin[0] = per_vis;
     contrl[0] = 104;
     contrl[1] = 0;
     contrl[3] = 1;
     contrl[6] = handle;
     vdi ();
     return intout[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	104	Opcode für VSF_PERIMETER
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	1	# Einträge in intin
contrl+8	contrl[4]	1	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	per_vis	
intout	intout[0]	Return-Wert	

Parameter:

per_vis:	Flag für Umrahmung (0: keine Umrahmung)
vsf_perimeter():	eingestellter Modus

Bemerkungen
Wird von "vr_recfl()" *nicht* beachtet.

SET USER-DEFINED FILL PATTERN (VDI 112)

Mit dieser Funktion kann das freidefinierbare Füllmuster ("vsf_interior()", style = 4) festgelegt werden. 16 jeweils 16-Bit-Worte bestimmen das Füllmuster. Das höchste Bit (15) des ersten Wortes entspricht der oberen linken Ecke des Füllmusters, das niedrigste Bit (0) des letzten Wortes der unteren rechten Ecke. Für jede Farbebene wird ein eigenes Füllmuster angelegt. Bei einer einzelnen Ebene entspricht ein gesetztes Bit der durch die Füllfarbe bestimmten Farbe, der Vordergrundfarbe, ein nicht gesetztes Bit entspricht der Hintergrundfarbe. Bei Mustern, die aus mehreren Farbebenen bestehen, wird die volle Anzahl der Ebenen zum Füllen benutzt. Nicht aufgeführte Ebenen werden durch Nullebenen ersetzt. "Vielfarbige" Muster können nur im Schreibmodus REPLACE eingesetzt werden.

Deklaration in C:

```
void vsf_udpat (WORD handle, WORD *pfill_pat, WORD planes)
{
     iioff = pfill_pat;
     contrl[0] = 112;
     contrl[1] = 0;
     contrl[3] = planes * 16;
     contrl[6] = handle;
     vdi ();
     iioff = intin;
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	112	Opcode für VSF_UDPAT
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	16*n	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	pfill_pat[0]	
.	.	.	
.	.	.	
.	.	.	
intin+32*n-2	intin[16*n-1]	pfill_pat[16*n-1]	

Parameter:

planes: Anzahl der Farbebenen
pfill_pat[0] bis
pfill_pat[15]: Erste Ebene des Füllmusters (Defaultwert ist das Atari- bzw. das DRI-Logo)
pfill_pat[16] bis
pfill_pat[31]: Zweite Ebene des Füllmusters
.
.
.
pfill_pat[16*n-16] bis
pfill_pat[16*n-1]: (n-te) Ebene des Füllmusters

Bemerkungen
Aufgrund eines Tippfehlers in der ursprünglichen Originaldokumentation auch häufig "vsf_updat()" genannt.

Rasteroperationen

COPY RASTER, OPAQUE (VDI 109)

Diese Funktion kopiert (pixelweise) ein rechteckiges Raster unter Beachtung der möglichen logischen Verknüpfungen auf ein anderes rechteckiges Raster. Sollten die Größen beider Raster nicht übereinstimmen, so wird die Größe des Quellrasters benutzt. Die Adresse des Zielrasters dient in diesem Fall lediglich als Zeiger. Ist die Adresse der Quelle und des Ziels gleich (und insbesondere ungleich 0 im MFDB), und überlappen die beiden Bereiche, so wird das Quellrechteck nicht verändert, solange nicht das Zielrechteck fertig kopiert ist.

Raster im Standardformat *können* und *dürfen* nicht kopiert werden, da man im allgemeinen keine Informationen über das gerätespezifische Format hat. Also bei Bedarf vorher "TRANSFORM FORM" benutzen!

Deklaration in C:

```
void vro_cpyfm (WORD handle, WORD wr_mode, WORD *pxyarray,
                MFDB *psrcMFDB, MFDB *pdesMFDB)
{
     intin[0] = wr_mode;
     i_ptr (psrcMFDB);  /* contrl[7/8]=psrcMFDB */
     i_ptr2 (pdesMFDB); /* contrl[9/10]=pdesMFDP */
     pioff = pxyarray;
     contrl[0] = 109;
     contrl[1] = 4;
     contrl[3] = 1;
     contrl[6] = handle;
     vdi () ;
     pioff = ptsin;
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	109	Opcode für VRO_CPYFM
contrl+2	contrl[1]	4	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	1	# Einträge in intin

Adresse	Feldelemente	Belegung
contrl+8	contrl[4]	0 # Einträge in intout
contrl+12	contrl[6]	handle Gerätekennung
contrl+14	contrl[7/8]	psrcMDFB
contrl+18	contrl[9/10]	pdesMFDB
intin	intin[0]	wr_mode
ptsin	ptsin[0..7]	pxyarray[0..7]

Parameter:

psrcMFDB: Zeiger auf MFDB des Quellrasters
pdesMFDB: Zeiger auf MFDB des Zielrasters
wr_mode: Eine der 16 möglichen logischen Verknüpfungen (nicht zu verwechseln mit den Schreibmodi!)
pxyarray[0]: X-Koordinate des Eckpunktes des Quellrasters in NDC/RC-Koordinaten
pxyarray[1]: Y-Koordinate des Eckpunktes des Quellrasters in NDC/RC-Koordinaten
pxyarray[2]: X-Koordinate der diagonal gegenüberliegenden Ecke des Quellrasters in NDC/RC-Koordinaten
pxyarray[3]: Y-Koordinate der diagonal gegenüberliegenden Ecke des Quellrasters in NDC/RC-Koordinaten
pxyarray[4]: X-Koordinate des Eckpunktes des Zielrasters in NDC/RC-Koordinaten
pxyarray[5]: Y-Koordinate des Eckpunktes des Zielrasters in NDC/RC-Koordinaten
pxyarray[6]: X-Koordinate der diagonal gegenüberliegenden Ecke des Zielrasters in NDC/RC-Koordinaten
pxyarray[7]: Y-Koordinate der diagonal gegenüberliegenden Ecke des Zielrasters in NDC/RC-Koordinaten

Bemerkungen:
Ein MFDB hat folgende Struktur:

```
typedef struct
{
     void *fd_addr;       /* Zeiger auf Speicherblock. Bei Übergabe
                             eines Nullzeigers werden automatisch
                             die Parameter für das physikalische Ge-
                             rät (z.B. Bildschirmspeicher im geräte-
                             spezifischen Format) gesetzt. */
     WORD fd_w;           /* Speicherblockbreite in Punkten */
```

```
    WORD fd_h;          /* Höhe des Speicherblocks in Punkten */
    WORD fd_wdwidth;    /* Breite des Speicherblocks in Words */
    WORD fd_stand;      /* 0: geräteabhängiges Format,
                           1: Standardformat */
    WORD fd_nplanes;    /* Anzahl der Bildebenen */
    WORD fd_r1,fd_r2, fd_r3;        /* reserviert */
} MFDB;
```

Um es noch einmal ausdrücklich hervorzuheben: Um den Bildschirm anzusprechen, muß ein Nullzeiger bei der Speicherblockadresse (und nicht einer irgendwie ermittelten Adresse) eingetragen werden. Die Parameter für den Bildschirm werden (intern) automatisch eingetragen, was jedoch nicht bedeutet, daß man nach einem Aufruf einer Rasterkopierfunktion die korrekten Werte in der MFDB-Struktur zurückerhält. Bei dem Nullzeiger ist darauf zu achten, daß es sich wirklich um einen Nullzeiger und nicht etwa um NIL (=−1), wie bei einigen Modula-2-Bibliotheken, handelt.

Die logischen Verknüpfungen (Q = Pixelwert des Quellpixels, Z = Pixelwert des Zielpixels, E = Pixelwert des Zielpixels nach Verknüpfung):

Bezeichnung	**Wirkung**
ALL_WHITE (0)	E=0
S_AND_D (1)	E=Q and Z
S_AND_NOTD (2)	E=Q and (not Z)
S_ONLY (3)	E=Q
NOTS_AND_D (4)	E=(not Q) and Z
D_ONLY (5)	E=Z
S_XOR_D (6)	E=Q xor Z
S_OR_D (7)	E=Q or Z
NOT_SORD (8)	E=not (Q or Z)
NOT_SXORD (9)	E=not (Q xor Z)
D_INVERT (10)	E=not Z
NOT_D (11)	E=Q or (not Z)
S_OR_NOTD (12)	E=not Q
NOTS_OR_D (13)	E=(not Q) or Z)
NOT_SANDD (14)	E=not (Q and Z)
ALL_BLACK (15)	E=1

COPY RASTER, TRANSPARENT (VDI 121)

Diese Funktion kopiert (pixelweise) ein monochromes, rechteckiges Raster unter Beachtung der Schreibmodi auf ein anderes (auch farbiges) rechteckiges Raster. Sollten die Größen beider Raster nicht übereinstimmen, so werden die Größe des Quellrasters und die obere linke Ecke des Zielrasters als Startpunkt benutzt. Als Quellraster darf niemals der Bildschirm angegeben werden.

Die Schreibmodi:

Modus 1: REPLACE
Ersetzt alle Pixel des Zielrechtecks, wobei der Vordergrundfarbindex allen Pixeln zugeordnet wird, in deren zugehörigem Quellraster eine 1 steht, und der Hintergrundfarbindex allen Pixeln zugeordnet wird, in deren Quellraster eine 0 steht.

Modus 2: TRANSPARENT
Hier werden alle Pixel des Zielrechtecks auf die Vordergrundfarbe gesetzt, bei denen im zugehörigen Quellraster eine 1 steht. Der Farbindex für die Hintergrundfarbe wird nicht benutzt.

Modus 3: XOR
Jede Farbebene wird logisch mit dem monochromen Quellraster X-oderiert, d. h. jedes Bit des Pixelwerts wird ebenenweise mit dem Wert des Quellrasters XORed. Damit ist insbesondere die Farbe, die sich aus dieser logischen Operation ergibt, nicht eindeutig definiert. Lediglich daß aus Weiß (alle Bits auf 0 gesetzt) Schwarz (alle Bits auf 1 gesetzt) wird (und umgekehrt), ist gewiß. Farbindizes werden nicht berücksichtigt.

Modus 4: REVERSE TRANSPARENT
Hier werden alle Pixel des Zielrechtecks auf die Hintergrundfarbe gesetzt, bei denen im zugehörigen Quellraster eine 0 steht. Der Farbindex für die Vordergrundfarbe wird nicht benutzt.

Deklaration in C:

```
void vrt_cpyfm (WORD handle, WORD wr_mode, WORD *pxyarray,
                MFDB *psrcMFDB, MFDB *pdesMFDB, WORD *color_index)
{
     intin[0] = wr_mode;
     intin[1] = *color_index++;
     intin[2] = *color_index;
```

```
    i_ptr (psrcMFDB); /* contrl[7/8]=psrcMFDB */
    i_ptr2 (pdesMFDB); /* contrl[9/10]=pdesMFDB */
    pioff = pxyarray;
    contrl[0] = 121;
    contrl[1] = 4;
    contrl[3] = 3;
    contrl[6] = handle;
    vdi ();
    pioff = ptsin;
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	121	Opcode für VRT_CPYFM
contrl+2	contrl[1]	4	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	3	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
contrl+14	contrl[7/8]	psrcMFDB	
contrl+18	contrl[9/10]	pdesMFDB	
intin	intin[0]	wr_mode	
intin+2	intin[1..2]	color_index[0..1]	
ptsin	ptsin[0..7]	pxyarray[0..7]	

Parameter:

psrcMFDB:	Zeiger auf MFDB des Quellrasters
pdesMFDB:	Zeiger auf MFDB des Zielrasters
wr_mode:	Schreibmodus (siehe oben)
color_index[0]:	Farbindex der als gesetzt zu betrachtenden Punkte (Vordergrund)
color_index[1]:	Farbindex der als nichtgesetzt zu betrachtenden Punkte (Hintergrund)
pxyarray[0]:	X-Koordinate des Eckpunktes des Quellrasters in NDC/RC-Koordinaten
pxyarray[1]:	Y-Koordinate des Eckpunktes des Quellrasters in NDC/RC-Koordinaten
pxyarray[2]:	X-Koordinate der diagonal gegenüberliegenden Ecke des Quellrasters in NDC/RC-Koordinaten

pxyarray[3]: Y-Koordinate der diagonal gegenüberliegenden Ecke des Quellrasters in NDC/RC-Koordinaten
pxyarray[4]: X-Koordinate des Eckpunktes des Zielrasters in NDC/RC-Koordinaten
pxyarray[5]: Y-Koordinate des Eckpunktes des Zielrasters in NDC/RC-Koordinaten
pxyarray[6]: X-Koordinate der diagonal gegenüberliegenden Ecke des Zielrasters in NDC/RC-Koordinaten
pxyarray[7]: Y-Koordinate der diagonal gegenüberliegenden Ecke des Zielrasters in NDC/RC-Koordinaten

TRANSFORM FORM (VDI 110)

Diese Funktion transformiert ein Raster vom Standardformat in das gerätespezifische Format oder umgekehrt. Beim Standardformat handelt es sich um ein rechnerunabhängiges Datenformat, das mithin beim Austausch von Dateien zwischen verschiedenen GEM-Systemen benutzt werden kann (siehe COPY RASTER, OPAQUE).

Die Anzahl der Farbebenen im der Quell-MFDB-Struktur bestimmt die Anzahl der zu transformierenden Planes. Das Format-Flag wird für die Ziel-MFDB-Struktur entsprechend verändert. Für die korrekte Angabe der restlichen Werte in der Ziel-MFDB-Struktur ist man selbst verantwortlich. Je nach geräteabhängigem Format kann der Umwandlungsprozeß einfach oder sehr aufwendig sein. Man erleichtert dem VDI die Arbeit, wenn in Quell- und Ziel-MFDB unterschiedliche Rasteradressen angegeben sind – ansonsten kann eine Transformation schon mal mehrere Minuten dauern (die Umwandlung "in place" ohne Zuhilfenahme von temporärem Speicher ist eben eine schwierige Arbeit!).

Deklaration in C:

```
void vr_trnfm (WORD handle, MFDB *psrcMFDB, MFDB *pdesMFDB)
{
     i_ptr (psrcMFDB);
     i_ptr2 (pdesMFDB);
     contrl[0] = 110;
     contrl[1] = contrl[3] = 0;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	110	Opcode für VR_TRNFM
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
contrl+14	contrl[7/8]	psrcMFDB	
contrl+18	contrl[9/10]	pdesMFDB	

Parameter:

psrcMFDB: Zeiger auf MFDB des Quellrasters
pdesMFDB: Zeiger auf MFDB des Zielrasters

GET PIXEL (VDI 105)

Diese Funktion ermittelt den Pixelwert und den Farbindex eines Pixels.

Der Farbindex ist die Farbnummer, wie man sie gegenüber dem VDI bei den Attributfunktionen angeben kann (zum Beispiel bei "vst_color()"). Der Pixelwert hingegen spiegelt den tatsächlichen Inhalt des Bildspeichers wider.

Deklaration in C:

```
void v_get_pixel (WORD handle, WORD x, WORD y, WORD *pel,
                  WORD *index)
{
     ptsin[0] = x;
     ptsin[1] = y;
     contrl[0] = 105;
     contrl[1] = 1;
     contrl[3] = 0;
     contrl[6] = handle;
     vdi ();
     *pel = intout[0];
     *index = intout[1];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	105	Opcode für V_GET_PIXEL
contrl+2	contrl[1]	1	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	2	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
ptsin	ptsin[0]	x	
ptsin+2	ptsin[1]	y	
intout	intout[0]	pel	
intout+2	intout[1]	index	

Parameter:

x,y: Koordinaten des Pixels
pel: Pixelwert
index: Farbindex des Pixels

Eingabefunktionen

SET INPUT MODE (VDI 33)

Diese Funktion setzt den Input-Modus auf REQUEST oder SAMPLE. Im REQUEST-Modus wartet das Eingabegerät auf eine Eingabe, im SAMPLE-Modus wird lediglich der Zustand oder die Position der Eingabeeinheit zurückgegeben.

Die Eingabefunktionen des VDI funktionieren nur auf der physikalischen Workstation, die im Normalfall von den AES benutzt wird.

Deklaration in C:

```
WORD vsin_mode (WORD handle, WORD dev_type, WORD mode)
{
     intin[0] = dev_type;
     intin[1] = mode;
     contrl[0] = 33;
     contrl[1] = 0;
     contrl[3] = 2;
     contrl[6] = handle;
     vdi ();
     return intout[0];   /* Vorsicht! Nicht alle Bindings liefern
                            einen Return-Wert */
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	33	Opcode für VSIN_MODE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	2	# Einträge in intin
contrl+8	contrl[4]	1	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	dev_type	
intin+2	intin[1]	mode	
intout	intout[0]	Return-Wert	

Parameter:

handle:	Nummer der physikalischen Workstation	
dev_type:	logische Eingabeeinheit	
	DEV_LOCATOR (1):	Positionseingabeeinheiten (Maus, Trackball, Joystick)
	DEV_VALUATOR (2):	Wertverändernde Eingabeeinheiten (Potentiometer, Cursor, im ROM-Treiber nicht implementiert)
	DEV_CHOICE (3):	auswählende Eingabeeinheiten (Funktionstasten)
	DEV_STRING (4):	Zeichenketteneingabeeinheiten (Tastatur)
mode:	Eingabemodus MODE_REQUEST (1), MODE_SAMPLE (2)	
vsin_mode():	ausgewählter Eingabemodus	

Bemerkungen

Man beachte, daß es für "INPUT LOCATOR", "INPUT VALUATOR", "INPUT CHOICE" und "INPUT STRING" für die beiden verschiedenen Eingabemodi zwar unterschiedliche Bindings und Funktionsnamen gibt, die Opcodes jedoch jeweils identisch sind!

INPUT LOCATOR, REQUEST MODE (VDI 28)

Mit dieser Funktion wird die Position des Grafikcursors gesetzt bzw. ermittelt. Das Ergebnis erhält der Benutzer allerdings erst, wenn eine Taste gedrückt wird. Der Grafikcursor wird auf jeden Fall an der angegebenen Position in der aktuellen Form auf dem Bildschirm sichtbar. Zu beachten ist unbedingt, daß jede beliebige Taste (auch Maustaste) gedrückt werden darf.

Üblich ist eine Bewegung des Grafikcursors in großen Schritten, wenn die Cursortasten gedrückt werden, bzw. in kleinen Schritten, wenn die Cursortasten in Verbindung mit <Shift> gedrückt werden. Auf dem Atari benötigt man jedoch zusätzlich die Alternate-Taste. Diese Funktion wird nicht von allen Gerätetreibern unterstützt.

Die Eingabefunktionen des VDI funktionieren nur auf der physikalischen Workstation, die im Normalfall von den AES benutzt wird.

Deklaration in C:

```
void vrq_locator (WORD handle, WORD x, WORD y,
                  WORD *xout, WORD *yout, WORD *term)
{
     ptsin[0] = x;
     ptsin[1] = y;
     contrl[0] = 28;
     contrl[1] = 1;
     contrl[3] = 0;
     contrl[6] = handle;
     vdi ();
     *xout = ptsout[0];
     *yout = ptsout[1];
     *term = intout[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	28	Opcode für VRQ_LOCATOR
contrl+2	contrl[1]	1	# Einträge in ptsin
contrl+4	contrl[2]	1	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin

Adresse	Feldelement	Belegung	
contrl+8	contrl[4]	1	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
ptsin	ptsin[0]	x	
ptsin+2	ptsin[1]	y	
intout	intout[0]	term	
ptsout	ptsout[0]	xout	
ptsout+2	ptsout[1]	yout	

Parameter:

handle: Nummer der physikalischen Workstation
x: X-Koordinate des Grafikcursors in NDC/RC-Koordinaten
y: Y-Koordinate des Grafikcursors in NDC/RC-Koordinaten
term: Im Low-Byte wird der Code der (Positions-) Abbruch-Taste eingetragen. Für die üblichen Tastaturtasten ist dies der ASCII-Code. Spezielle Tasten am Eingabegerät (also die Maustasten) erhalten Werte ab 32 aufwärts und können so nicht ohne weiteres von Tasten auf der Tastatur unterschieden werden.
xout: X-Koordinate des Grafikcursors bei Rückgabe in NDC/RC-Koordinaten
yout: Y-Koordinate des Grafikcursors bei Rückgabe in NDC/RC-Koordinaten

INPUT LOCATOR, SAMPLE MODE (VDI 28)

Mit dieser Funktion wird die Position des Grafikcursors gesetzt bzw. ermittelt. Der Grafikcursor wird nicht sichtbar. Um ihn sichtbar zu machen, muß SHOW CURSOR aufgerufen werden. Tastenbetätigungen oder Grafikcursorbewegungen werden nur dann gemeldet, wenn diese tatsächlich erfolgt sind.

Diese Funktion wird nicht von allen Gerätetreibern unterstützt.

Die Eingabefunktionen des VDI funktionieren nur auf der physikalischen Workstation, die im Normalfall von den AES benutzt wird.

Deklaration in C:

```
WORD vsm_locator (WORD handle, WORD x, WORD y,
                  WORD *xout, WORD *yout, WORD *term)
{
     ptsin[0] = x;
     ptsin[1] = y;
     contrl[0] = 28;
     contrl[1] = 1;
     contrl[3] = 0;
     contrl[6] = handle;
     vdi ();
     *xout = ptsout[0];
     *yout = ptsout[1];
     *term = intout[0];
     return (contrl[4]<<1)|contrl[2];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	28	Opcode für VSM_LOCATOR
contrl+2	contrl[1]	1	# Einträge in ptsin
contrl+4	contrl[2]	0 oder 1	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0 oder 1	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung

Adresse	Feldelement	Belegung
ptsin	ptsin[0]	x
ptsin+2	ptsin[1]	y
intout	intout[0]	term
ptsout	ptsout[0]	xout
ptsout+2	ptsout[1]	yout

Parameter:

handle: Nummer der physikalischen Workstation
status: Bit 0: Koordinaten verändert?
Bit 1: Taste gedrückt?
x: X-Koordinate des Grafikcursors in NDC/RC-Koordinaten
y: Y-Koordinate des Grafikcursors in NDC/RC-Koordinaten
term: Im Low-Byte wird der Code der (Positions-) Abbruch-Taste eingegeben. Spezielle Tasten am Eingabegerät (also die Maustasten) erhalten Werte ab 32 aufwärts und können so nicht ohne weiteres von Tasten auf der Tastatur unterschieden werden.
xout: X-Koordinate des Grafikcursors in NDC/RC-Koordinaten bei Rückgabe
yout: Y-Koordinate des Grafikcursors in NDC/RC-Koordinaten bei Rückgabe

INPUT VALUATOR, REQUEST MODE (VDI 29)

Mit dieser Funktion wird eine Wertveränderung ermittelt. Es werden immer Werte zwischen 1 und 100 (nach Tastendruck) zurückgegeben. Typische Tasten für Wertveränderung sind die zwei Cursorbewegungstasten (<↑> und <↓>). Für die Cursorbewegungstasten erhält man folgende Wertveränderungen (jeweils zum aktuellen Wert):

Cursorbewegung	Wert
<↑>	+ 10
<↓>	– 10
<↑> und <Shift>	+ 1
<↓> und <Shift>	– 1

Diese Funktion wird nicht von allen Gerätetreibern (auch nicht dem im ROM) unterstützt. Die Eingabefunktionen des VDI funktionieren nur auf der physikalischen Workstation, die im Normalfall von den AES benutzt wird.

Deklaration in C:

```
void vrq_valuator (WORD handle, WORD valuator_in,
                   WORD *valuator_out, WORD *terminator)
{
     intin[0] = valuator_in;
     contrl[0] = 29;
     contrl[1] = 0;
     contrl[3] = 1;
     contrl[6] = handle;
     vdi ();
     *valuator_out = intout[0];
     *terminator = intout[1];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	29	Opcode für VRQ_VALUATOR
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	1	# Einträge in intin

Adresse	Feldelemente	Belegung
contrl+8	contrl[4]	2 # Einträge in intout
contrl+12	contrl[6]	handle Gerätekennung
intin	intin[0]	valuator_in
intout	intout[0]	valuator_out
intout+2	intout[1]	terminator

Parameter:

handle: Nummer der physikalischen Workstation
valuator_in: initialisierender Wert
valuator_out: Ausgabewert
terminator: betätigte Taste

INPUT VALUATOR, SAMPLE MODE (VDI 29)

Mit dieser Funktion wird eine Wertveränderung (ähnlich der Funktion im REQUEST-Modus) ermittelt. Es werden Werte zwischen 1 und 100 zurückgegeben, falls ein entsprechendes Ereignis aufgetreten ist. Sonst wird kein Wert zurückgegeben.

Diese Funktion wird nicht von allen Gerätetreibern (auch nicht dem im ROM) unterstützt. Die Eingabefunktionen des VDI funktionieren nur auf der physikalischen Workstation, die im Normalfall von den AES benutzt wird.

Deklaration in C:

```
void vsm_valuator (WORD handle, WORD val_in, WORD *val_out,
                   WORD *term, WORD *status);
{
     intin[0] = val_in;
     contrl[0] = 29;
     contrl[1] = 0;
     contrl[3] = 1;
     contrl[6] = handle;
     vdi ();
     *val_out = intout[0];
     *term = intout[1];
     *status = contrl[4];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	29	Opcode für VSM_VALUATOR
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	1	# Einträge in intin
contrl+8	contrl[4]	status	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	val_in	
intout	intout[0]	val_out	
intout+2	intout[1]	term	

Parameter:

handle:	Nummer der physikalischen Workstation
status:	Auswertung
	0: keine Änderung
	1: Wert verändert
	2: Taste betätigt
val_in:	initialisierender Wert
val_out:	Ausgabewert
term:	ggf. betätigte Taste

INPUT CHOICE, REQUEST MODE (VDI 30)

Mit dieser Funktion wird die Betätigung einer Auswahltaste (abgewartet und) ermittelt. Als Auswahltasten dienen zum Beispiel die Funktionstasten.

Die Anzahl der Funktionstasten ist geräteabhängig. Sollte eine andere Taste gedrückt werden, so wird die Codenummer dieser Taste zurückgegeben.

Diese Funktion ist zwar im ROM implementiert, scheint aber in keiner Weise zu funktionieren. Da sie nicht unbedingt nötig ist, wird sie von einigen Gerätetreibern nicht unterstützt.

Die Eingabefunktionen des VDI funktionieren nur auf der physikalischen Workstation, die im Normalfall von den AES benutzt wird.

Deklaration in C:

```
void vrq_choice (WORD handle, WORD ch_in, WORD *ch_out)
{
     intin[0] = ch_in;
     contrl[0] = 30;
     contrl[1] = 0;
     contrl[3] = 1;
     contrl[6] = handle;
     vdi ();
     *ch_out = intout[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	30	Opcode für VRQ_CHOICE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	1	# Einträge in intin
contrl+8	contrl[4]	1	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	ch_in	
intout	intout[0]	ch_out	

Parameter:

handle:	Nummer der physikalischen Workstation
ch_in:	initialisierende Auswahltaste (ausgewählt wird aus dem Zahlenbereich von 1 bis a, wobei die Auswahltastenanzahl a geräteabhängig ist, Beispiel: a = 10, zehn (Funktions- oder) Auswahltasten)
ch_out:	ausgewählte Taste (oder Codenummer)

INPUT CHOICE, SAMPLE MODE (VDI 30)

Mit dieser Funktion wird die Nummer der zuletzt betätigten Auswahltaste ermittelt. Als Auswahltasten dienen zum Beispiel die Funktionstasten.

Die Anzahl der Funktionstasten ist geräteabhängig. Sollte eine andere Taste gedrückt worden sein, so wird die Codenummer dieser Taste zurückgegeben.

Diese Funktion ist zwar im ROM implementiert, scheint jedoch nicht richtig zu funktionieren. Ferner ist die Funktion nicht unbedingt nötig und wird daher von einigen Gerätetreibern nicht unterstützt.

Die Eingabefunktionen des VDI funktionieren nur auf der physikalischen Workstation, die im Normalfall von den AES benutzt wird.

Deklaration in C:

```
WORD vsm_choice (WORD handle, WORD *choice)
{
     contrl[0] = 30;
     contrl[1] = contrl[3] = 0;
     contrl[6] = handle;
     vdi ();
     *choice = intout[0];
     return contrl[4];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	30	Opcode für VSM_CHOICE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	status	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intout	intout[0]	choice	

Parameter:

handle: Nummer der physikalischen Workstation
status: Auswahlstatus
0: keine Taste betätigt
1: Taste betätigt
choice: betätigte Auswahltaste (oder Code-Nummer) oder 0, falls keine Taste gedrückt wurde.

INPUT STRING, REQUEST MODE (VDI 31)

Diese Funktion gibt einen String (abgeschlossen mit RETURN oder bei Erreichen des String-Endes) von der Tastatur zurück. Ein Echo auf dem Bildschirm kann erfolgen. Im Fall der Echoausgabe werden die Textattribute berücksichtigt. Die Echoausgabe ist nicht auf allen Geräten verfügbar.

Die Eingabefunktionen des VDI funktionieren nur auf der physikalischen Workstation, die im Normalfall von den AES benutzt wird.

Deklaration in C:

```
void vrq_string (WORD handle, WORD max_length, WORD echo_mode,
                 WORD *echo_xy, char *string);
{
     WORD tmp;
     intin[0] = max_length;
     intin[1] = echo_mode;
     pioff = echo_xy;
     contrl[0] = 31;
     contrl[1] = 1;
     contrl[3] = 2;
     contrl[6] = handle;
     vdi ();
     for (tmp = 0; tmp<contrl[4]; tmp++)
          *string++ = intout[tmp];
     *string = 0;
     pioff = ptsin;
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	31	Opcode für VRQ_STRING
contrl+2	contrl[1]	1	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	2	# Einträge in intin
contrl+8	contrl[4]	n	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung

Adresse	Feldelemente	Belegung
intin	intin[0]	max_length
intin+2	intin[1]	echo_mode
ptsin	ptsin[0..1]	echo_xy[0..1]
intout	intout[0..n–1]	string[0..n–1]

Parameter:

handle: Nummer der physikalischen Workstation
max_length: maximale Länge des Strings. Im Fall einer negativen Zahl wird der Absolut-Betrag der Zahl als Länge genommen (hier nicht mit ASCII-Zeichen, sondern mit den Codenummern der VDI-Standard-Tastatur, siehe Anhang).
echo_mode: Echo-Modus (0: aus, 1: Echo ein (also gleichzeitige Textausgabe auf dem Monitor))
echo_xy[0]: X-Koordinate für Startpunkt Echoausgabe in NDC/RC-Koordinaten
echo_xy[1]: Y-Koordinate für Startpunkt Echoausgabe in NDC/RC-Koordinaten
string: Zeiger auf Eingabepuffer (für ein C-Binding Null-terminiert, die Länge beinhaltet die 0.)

INPUT STRING, SAMPLE MODE (VDI 31)

Diese Funktion gibt einen String von der Tastatur zurück. Die Eingabe wird abgebrochen durch

- RETURN
- Erreichen der maximalen Eingabelänge
- nicht verfügbare Datenwerte.

Soll die Stringeingabe grundsätzlich mit RETURN abgeschlossen werden, so ist der REQUEST-Modus zu wählen.

Ein Echo auf dem Bildschirm kann erfolgen, falls für das Gerät verfügbar. Im Fall der Echoausgabe werden die Textattribute berücksichtigt.

Die Eingabefunktionen des VDI funktionieren nur auf der physikalischen Workstation, die im Normalfall von den AES benutzt wird.

Deklaration in C:

```
WORD vsm_string (WORD handle, WORD max_length, WORD echo_mode,
                 WORD *echo_xy, char *string)
{
     WORD tmp;

     intin[0] = max_length;
     intin[1] = echo_mode;
     pioff = echo_xy;
     contrl[0] = 31;
     contrl[1] = 1;
     contrl[3] = 2;
     contrl[6] = handle;
     vdi ();
     for(tmp = 0; tmp<contrl[4]; tmp++)
          *string++ = intout[tmp];
     *string = 0;
     pioff = ptsin;
     return contrl[4];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	31	Opcode für VSM_STRING
contrl+2	contrl[1]	1	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	2	# Einträge in intin
contrl+8	contrl[4]	status	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	max_length	
intin+2	intin[1]	echo_mode	
ptsin	ptsin[0..1]	echo_xy[0..1]	
intout	intout[0..n–1]	string[0..n–1]	

Parameter:

handle: Nummer der physikalischen Workstation

status: Eingabe durch ungültige Zeicheneingabe abgebrochen (0) oder Länge des Strings (>0)

max_length: maximale Länge des Strings. Im Falle einer negativen Zahl wird der Absolut-Betrag der Zahl als Länge genommen (hier nicht mit ASCII-Zeichen, sondern mit den Code-Nummern der VDI-Standard-Tastatur (siehe Anhang)).

echo_mode: Echo-Modus (0: aus, 1: Echo ein (gleichzeitige Ausgabe auf Bildschirm))

echo_xy[0]: X-Koordinate für Startpunkt Echoausgabe in NDC/RC-Koordinaten

echo_xy[1]: Y-Koordinate für Startpunkt Echoausgabe in NDC/RC-Koordinaten

string: Zeiger auf eingelesene Zeichenkette (auf ausreichende Länge achten), bei einem C-Binding nullterminiert. Die 0 zählt dann mit zur Länge.

SET MOUSE FORM (VDI 111)

Mit dieser Funktion kann die Form des Grafik-Cursors (Mauszeigers) frei definiert werden. Die Hintergrund- und Vordergrundmaske werden getrennt definiert.

Jede Maske wird als Feld mit 16 Wörtern zu je 16 Bit festgelegt. Hierbei ist Bit 15 des ersten Wortes die obere linke Ecke der Maske und Bit 0 des 16. Wortes die rechte untere Ecke.

Für beide Masken wird ebenso die Farbe getrennt angegeben. Als Hintergrundfarbe wird in der Regel 0 (weiß), als Vordergrundfarbe 1 (schwarz) gewählt.

Zusätzlich wird der Hot-Spot, die exakte Grafik-Cursor-Position, definiert. Bei dem Mauspfeil würde dies der Zeigerspitze, bei dem Kreuz der Mitte entsprechen. Die Koordinaten werden relativ zur oberen linken Ecke der Maske angegeben.

Die Ausgabe der Maus erfolgt folgendermaßen:

- Die Bilddaten unter dem Mauszeiger werden gerettet und bei einer Mausbewegung wiederhergestellt.
- Gesetzte Flächen der Hintergrundmaske werden in der Maskenfarbe ausgegeben.
- Gesetzte Flächen der Vordergrundfarbe werden in der Datenfarbe ausgegeben.

Die Eingabe-Funktionen des VDI funktionieren nur auf der physikalischen Workstation, die im Normalfall von den AES benutzt wird. Statt dessen sollte man "graf_mouse()" einsetzen!

Deklaration in C:

```
void vsc_form (WORD handle, WORD *pcur_form)
{
     iioff = pcur_form;
     contrl[0] = 111;
     contrl[1] = 0;
     contrl[3] = 37;
     contrl[6] = handle;
     vdi ();
     iioff = intin;
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	111	Opcode für VSC_FORM
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	37	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0..36]	pcur_form[0..36]	

Parameter:

handle: Nummer der physikalischen Workstation
pcur_form[0]: (relative) X-Koordinate des Hot-Spot
pcur_form[1]: (relative) Y-Koordinate des Hot-Spot
pcur_form[2]: 1 (für zukünftige Anwendung)
pcur_form[3]: Farbindex der Hintergrundmaske (üblich: 0) (Maske)
pcur_form[4]: Farbindex der Vordergrundmaske (üblich: 1) (Daten)
pcur_form[5] bis
pcur_form[20]: Hintergrundmasken-Definition
pcur_form[21] bis
pcur_form[36]: Vordergrundmasken-Definition

EXCHANGE TIMER INTERRUPT VECTOR (VDI 118)

Möchte man eine eigene Anwendung mit einem Timerinterrupt aufrufen, so kann er mit dieser Funktion auf diese Anwendung gelegt werden. Es ist die Aufgabe der Anwendung, die alte Interrupt-Adresse zu speichern und am Ende das Register wiederherzustellen. Die Eingabe-Funktionen des VDI funktionieren nur auf der physikalischen Workstation, die im Normalfall von den AES benutzt wird.

Deklaration in C:

```
void vex_timv (WORD handle, LONG tim_addr, LONG *otim_addr,
               WORD *tim_conv)
{
     i_ptr (tim_addr);   /* contrl[7/8]=tim_addr */
     contrl[0] = 118;
     contrl[1] = contrl[3] = 0;
     contrl[6] = handle;
     vdi ();
     m_lptr2 (otim_addr); /* *otim_addr=contrl[9/10] */
     *tim_conv = intout[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	118	Opcode für VEX_TIMV
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	1	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
contrl+14	contrl[7/8]	tim_addr	
contrl+18	contrl[9/10]	otim_addr	
intout	intout[0]	tim_conv	

Parameter:

handle:	Nummer der physikalischen Workstation
tim_addr:	Adresse der neuen Timerinterruptroutine
otim_addr:	Adresse der alten Timerinterruptroutine
tim_conv:	Interruptintervall in Millisekunden

SHOW CURSOR (VDI 122)

Diese Funktion zeigt den Grafik-Cursor, genauer: annulliert einen Aufruf der Hide-Cursor-Funktion. Erst mit Erscheinen des Grafik-Cursors auf dem Bildschirm kann dieser mit der Maus bewegt werden.

Zu beachten ist, daß gegebenenfalls für jedes Hide-Cursor-Kommando auch ein Show-Cursor-Kommando gegeben werden muß. Es besteht also die Möglichkeit, die Show- und Hide-Cursor-Kommandos ineinander zu verschachteln. Auf die saubere Verschachtelung muß selbst geachtet werden. Die Eingabe-Funktionen des VDI funktionieren nur auf der physikalischen Workstation, die im Normalfall von den AES benutzt wird. Statt dessen sollte man "graf_mouse()" benutzen.

Deklaration in C:

```
void v_show_c (WORD handle, WORD reset)
{
     intin[0] = reset;
     contrl[0] = 122;
     contrl[1] = 0;
     contrl[3] = 1;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	122	Opcode für V_SHOW_C
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	1	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	reset	

Parameter:

handle: Nummer der physikalischen Workstation

reset: Reset-Flag. 0: Anzahl der Hide-Cursor-Aufrufe wird ignoriert (Maus erscheint sofort wieder), sonst: normale Funktion (ein Hide-Cursor-Aufruf wird annulliert).

HIDE CURSOR (VDI 123)

Diese Funktion bewirkt das Ausschalten des Grafik-Cursors, womit der Benutzer der Anwendung keinen Einfluß auf diesen Cursor mehr hat.

Die Eingabe-Funktionen des VDI funktionieren nur auf der physikalischen Workstation, die im Normalfall von den AES benutzt wird. Statt dessen sollte man "graf_mouse()" benutzen.

Deklaration in C:

```
void v_hide_c (WORD handle)
{
     contrl[0] = 123;
     contrl[1] = contrl[3] = 0;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	123	Opcode für V_HIDE_C
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung

Parameter:

handle: Nummer der physikalischen Workstation

SAMPLE MOUSE BUTTON STATE (VDI 124)

Diese Funktion gibt eine Mausinformation, Position des Grafik-Cursors und Status der Maustasten, zurück.

Die Eingabe-Funktionen des VDI funktionieren nur auf der physikalischen Workstation, die im Normalfall von den AES benutzt wird.

Statt dessen sollte man "graf_mkstate()" oder die Event-Funktionen benutzen.

Deklaration in C:

```
void vq_mouse (WORD handle, WORD *pstatus, WORD *x, WORD *y)
{
     contrl[0] = 124;
     contrl[1] = 0;
     contrl[3] = 0;
     contrl[6] = handle;
     vdi ();
     *pstatus = intout[0];
     *x = ptsout[0];
     *y = ptsout[1];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	124	Opcode für VQ_MOUSE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	1	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	1	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intout	intout[0]	pstatus	
ptsout	ptsout[0]	x	
ptsout+2	ptsout[1]	y	

Parameter:

handle:	Nummer der physikalischen Workstation
pstatus:	Maustastenstatus (Bit 0: linke Taste, Bit 1: Taste rechts daneben, weitere Bits: weitere Tasten)
x:	X-Koordinate des Grafik-Cursors in NDC/RC-Koordinaten
y:	Y-Koordinate des Grafik-Cursors in NDC/RC-Koordinaten

EXCHANGE BUTTON CHANGE VECTOR (VDI 125)

Diese Funktion erlaubt den Aufruf einer Anwender-Routine durch Maustastendruck. Hierzu wird lediglich die Adresse der Maustasten-Status-Routine geändert. Das heißt, die Adresse der Routine, die vorher auf Maustastendruck reagierte, wird in eine anwendereigene Adresse umgewandelt.

Es ist die Aufgabe der Anwendung, die alte Maustasten-Status-Adresse zu speichern und am Ende das Register wiederherzustellen.

Die Eingabe-Funktionen des VDI funktionieren nur auf der physikalischen Workstation, die im Normalfall von den AES benutzt wird.

Deklaration in C:

```
void vex_butv (WORD handle, LONG pusrcode, LONG *psavcode)
{
     i_ptr (pusrcode); /* contrl[7/8]=pusrcode */
     contrl[0] = 125;
     contrl[1] = contrl[3] = 0;
     contrl[6] = handle;
     vdi ();
     m_lptr2 (psavcode); /* *psavcode=contrl[9/10] */
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	125	Opcode für VEX_BUTV
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
contrl+14	contrl[7/8]	pusrcode	
contrl+18	contrl[9/10]	psavcode	

Parameter:

handle: Nummer der physikalischen Workstation

pusrcode: Adresse der neuen Maustasten-Status-Routine. In Register D0 erhält die eigene Routine den Status der Maustasten. Das VDI bearbeitet D0 nach Beendigung der Routine weiter – man kann also beispielsweise ein Bit abfragen, bearbeiten und löschen und das andere unversehrt lassen.

psavcode: Adresse der alten Maustasten-Status-Routine

EXCHANGE MOUSE MOVEMENT VECTOR (VDI 126)

Diese Funktion erlaubt den Aufruf einer Anwender-Routine durch Mausbewegung. Hierzu wird lediglich die Adresse der Mausbewegungs-Routine geändert. Das heißt, die Adresse der Routine, die vorher auf Mausbewegungen reagierte, wird in eine anwendereigene Adresse umgewandelt. Die Verzweigung in eine anwendereigene Routine erfolgt nach Änderung der Grafik-Cursor-Position, aber vor Aktualisierung der neuen Position auf dem Bildschirm. Es ist die Aufgabe der Anwendung, die alte Mausbewegungs-Adresse zu speichern und am Ende das Register wiederherzustellen. Die Eingabe-Funktionen des VDI funktionieren nur auf der physikalischen Workstation, die im Normalfall von den AES benutzt wird.

Deklaration in C:

```
void vex_motv (WORD handle, LONG pusrcode, LONG *psavcode)
{
     i_ptr (pusrcode); /* contrl[7/8]=pusrcode */
     contrl[0] = 126;
     contrl[1] = contrl[3] = 0;
     contrl[6] = handle;
     vdi();
     m_lptr2 (psavcode); /* *psavcode=contrl[9/10] */
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	126	Opcode für VEX_MOTV
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
contrl+14	contrl[7/8]	pusrcode	
contrl+18	contrl[9/10]	psavcode	

Parameter:

handle: Nummer der physikalischen Workstation

pusrcode: Adresse der neuen Mausbewegungs-Routine (in D0 und D1 werden die aktuellen Koordinaten der Maus geliefert)

psavcode: Adresse der alten Mausbewegungs-Routine

EXCHANGE CURSOR CHANGE VECTOR (VDI 127)

Diese Funktion erlaubt den Aufruf einer Anwender-Routine durch Grafik-Cursor-Veränderung. Hierzu wird lediglich die Adresse der Grafik-Cursor-Zeichenroutine geändert.

Das heißt, die Adresse, die vorher auf eine Grafik-Cursor-Neuzeichenroutine zeigte, wird in eine anwendereigene Adresse umgewandelt.

Die Verzweigung in eine anwendereigene Routine erfolgt nach Neuzeichnen des Grafik-Cursors. Es ist die Aufgabe der Anwendung, die Adresse der alten Grafik-Cursor-Zeichenroutine zu speichern und am Ende das Register wiederherzustellen.

Die Eingabe-Funktionen des VDI funktionieren nur auf der physikalischen Workstation, die im Normalfall von den AES benutzt wird.

Deklaration in C:

```
void vex_curv (WORD handle, LONG pusrcode, LONG *psavcode)
{
     i_ptr (pusrcode); /* contrl[7/8]=pusrcode */
     contrl[0] = 127;
     contrl[1] = contrl[3] = 0;
     contrl[6] = handle;
     vdi ();
     m_lptr2 (psavcode); /* *psavcode=contrl[9/10] */
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	127	Opcode für VEX_CURV
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
contrl+14	contrl[7/8]	pusrcode	
contrl+18	contrl[9/10]	psavcode	

Parameter:

handle:	Nummer der physikalischen Workstation
pusrcode:	Adresse der neuen Neuzeichen-Routine (in D0 und D1 werden die aktuellen Koordinaten der Maus geliefert). Wenn die eigene Routine den Grafik-Cursor *nicht* neuzeichnet, sollte man am Ende in die bisherige Routine weiterverzweigen!
psavcode:	Adresse der alten Neuzeichen-Routine

SAMPLE KEYBOARD STATE INFORMATION (VDI 128)

Diese Funktion gibt eine Tastaturinformation, Status der Control-, der Alternate-, der rechten und linken Shifttaste, zurück. Die Eingabe-Funktionen des VDI funktionieren nur auf der physikalischen Workstation, die im Normalfall von den AES benutzt wird. Statt dessen sollte man "graf_mkstate()" oder die Event-Funktionen benutzen.

Deklaration in C:

```
void vq_key_s (WORD handle, WORD *pstatus)
{
     contrl[0] = 128;
     contrl[1] = contrl[3] = 0;
     contrl[6] = handle;
     vdi ();
     *pstatus = intout[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	128	Opcode für VQ_KEY_S
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	1	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intout	intout[0]	pstatus	

Parameter:

handle:	Nummer der physikalischen Workstation	
pstatus:	Tastaturstatus:	Bitbelegung:
	K_RSHIFT (0x0001):	rechte Shift-Taste
	K_LSHIFT (0x0002):	linke Shift-Taste
	K_CTRL (0x0004):	Control-Taste
	K_ALT (0x0008):	Alternate-Taste

Auskunftsfunktionen

EXTENDED INQUIRE FUNCTION (VDI 102)

Diese Funktion gibt entweder Auskunft über die mit "OPEN WORKSTATION" ("v_opnwk()") eingestellten Parameter oder eine erweiterte Auskunft. Die Beschreibung der Parameter bezüglich "OPEN WORKSTATION" ist unter dem entsprechenden Funktionsaufruf zu finden.

Deklaration in C:

```
void vq_extnd (WORD handle, WORD owflag, WORD *work_out)
{
     iioff = intin;
     pioff = ptsin;
     iooff = work_out;
     pooff = &(work_out[45]);
     intin[0]= owflag;
     contrl[0] = 102;
     contrl[1] = 0;
     contrl[3] = 1;
     contrl[6] = handle;
     vdi ();
     iioff = intout;
     pioff = ptsout;
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	102	Opcode für VQ_EXTND
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	6	# Einträge in ptsout
contrl+6	contrl[3]	1	# Einträge in intin
contrl+8	contrl[4]	45	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	owflag	
intout	intout[0..44]	work_out[0..44]	
ptsout	ptsout[0..11]	work_out[45..56]	

Parameter:

owflag: Informationstyp (0: V_OPNWK-Parameter, Standardparameter, 1: erweiterte Parameter)

work_out[0]: Bildschirmtyp
- 0: kein Bildschirm
- 1: getrennter Alpha- und Grafikkontroller und getrennte Bildschirme
- 2: getrennter Alpha- und Grafikkontroller mit gemeinsamem Bildschirm
- 3: gemeinsamer Alpha- und Grafikkontroller mit getrenntem Bildspeicher
- 4: gemeinsamer Alpha- und Grafikkontroller mit gemeinsamem Bildspeicher

work_out[1]: Anzahl der verfügbaren Hintergrundfarben

work_out[2]: Bit-Vektor der verfügbaren Texteffekte (wie bei "vst_effects()")

work_out[3]: Flag für Vergrößerungsraster (0: Vergrößern nicht möglich, 1: Vergrößern möglich)

work_out[4]: Anzahl der Farbebenen

work_out[5]: "Lookup-table"-Unterstützung (0: nicht möglich, 1: möglich)

work_out[6]: Anzahl der 16x16-Pixel-Raster-Operationen pro Sekunde

work_out[7]: CONTOUR-FILL Verfügbarkeit (0: nicht verfügbar, 1: verfügbar)

work_out[8]: Textrotation (0: nicht möglich, 1: 90 Grad-Drehungen, 2: beliebig)

work_out[9]: Anzahl der Schreibmodi

work_out[10]: höchster Grad der Eingabemodi (0: keine, 1: request, 2: sample)

work_out[11]: Textausrichtungsverfügbarkeit (0: nicht verfügbar, 1: verfügbar)

work_out[12]: Farbstiftwechsel am Ausgabegerät (0: nicht möglich, 1: möglich)

work_out[13]: Farbbandwechselmöglichkeit (0: nicht möglich, 1: farbige Zeilen, 2: farbige Zeilen und Rechtecke)

work_out[14]: maximale Anzahl von Koordinatenpaaren für Polyline, Polymarker und Filled Area (–1: unbegrenzt). TOS 3.01 liefert an dieser Stelle einen *falschen* Wert (512 statt 256). Abhilfe: TOS 3.05 einsetzen.

work_out[15]: maximale Größe des INTIN-Arrays (–1: unbegrenzt)

work_out[16]: Anzahl der Maustasten

work_out[17]: Linientypen für breite Linien (über 1 Pixel Breite) (0: nicht möglich, 1: möglich)

work_out[18]: Schreibmodi für breite Linien (0: nicht verfügbar, 1: verfügbar)

work_out[19]: Clipping aus (0) oder an (1). Nur auf PC-GEM ab Version 2.0!

work_out[20] bis work_out[44]: reserviert, enthält 0 als Ausgabewert.

work_out[45]: obere linke X-Koordinate des Clipping-Rechtecks in NDC/RC-Koordinaten

work_out[46]: obere linke Y-Koordinate des Clipping-Rechtecks in NDC/RC-Koordinaten
work_out[47]: untere linke X-Koordinate des Clipping-Rechtecks in NDC/RC-Koordinaten
work_out[48]: untere linke Y-Koordinate des Clipping-Rechtecks in NDC/RC-Koordinaten
work_out[49] bis
work_out[56]: reserviert, enthält 0 als Ausgabewert.

Bemerkungen
Zusätzliche Funktion für Matrixdrucker: Es kann die maximale Auflösung angegeben werden:

ptsin[0]: maximale X-Auflösung
ptsin[1]: maximale Y-Auflösung
contrl[1]: auf 1 setzen

INQUIRE COLOR REPRESENTATION (VDI 26)

Diese Funktion gibt über die eingestellten Farbintensitäten der Farbpalette Auskunft. Es besteht die Möglichkeit, die eingestellte oder die tatsächliche Intensitätsverteilung zu erfragen.

Deklaration in C:

```
WORD vq_color (WORD handle, WORD color_index, WORD set_flag,
               WORD *rgb)
{
     intin[0] = color_index;
     intin[1] = set_flag;
     contrl[0] = 26;
     contrl[1] = 0;
     contrl[3] = 2;
     contrl[6] = handle;
     vdi ();
     rgb[0] = intout[1];
     rgb[1] = intout[2];
     rgb[2] = intout[3];
     return intout[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	26	Opcode für VQ_COLOR
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	2	# Einträge in intin
contrl+8	contrl[4]	4	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	color_index	
intin+2	intin[1]	set_flag	
intout	intout[0]	Return-Wert	
intout+2	intout[1..3]	rgb[0..2]	

Parameter:

color_index:	Farbregister, über das Auskunft erwünscht ist
set_flag:	Flag für gesetzte oder tatsächliche Farbintensität
	0: gesetzte Intensität
	1: tatsächliche Intensität
rgb[0]:	Rotintensität (in Promille, 01000)
rgb[1]:	Grünintensität (in Promille, 0–1000)
rgb[2]:	Blauintensität (in Promille, 0–1000)
vq_color():	1, (gewünschter/gesetzter) Farbindex außerhalb der Grenzen

Bemerkungen

Gesetzte und tatsächliche Intensität können unterschiedlich sein. Die Intensitäten werden in Werten von 0 bis 1000 angegeben. Verfügt etwa ein Monitor nur über zwei Intensitäten, so teilt sich der Bereich in die Teilbereiche 0 bis 500 und 501 bis 1000. Die tatsächliche Intensität kann aber nur 0 oder 1000 sein. Egal welche Zahl man aus dem Bereich 0 bis 500 wählt, immer wird die Intensität 0 gesetzt. Also kann beispielsweise die gesetzte Intensität den Wert 47 und die tatsächliche den Wert 0 haben.

INQUIRE CURRENT POLYLINE ATTRIBUTES (VDI 35)

Diese Funktion gibt über die aktuellen gesetzten Linienattribute Auskunft.

Deklaration in C:

```
void vql_attributes (WORD handle, WORD *attrib)
{
     iooff = attrib;
     contrl[0] = 35;
     contrl[1] = contrl[3] = 0;
     contrl[6] = handle;
     vdi ();
     iooff = intout;
     attrib[3] = ptsout[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	35	Opcode für VQL_ATTRIBUTES
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	1	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	3	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intout	intout[0..2]	attrib[0..2]	
ptsout	ptsout[0]	attrib[5]	
ptsout+2	ptsout[1]	0	

Parameter:

attrib[0]: Linientyp
attrib[1]: Linienfarbindex
attrib[2]: Schreibmodus
attrib[3]: Linienbreite (bzgl. X-Achse in NDC/RC-Koordinaten)

Bemerkungen

Die hier gemachten Angaben gelten nur für die im ROM implementierte Funktion. Auch die gebräuchlichen Bindings gehen davon aus, obwohl die Beschreibung der Funktion vql_attributes in den Anleitungen manchmal etwas anderes behauptet. Nach Digital Research ist die Funktion jedoch ein bißchen anders implementiert (in Übereinstimmung beispielsweise mit den GDOS-Druckertreibern).

Demnach erhält man in intout[0] den Linientyp, in intout[1] den Linienfarbindex, in intout[2] den Schreibmodus, in intout[3] den Linien-Anfangsstil, in intout[4] den Linien-Endstil und in ptsout[0] die Liniendicke zurück. Man beachte daher nach Rückkehr vom Funktionsaufruf contrl[4]!

INQUIRE CURRENT POLYMARKER ATTRIBUTES (VDI 36)

Diese Funktion gibt über die aktuellen gesetzten Markerattribute Auskunft.

Deklaration in C:

```
void vqm_attributes (WORD handle, WORD *attrib)
{
     iooff = attrib;
     contrl[0] = 36;
     contrl[1] = contrl[3] = 0;
     contrl[6] = handle;
     vdi ();
     iooff = intout;
     attrib[3] = ptsout[1];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	36	Opcode für VQM_ATTRIBUTES
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	1	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	3	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intout	intout[0..2]	attrib[0..2]	
ptsout	ptsout[0]	0	
ptsout+2	ptsout[1]	attrib[3]	

Parameter:

attrib[0]: Markertyp
attrib[1]: Markerfarbindex
attrib[2]: Schreibmodus
attrib[3]: Markerhöhe (bzgl. der Y-Achse in NDC/RC-Koordinaten)

Bemerkungen
Die hier gemachten Angaben gelten nur für die im ROM implementierte Funktion. Auch die gebräuchlichen Bindings gehen davon aus. Nach Digital Research ist die Funktion jedoch ein bißchen anders implementiert (in Übereinstimmung beispielsweise mit den GDOS-Druckertreibern).

Demnach erhält man in intout[0] den Markertyp, in intout[1] den Markerfarbindex, in intout[2] den Schreibmodus, in ptsout[0] die Markerbreite (bzgl. der X-Achse) und in ptsout[1] die Markerhöhe zurück. Man beachte daher bei Rückkehr vom Funktionsaufruf auf ptsout[0], ob dort ein Wert ungleich 0 zurückgegeben wurde!

Es existieren Bindings, bei denen attrib[3] den Wert 0 hat und in attrib[4] die Markerhöhe zurückgeliefert wird!

INQUIRE CURRENT FILL AREA ATTRIBUTES (VDI 37)

Diese Funktion gibt über die aktuell gesetzten Füllattribute Auskunft.

Deklaration in C:

```
void vqf_attributes (WORD handle, WORD *attrib)
{
     iooff = attrib;
     contrl[0] = 37;
     contrl[1] = contrl[3] = 0;
     contrl[6] = handle;
     vdi ();
     iooff = intout;
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	37	Opcode für VQF_ATTRIBUTES
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	5	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intout	intout[0..4]	attrib[0..4]	

Parameter:

attrib[0]: Füllmuster ("Fill interior index")
attrib[1]: Füllfarbindex
attrib[2]: Musterindex ("Fill style index")
attrib[3]: Schreibmodus
attrib[4]: Umrahmungsstatus (=0: unsichtbar, <>0: sichtbar)

INQUIRE CURRENT GRAPHIC TEXT ATTRIBUTES (VDI 38)

Diese Funktion gibt über die aktuellen gesetzten Textattribute Auskunft.

Deklaration in C:

```
void vqt_attributes (WORD handle, WORD *attrib)
{
     iooff = attrib;
     pooff = &(attrib[6]);
     contrl[0] = 38;
     contrl[1] = contrl[3] = 0;
     contrl[6] = handle;
     vdi ();
     iooff = intout;
     pooff = ptsout;
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	38	Opcode für VQT_ATTRIBUTES
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	2	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	6	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intout	intout[0..5]	attrib[0..5]	
ptsout	ptsout[0..3]	attrib[6..9]	

Parameter:

attrib[0]: Zeichensatz
attrib[1]: Textfarbindex
attrib[2]: Textrotationsrichtung (in 1/10 Grad)
attrib[3]: horizontale Ausrichtung (0: linksjustiert, 1: zentriert, 2: rechtsjustiert)
attrib[4]: vertikale Ausrichtung (0: Baseline, 1: Half line, 2: Ascent line, 3: Bottom line, 4: Descent line, 5: Top line)

attrib[5]: Schreibmodus
attrib[6]: Zeichenbreite (bzgl. X-Achse in NDC/RC-Koordinaten)
attrib[7]: Zeichenhöhe (bzgl. Y-Achse in NDC/RC-Koordinaten)
attrib[8]: Zeichenzellenbreite (bzgl. X-Achse in NDC/RC-Koordinaten)
attrib[9]: Zeichenzellenhöhe (bzgl. Y-Achse in NDC/RC-Koordinaten)

INQUIRE TEXT EXTENT (VDI 116)

Diese Funktion berechnet die Ausmaße eines minimalen Rechteckes, das den übergebenen String enthält. Die Koordinaten der vier Eckpunkte werden relativ zu einem rechtwinkligen Koordinatensystem ausgegeben.

Der niedrigste Eckpunkt des Textrechteckes liegt auf der X-Achse, der am weitesten links liegende Eckpunkt auf der Y-Achse. Die Eckpunkte sind entgegen dem Uhrzeigersinn durchnumeriert. Der erste Punkt liegt – wenn der Text normal waagerecht steht – in der unteren linken Ecke des Textrechteckes. Für die Berechnung der Ausmaße des Textrechteckes werden alle gesetzten Attribute berücksichtigt.

Deklaration in C:

```
void vqt_extent (WORD handle, const char *string, WORD *extent)
{
     WORD *tmp;
     tmp = intin;
     while (*tmp++ = *string++);
     pooff = extent;
     contrl[0] = 116;
     contrl[1] = 0;
     contrl[3] = (int)(tmp-intin)-1);
     contrl[6] = handle;
     vdi ();
     pooff = ptsout;
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung
contrl	contrl[0]	116 Opcode für VQT_EXTENT
contrl+2	contrl[1]	0 # Einträge in ptsin
contrl+4	contrl[2]	4 # Einträge in ptsout
contrl+6	contrl[3]	n # Einträge in intin
contrl+8	contrl[4]	0 # Einträge in intout
contrl+12	contrl[6]	handle Gerätekennung
intin	intin[0..n–1]	string[0..n–1]
ptsout	ptsout[0..7]	extent[0..7]

Parameter:

string:	Zeiger auf die Zeichenkette (bei C-Bindings nullterminiert)
extent[0]:	relative X-Koordinate des 1. Punktes des Textrechtecks (bzgl. der X-Achse in NDC/RC-Koordinaten)
extent[1]:	relative Y-Koordinate des 1. Punktes des Textrechtecks (bzgl. der Y-Achse in NDC/RC-Koordinaten)
extent[2]:	relative X-Koordinate des 2. Punktes des Textrechtecks (bzgl. der X-Achse in NDC/RC-Koordinaten)
extent[3]:	relative Y-Koordinate des 2. Punktes des Textrechtecks (bzgl. der Y-Achse in NDC/RC-Koordinaten)
extent[4]:	relative X-Koordinate des 3. Punktes des Textrechtecks (bzgl. der X-Achse in NDC/RC-Koordinaten)
extent[5]:	relative Y-Koordinate des 3. Punktes des Textrechtecks (bzgl. der Y-Achse in NDC/RC-Koordinaten)
extent[6]:	relative X-Koordinate des 4. Punktes des Textrechtecks (bzgl. der X-Achse in NDC/RC-Koordinaten)
extent[7]:	relative Y-Koordinate des 4. Punktes des Textrechtecks (bzgl. der Y-Achse in NDC/RC-Koordinaten)

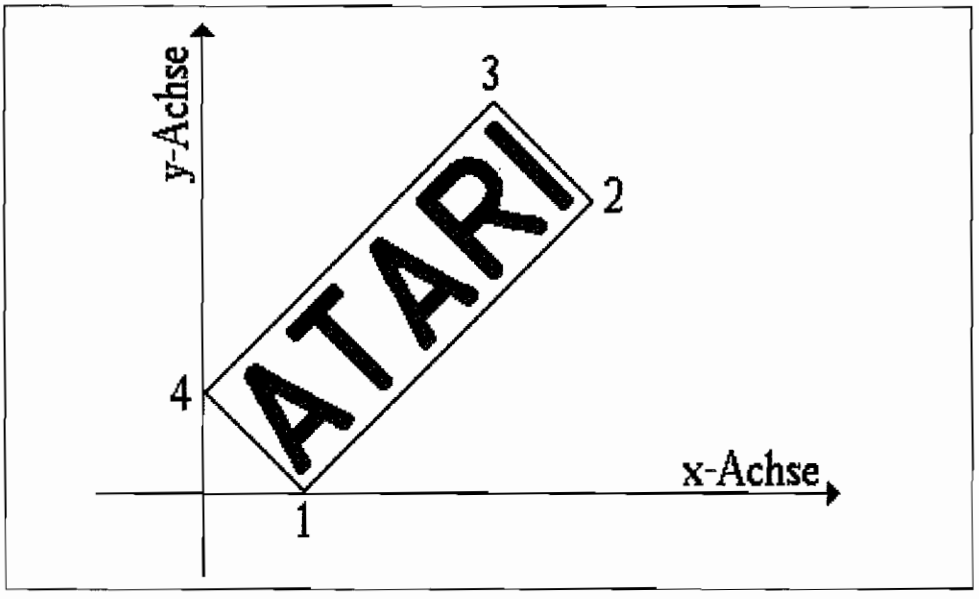


Abb. 3.10: Koordinatenpunkte bei vqt_attributes

INQUIRE CHARACTER CELL WIDTH (VDI 117)

Diese Funktion berechnet die horizontalen Ausmaße eines Zeichens des aktuellen Zeichensatzes und der Zeichenzelle. Die speziellen Texteffekte und Drehungen werden nicht berücksichtigt.

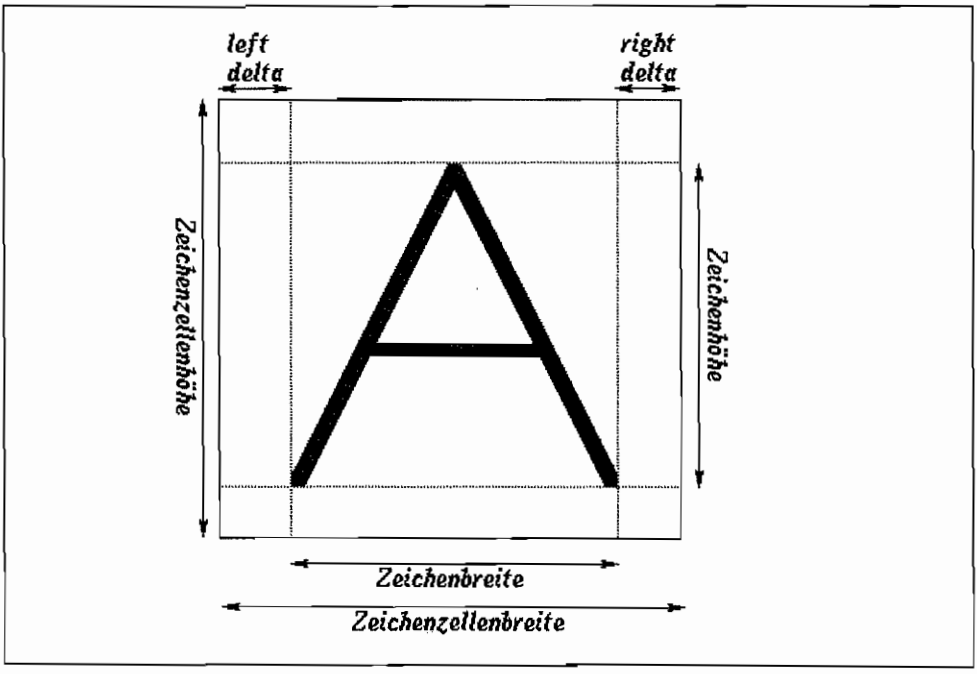


Abb. 3.11: Die Zeichenzelle

Deklaration in C:

```
WORD vqt_width (WORD handle, WORD character, WORD *cell_width,
                WORD *left_delta, *right_delta)
{
     intin[0] = character;
     contrl[0] = 117;
     contrl[1] = 0;
     contrl[3] = 1;
     contrl[6] = handle;
     vdi ();
```

```
    *cell_width = ptsout[0];
    *left_delta = ptsout[2];
    *right_delta = ptsout[4];
    return intout[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	117	Opcode für VQT_WIDTH
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	3	# Einträge in ptsout
contrl+6	contrl[3]	1	# Einträge in intin
contrl+8	contrl[4]	1	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	character	
intout	intout[0]	Return-Wert	
ptsout	ptsout[0]	cell_width	
ptsout+2	ptsout[1]	0	
ptsout+4	ptsout[2]	left_delta	
ptsout+6	ptsout[3]	0	
ptsout+8	ptsout[4]	right_delta	
ptsout+10	ptsout[5]	0	

Parameter:

character: Zeichennummer
vqt_width(): ASCII-Wert des nachgefragten Zeichens (–1 bei fehlerhafter Wertübergabe)
cell_width: Zeichenzellenbreite (bzgl. der X-Achse in NDC/RC-Koordinaten)
left_delta: Abstand linker Zeichenzellenrand zum linken Zeichenrand (bzgl. der X-Achse in NDC/RC-Koordinaten)
right_delta: Abstand rechter Zeichenzellenrand zum rechten Zeichenrand (bzgl. der Y-Achse in NDC/RC-Koordinaten)

Bemerkungen
Die Werte in "left_delta" und "right_delta" entsprechen exakt den entsprechenden Positionen im "Horizontal Offset Table". Fehlt diese Tabelle im Zeichensatzkopf (und das ist bei den meisten Zeichensätzen der Fall), erhält man als Ergebnis eine Null.

INQUIRE FACE NAME AND INDEX (VDI 130)

Diese Funktion gibt über einen Zeichensatznamen und dessen Indexnummer Auskunft. Sie ermöglicht es damit erst, bei anderen Funktionen den Zeichensatz über seine Indexnummer anzusprechen.

Deklaration in C:

```
WORD vqt_name (WORD handle, WORD element_num, char *name)
{
     WORD tmp;
     intin[0] = element_num;
     contrl[0] = 130;
     contrl[1] = 0;
     contrl[3] = 1;
     contrl[6] = handle;
     vdi ();
     for (tmp = 0; tmp<32; tmp++)
          name[tmp] = intout[tmp+1];
     return intout[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	130	Opcode für VQT_NAME
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	1	# Einträge in intin
contrl+8	contrl[4]	33	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	element_num	
intout	intout[0]	Return-Wert	
intout+2	intout[1..32]	name[0..31]	

Parameter:

element_num:	Nummer des Zeichensatzes (1 bis Maximalanzahl der verfügbaren Zeichensätze. Sie ergibt sich aus der Summe des Rückgabewertes beim Öffnen der Workstation (work_out[10]) und dem Rückgabewert von "vst_load_fonts()"))
vqt_name():	Index des Zeichensatzes
name:	Name des Zeichensatzes als maximal 32 Zeichen lange Zeichenkette (z. B. "Swiss 721").

INQUIRE CELL ARRAY (VDI 27)

Diese Funktion gibt Auskunft über die Definition des Cell-Arrays. Zu beachten ist, daß diese Funktion nicht auf allen Geräten verfügbar ist.

Deklaration in C:

```
void vq_cellarray (WORD handle, WORD *pxyarray, WORD row_length,
                   WORD num_rows, WORD *el_used, WORD *rows_used,
                   WORD *status, WORD *colarray)
{
     pioff = pxyarray;
     iioff = colarray;
     contrl[0] = 27;
     contrl[1] = 2;
     contrl[3] = 0;
     contrl[6] = handle;
     contrl[7] = row_length;
     contrl[8] = num_rows;
     vdi ();
     *el_used = contrl[9];
     *rows_used = contrl[10];
     *status = contrl[11];
     pioff = ptsin;
     iioff = intout;
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	27	Opcode für VQ_CELLARRAY
contrl+2	contrl[1]	2	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	n	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
contrl+14	contrl[7]	row_length	
contrl+16	contrl[8]	num_rows	
contrl+18	contrl[9]	el_used	

Adresse	Feldelement	Belegung
contrl+20	contrl[10]	rows_used
contrl+22	contrl[11]	status
ptsin	ptsin[0..3]	pxyarray[0..3]
intout	intout[0..n–1]	colarray[0..n–1]

Parameter:

row_length:	Zeilenlänge im Farbindex-Array
num_rows:	Anzahl der Zeilen im Farbindex-Array
el_used:	Spaltenanzahl
rows_used:	Anzahl der benutzten Spalten im Farbindex-Array
status:	0: keine Fehler, 1: für einige Pixel konnte der Farbwert nicht bestimmt werden.
pxyarray[0]:	X-Koordinate der oberen linken Ecke des Begrenzungsrechteckes
pxyarray[1]:	Y-Koordinate der oberen linken Ecke des Begrenzungsrechteckes
pxyarray[2]:	X-Koordinate der unteren rechten Ecke des Begrenzungsrechteckes
pxyarray[3]:	Y-Koordinate der unteren rechten Ecke des Begrenzungsrechteckes
colarray[]:	Farbindex-Array, enthält die Farbinformation zeilenweise, –1 falls Farbindex für den vorgegebenen Bereich nicht bestimmt werden konnte.

INQUIRE INPUT MODE (VDI 115)

Diese Funktion gibt Auskunft über den aktuellen Eingabemodus für das spezifizierte, logische Eingabegerät.

Deklaration in C:

```
void vqin_mode (WORD handle, WORD dev_type, WORD *input_mode)
{
     intin[0] = dev_type;
     contrl[0] = 115;
     contrl[1] = 0;
     contrl[3] = 1;
     contrl[6] = handle;
     vdi ();
     *input_mode = intout[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	115	Opcode für VQIN_MODE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	1	# Einträge in intin
contrl+8	contrl[4]	1	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	dev_type	
intout	intout[0]	input_mode	

Parameter:

handle: Kennung der physikalischen Workstation
dev_type: logische Eingabeeinheit
- 1: Positions-Eingabeeinheiten
- 2: wertverändernde Eingabeeinheiten
- 3: auswählende Eingabeeinheiten
- 4: Zeichenketten-Eingabeeinheiten

input_mode: Eingabe-Modus (1: REQUEST, 2: SAMPLE)

INQUIRE CURRENT FACE INFORMATION (VDI 131)

Diese Funktion gibt über den aktuellen Zeichensatz Auskunft. Es wird die gegenwärtige Größe ausgegeben. Die aktuellen Texteffekte werden ebenso berücksichtigt. Sollten die Zeichen in Kursivschrift dargestellt sein, so wird die Neigung als rechter und linker Offsetwert bedacht. Rechter Offsetwert ist der horizontale Abstand der Zeichenposition (linkes Basislinienende) zum Lotfußpunkt des Lotes auf die Basislinie durch die obere rechte Ecke. Linker Offsetwert ist der Abstand der Zeichenposition zum Lotfußpunkt des Lotes auf die Basislinie durch die untere linke Ecke.

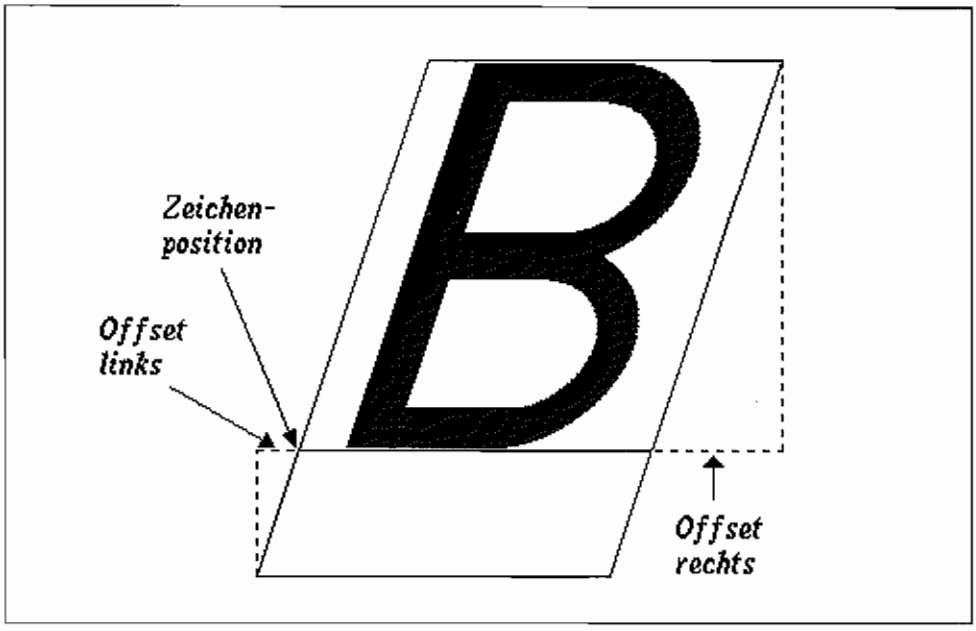


Abb. 3.12: Offsetinformationen über den aktuell eingestellten Zeichensatz

Deklaration in C:

```
void vqt_fontinfo (WORD handle, WORD *minADE, WORD *maxADE,
                   WORD *distances, WORD *maxwidth, WORD *effects)
{
     contrl[0] = 131;
     contrl[1] = contrl[3] = 0;
```

```
    contrl[6] = handle;
    vdi ();
    *minADE = intout[0];
    *maxADE = intout[1];
    *maxwidth = ptsout[0];
    distances[0] = ptsout[1];
    distances[1] = ptsout[3];
    distances[2] = ptsout[5];
    distances[3] = ptsout[7];
    distances[4] = ptsout[9];
    effects[0] = ptsout[2];
    effects[1] = ptsout[4];
    effects[2] = ptsout[6];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	131	Opcode für VQT_FONTINFO
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	5	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	2	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intout	intout[0]	minADE	
intout+2	intout[1]	maxADE	
ptsout	ptsout[0]	maxwidth	
ptsout+2	ptsout[1]	distances[0]	
ptsout+4	ptsout[2]	effects[0]	
ptsout+6	ptsout[3]	distances[1]	
ptsout+8	ptsout[4]	effects[1]	
ptsout+10	ptsout[5]	distances[2]	
ptsout+12	ptsout[6]	effects[2]	
ptsout+14	ptsout[7]	distances[3]	
ptsout+16	ptsout[8]	0	
ptsout+18	ptsout[9]	distances[4]	

Parameter:

minADE:	niedrigster ASCII-Wert, erstes Zeichen des Zeichensatzes
minADE:	höchster ASCII-Wert, letztes Zeichen des Zeichensatzes
maxwidth:	maximale Zeichenzellenbreite ohne Texteffekte
distances[0]:	Abstand Untergrenze Zeichenzelle zu Basislinie (bzgl. Y-Achse)
distances[1]:	Abstand Untergrenze Zeichen (Unterlängen) zur Basislinie (bzgl. Y-Achse)
distances[2]	Abstand Halblinie (Obergrenze Kleinbuchstaben (wie a, c, e, g,)) zur Basislinie (bzgl. Y-Achse)
distances[3]:	Abstand Obergrenze Zeichen zur Basislinie (bzgl. Y-Achse)
distances[4]:	Abstand Obergrenze Zeichenzelle zur Basislinie (bzgl. Y-Achse)
effects[0]:	Betrag, um den die Zeichenbreite bei den aktuell eingestellten Texteffekten zunimmt (bzgl. X-Achse)
effects[1]:	linker Offsetwert (bzgl. X-Achse)
effects[2]:	rechter Offsetwert (bzgl. X-Achse)

INQUIRE JUSTIFIED GRAPHICS TEXT (VDI 132) – nur auf PC-GEM ab Version 2.0!

Fragt für jedes Zeichen eines Strings die X- und Y-Offsets von einem gegebenen Ausrichtungspunkt ab.

Deklaration in C:

```
void vqt_justified (WORD handle, WORD x, WORD y,
                    const char *string, WORD length,
                    WORD word_space, WORD char_space,
                    WORD *offsets)
{
     WORD i;

     contrl[0] = 132;
     contrl[1] = 2;
     intin[0] = word_space;
     intin[1] = char_space;
     i = 0;
     while (tmp[i] = string[i++]) ;
     contrl[3] = i;
     contrl[6] = handle;
     ptsin[0] = x;
     ptsin[1] = y;
     ptsin[2] = length;
     ptsin[3] = 0;
     pooff = offsets;
     vdi ();
     pooff = ptsout;
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	132	Opcode für VQT_JUSTIFIED
contrl+2	contrl[1]	2	# Einträge in ptsin
contrl+4	contrl[2]	n	# Einträge in ptsout
contrl+6	contrl[3]	2+n	# Einträge in intin

Adresse	Feldelement	Belegung	
contrl+8	contrl[4]	0	# Einträge in intout
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	word_space	
intin+2	intin[1]	char_space	
intin+4	intin[2..n+1]	string[0..n+1]	
ptsin	ptsin[0]	x	
ptsin+2	ptsin[1]	y	
ptsin+4	ptsin[2]	length	
ptsin+6	ptsin[3]	0	
ptsout	ptsout[...]	offsets[...]	

Parameter:

word_space: Flag für Wortzwischenräume
0: keine Dehnung der Wortzwischenräume
nicht 0: Dehnung der Wortzwischenräume
char_space: Flag für Zeichenzwischenräume
0: keine Dehnung der Zeichenzwischenräume
nicht 0: Dehnung der Zeichenzwischenräume
string: Zeiger auf Zeichenkette
x: X-Koordinate des Textausrichtungspunktes
y: Y-Koordinate des Textausrichtungspunktes
length: Textlänge (in Pixeln) bzgl. der X-Richtung
offsets[]: jeweils für jedes Zeichen in der Zeichenkette der X- und Y-Offsets (das Array ist also genau doppelt so lang wie die Zeichenkette)

Escapes

ESCAPE (VDI 5)

Die Escape-Funktionen erlauben dem Anwendungsprogramm, spezielle Fähigkeiten von Ausgabegeräten auszunutzen. Das betreffende Ausgabegerät ist jeweils neben bzw. unter dem Funktionsnamen angegeben. Der Parameter "handle" ist bei allen Funktionen das Gerätehandle.

INQUIRE ADDRESSABLE ALPHA CHARACTER CELLS (VDI 5, Escape 1) (Bildschirm)

Diese Funktion gibt über die Anzahl der mit dem Alpha-Cursor ansprechbaren Zeilen und Spalten Auskunft.

Deklaration in C:

```
void vq_chcells (WORD handle, WORD *rows, WORD *columns)
{
     contrl[0] = 5;
     contrl[1] = contrl[3] = 0;
     contrl[5] = 1;
     contrl[6] = handle;
     vdi ();
     *rows = intout[0];
     *columns = intout[1];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	2	# Einträge in intout
contrl+10	contrl[5]	1	VQ_CHCELLS
contrl+12	contrl[6]	handle	Gerätekennung
intout	intout[0]	rows	
intout+2	intout[1]	columns	

Parameter:

rows: Anzahl der adressierbaren Zeilen, –1 falls keine Adressierung möglich
columns: Anzahl der adressierbaren Spalten, –1 falls keine Adressierung möglich

EXIT ALPHA MODE (VDI 5, Escape 2) (Bildschirm, Metafile)

Diese Funktion schaltet den Alpha-Modus aus, sofern Alpha- und Grafikmodus unterschiedlich sind. Gleichzeitig wird der Grafikbildschirm gelöscht (bei einem Metafile wird ein entsprechender Eintrag vorgenommen).

Deklaration in C:

```
void v_exit_cur (WORD handle)
{
     contrl[0] = 5;
     contrl[1] = contrl[3] = 0;
     contrl[5] = 2;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	2	V_EXIT_CUR
contrl+12	contrl[6]	handle	Gerätekennung

Bemerkungen

Der Alpha-Modus ist der normale Textmodus ohne Grafik, wie er etwa vom Desktop für die Anzeige von Dateien benutzt wird.

ENTER ALPHA MODE (VDI 5, Escape 3) (Bildschirm, Metafile)

Diese Funktion bewirkt das Verlassen des Grafik-Modus, falls dieser nicht identisch mit dem Alpha-Modus ist. Ebenso wird der Alpha-Cursor in die obere linke Zeichenzelle gesetzt und der Alpha-Bildschirm gelöscht (bei einem Metafile wird ein entsprechender Eintrag vorgenommen).

Deklaration in C:

```
void v_enter_cur (WORD handle)
{
     contrl[0] = 5;
     contrl[1] = contrl[3] = 0;
     contrl[5] = 3;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	3.	V_ENTER_CUR
contrl+12	contrl[6]	handle	Gerätekennung

ALPHA CURSOR UP (VDI 5, Escape 4) (Bildschirm)

Diese Funktion bewegt den Alpha-Cursor eine Zeile höher, ohne die horizontale Position zu verändern. Sollte sich der Cursor in der oberen Zeile befinden, so passiert nichts.

Deklaration in C:

```
void v_curup (WORD handle)
{
     contrl[0] = 5;
     contrl[1] = contrl[3] = 0;
     contrl[5] = 4;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	4	V_CURUP
contrl+12	contrl[6]	handle	Gerätekennung

ALPHA CURSOR DOWN (VDI 5, Escape 5) (Bildschirm)

Diese Funktion bewegt den Alpha-Cursor eine Zeile tiefer, ohne die horizontale Position zu verändern. Sollte sich der Cursor in der untersten Zeile befinden, so passiert nichts.

Deklaration in C:

```
void v_curdown (WORD handle)
{
     contrl[0] = contrl[5] = 5;
     contrl[1] = contrl[3] = 0;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	5	V_CURDOWN
contrl+12	contrl[6]	handle	Gerätekennung

ALPHA CURSOR RIGHT (VDI 5, Escape 6) (Bildschirm)

Diese Funktion bewegt den Alpha-Cursor eine Spalte nach rechts, ohne die vertikale Position zu verändern. Sollte sich der Cursor bereits am rechten Bildschirmrand befinden, so passiert nichts.

Deklaration in C:

```
void v_curright (WORD handle)
{
     contrl[0] = 5;
     contrl[1] = contrl[3] = 0;
     contrl[5] = 6;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	6	V_CURRIGHT
contrl+12	contrl[6]	handle	Gerätekennung

ALPHA CURSOR LEFT (VDI 5, Escape 7) (Bildschirm)

Diese Funktion bewegt den Alpha-Cursor eine Spalte nach links, ohne die vertikale Position zu verändern. Sollte sich der Cursor bereits am linken Rand befinden, so passiert nichts.

Deklaration in C:

```
void v_curleft (WORD handle)
{
     contrl[0] = 5;
     contrl[1] = contrl[3] = 0;
     contrl[5] = 7;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	7	V_CURLEFT
contrl+12	contrl[6]	handle	Gerätekennung

HOME ALPHA CURSOR (VDI 5, Escape 8) (Bildschirm)

Diese Funktion bewegt den Alpha-Cursor in seine "Heim"-Stellung, normalerweise oben links auf dem Bildschirm.

Deklaration in C:

```
void v_curhome (WORD handle)
{
     contrl[0] = 5;
     contrl[1] = contrl[3] = 0;
     contrl[5] = 8;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	8	V_CURHOME
contrl+12	contrl[6]	handle	Gerätekennung

ERASE TO END OF ALPHA SCREEN (VDI 5, Escape 9) (Bildschirm)

Diese Funktion löscht den Bildschirm ab der aktuellen Alpha-Cursor-Position, die hierbei nicht verändert wird.

Deklaration in C:

```
void v_eeos (WORD handle)
{
     contrl[0] = 5;
     contrl[1] = contrl[3] = 0;
     contrl[5] = 9;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	9	V_EEOS
contrl+12	contrl[6]	handle	Gerätekennung

ERASE TO END OF ALPHA TEXT LINE (VDI 5, Escape 10) (Bildschirm)

Diese Funktion löscht ab der aktuellen Alpha-Cursor-Position die aktuelle Textzeile, wobei die Position des Alpha-Cursors nicht verändert wird.

Deklaration in C:

```
void v_eeol (WORD handle)
{
     contrl[0] = 5;
     contrl[1] = contrl[3] = 0;
     contrl[5] = 10;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	10	V_EEOL
contrl+12	contrl[6]	handle	Gerätekennung

DIRECT ALPHA CURSOR ADDRESS (VDI 5, Escape 11) (Bildschirm)

Diese Funktion bewegt den Alpha-Cursor zu einer bestimmten Spalte und Zeile. Sollten Adressen oberhalb der Maximalgrenze angegeben werden, so nimmt die Funktion den am nächsten liegenden Wert.

Deklaration in C:

```
void v_curaddress (WORD handle, WORD row, WORD column)
{
     intin[0] = row;
     intin[1] = column;
     contrl[0] = 5;
     contrl[1] = 0;
     contrl[3] = 2;
     contrl[5] = 11;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	2	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	11	V_CURADDRESS
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	row	
intin+2	intin[1]	column	

Parameter:

row:	Zeile für Cursor-Position (1 bis Maximalanzahl der Zeilen)
column:	Spalte für Cursor-Position (1 bis Maximalanzahl der Spalten)

OUTPUT CURSOR ADDRESSABLE ALPHA TEXT (VDI 5, Escape 12) (Bildschirm)

Diese Funktion veranlaßt die Ausgabe einer Zeichenkette ab der aktuellen Alpha-Cursor-Position. Man beachte, daß man hiermit schneller (!) Text ausgeben kann als über das GEM-DOS! Der Cursor wird dabei für jedes Zeichen eine Stelle weitergerückt (der Alpha-Modus muß logischerweise angeschaltet sein).

Deklaration in C:

```
void v_curtext (WORD handle, const char *string)
{
     WORD *tmp;
     tmp = intin;
     while (*tmp++ = *string++);
     contrl[0] = 5;
     contrl[1] = 0;
     contrl[3] = ((int)(tmp-intin)-1);
     contrl[5] = 12;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	n	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	12	V_CURTEXT
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0..n–1]	string[0..n–1]	

Parameter:

string: Zeiger auf die auszugebende Zeichenkette

REVERSE VIDEO ON (VDI 5, Escape 13) (Bildschirm)

Mit dieser Funktion wird die inverse Alpha-Textdarstellung eingestellt.

Deklaration in C:

```
void v_rvon (WORD handle)
{
     contrl[0] = 5;
     contrl[1] = contrl[3] = 0;
     contrl[5] = 13;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	13	V_RVON
contrl+12	contrl[6]	handle	Gerätekennung

REVERSE VIDEO OFF (VDI 5, Escape 14) (Bildschirm)

Mit dieser Funktion wird die inverse Alpha-Textdarstellung abgeschaltet.

Deklaration in C:

```
void v_rvoff (WORD handle)
{
     contrl[0] = 5;
     contrl[1] = contrl[3] = 0;
     contrl[5] = 14;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	14	V_RVOFF
contrl+12	contrl[6]	handle	Gerätekennung

INQUIRE CURRENT ALPHA CURSOR ADDRESS (VDI 5, Escape 15) (Bildschirm)

Diese Funktion gibt über die aktuelle Alpha-Cursor-Position Auskunft.

Deklaration in C:

```
void vq_curaddress (WORD handle, WORD *row, WORD *column)
{
     contrl[0] = 5;
     contrl[1] = contrl[3] = 0;
     contrl[5] = 15;
     contrl[6] = handle;
     vdi ();
     *row = intout[0];
     *column = intout[1];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	2	# Einträge in intout
contrl+10	contrl[5]	15	VQ_CURADDRESS
contrl+12	contrl[6]	handle	Gerätekennung
intout	intout[0]	row	
intout+2	intout[1]	column	

Parameter:

row:	aktuelle Zeilenposition des Alpha-Cursors
column:	aktuelle Spaltenposition des Alpha-Cursors

INQUIRE TABLET STATUS (VDI 5, Escape 16) (Bildschirm)

Diese Funktion gibt über die Verfügbarkeit eines Grafik-Tabletts, einer Maus, eines Joysticks oder eines ähnlichen Gerätes Auskunft.

Deklaration in C:

```
WORD vq_tabstatus (WORD handle)
{
     contrl[0] = 5;
     contrl[1] = contrl[3] = 0;
     contrl[5] = 16;
     contrl[6] = handle;
     vdi ();
     return intout[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	1	# Einträge in intout
contrl+10	contrl[5]	16	VQ_TABSTATUS
contrl+12	contrl[6]	handle	Gerätekennung
intout	intout[0]	Return-Wert	

Parameter:

vq_tabstatus(): Gerätestatus (0: nicht verfügbar, 1: verfügbar)

HARD COPY (VDI 5, Escape 17) (Bildschirm)

Diese Funktion bewirkt die Ausgabe einer Hardcopy auf einen Drucker oder ein entsprechendes Gerät. Man beachte, daß dies eine Funktion des *Bildschirmtreibers* ist (der dazu auf das XBIOS zugreift).

Deklaration in C:

```
void v_hardcopy (WORD handle)
{
     contrl[0] = 5;
     contrl[1] = contrl[3] = 0;
     contrl[5] = 17;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	17	V_HARDCOPY
contrl+12	contrl[6]	handle	Gerätekennung

PLACE GRAPHIC CURSOR AT LOCATION (VDI 5, Escape 18) (Bildschirm)

Diese Funktion setzt den Grafik-Cursor an eine bestimmte Stelle. Sie ist nur auf Geräten verfügbar, die Positionseingaben (Locator) – etwa mit Maus, Joystick oder Trackball – zulassen.

Deklaration in C:

```
void v_dspcur (WORD handle, WORD x, WORD y)
{
     ptsin[0] = x;
     ptsin[1] = y;
     contrl[0] = 5;
     contrl[1] = 1;
     contrl[3] = 0;
     contrl[5] = 18;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	1	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	18	V_DSPCUR
contrl+12	contrl[6]	handle	Gerätekennung
ptsin	ptsin[0]	x	
ptsin+2	ptsin[1]	y	

Parameter:

x: X-Koordinate der neuen Grafik-Cursor-Stellung
y: Y-Koordinate der neuen Grafik-Cursor-Stellung

REMOVE LAST GRAPHIC CURSOR (VDI 5, Escape 19) (Bildschirm)

Diese Funktion entfernt den zuletzt dargestellten Grafik-Cursor.

Deklaration in C:

```
void v_rmcur (WORD handle)
{
     contrl[0] = 5;
     contrl[1] = contrl[3] = 0;
     contrl[5] = 19;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	19	V_RMCUR
contrl+12	contrl[6]	handle	Gerätekennung

FORM ADVANCE (VDI 5, Escape 20) (Drucker, Metafile)

"FORM ADVANCE" bewirkt einen Seitenvorschub (ähnlich "CLEAR WORKSTATION"). Der Datenpuffer wird nicht gelöscht. Falls das "Ausgabegerät" ein Metafile ist, wird ein entsprechender Eintrag geschrieben.

Deklaration in C:

```
void v_form_adv (WORD handle)
{
     contrl[0] = 5;
     contrl[1] = contrl[3] = 0;
     contrl[5] = 20;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	20	V_FORM_ADV
contrl+12	contrl[6]	handle	Gerätekennung

OUTPUT WINDOW (VDI 5, Escape 21) (Drucker)

"OUTPUT WINDOW" ist eine Verallgemeinerung von "UPDATE WORKSTATION", die es erlaubt, eine einzige Abbildung auf mehrere Blätter verteilt zu drucken. Einziger Unterschied zu "UPDATE WORKSTATION": Es kann ein Ausschnitt aus dem Drucker-Koordinatensystem ausgewählt werden.

Deklaration in C:

```
void v_output_window (handle, WORD *xyarray)
{
     pioff = xyarray;
     contrl[0] = 5;
     contrl[1] = 2;
     contrl[3] = 0;
     contrl[5] = 21;
     contrl[6] = handle;
     vdi ();
     pioff = ptsin;
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	2	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	21	V_OUTPUT_WINDOW
contrl+12	contrl[6]	handle	Gerätekennung
ptsin	ptsin[0..3]	xyarray[0..3]	

Parameter:

xyarray[0]: X-Koordinate einer Ecke des Ausgaberechteckes
xyarray[1]: Y-Koordinate einer Ecke des Ausgaberechteckes
xyarray[2]: X-Koordinate der diagonal gegenüberliegenden Ecke des Ausgaberechteckes
xyarray[3]: Y-Koordinate der diagonal gegenüberliegenden Ecke des Ausgaberechteckes

CLEAR DISPLAY LIST (VDI 5, Escape 22) (Drucker, Metafile)

Mit "CLEAR DISPLAY LIST" kann der Druckerpuffer (ähnlich "CLEAR WORKSTATION") gelöscht werden. Ein Seitenvorschub wird nicht durchgeführt.

Deklaration in C:

```
void v_clear_disp_list (WORD handle)
{
     contrl[0] = 5;
     contrl[1] = contrl[3] = 0;
     contrl[5] = 22;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	22	V_CLEAR_DISP_LIST
contrl+12	contrl[6]	handle	Gerätekennung

OUTPUT BIT IMAGE FILE (VDI 5, Escape 23) (Drucker, Metafile)

Mit "OUTPUT BIT IMAGE FILE" kann die in einer IMG-Datei gespeicherte Bild-Information gelesen und auf dem Ausgabegerät ausgegeben werden.

Deklaration in C:

```
void v_bit_image (WORD handle, const char *filename, WORD aspect,
                  WORD x_scale, WORD y_scale, WORD h_align,
                  WORD v_align, WORD *xyarray)
{
     WORD tmp;

     pioff = xyarray;
     intin[0] = aspect;
     intin[1] = x_scale;
     intin[2] = y_scale;
     intin[3] = h_align;
     intin[4] = v_align;
     tmp = 4;
     while (intin[tmp++] = *filename++);
     contrl[0] = 5;
     contrl[1] = 2;
     contrl[3] = -tmp;
     contrl[5] = 23;
     contrl[6] = handle;
     vdi ();
     pioff = ptsin;
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	2	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	n+5	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout

Adresse	Feldelement	Belegung
contrl+10	contrl[5]	23 V_BIT_IMAGE
contrl+12	contrl[6]	handle Gerätekennung
intin	intin[0]	aspect
intin+2	intin[1]	x_scale
intin+4	intin[2]	y_scale
intin+6	intin[3]	h_align
intin+8	intin[4]	v_align
intin+10	intin[5..n+4]	filename[0..n-1]
ptsin	ptsin[0..3]	xyarray[0..3]

Parameter:

aspect: Aspect Ratio (Abbildungsmaßstab)
0: Aspect Ratio wird ignoriert
1: Pixel werden berücksichtigt (z.B. werden Kreise wieder auf Kreise abgebildet)

x_scale: Skalierung der X-Achse (0: gebrochen (exakter), 1: ganzzahlig)

y_scale: Skalierung der Y-Achse (0: gebrochen (exakter), 1: ganzzahlig)

h_align: horizontale Ausrichtung (0: links, 1: zentriert, 2: rechts)

v_align: vertikale Ausrichtung (0: oben, 1: zentriert, 2: unten)

xyarray[0]: ggf. X-Koordinate der oberen linken Ecke des Ausgaberechteckes

xyarray[1]: ggf. Y-Koordinate der oberen linken Ecke des Ausgaberechteckes

xyarray[2]: ggf. X-Koordinate der unteren rechten Ecke des Ausgaberechteckes

xyarray[3]: ggf. Y-Koordinate der unteren rechten Ecke des Ausgaberechteckes

INQUIRE PRINTER SCAN (VDI 5, Escape 24) (Drucker)

Erlaubt die Abfrage verschiedener druckerspezifischer Parameter.

Deklaration in C:

```
void vq_scan (WORD handle, WORD *g_slice, WORD *g_page,
              WORD *a_slice, WORD *a_page, WORD *div_fac)
{
     contrl[0] = 5;
     contrl[1] = contrl[3] = 0;
     contrl[5] = 24;
     contrl[6] = handle;
     vdi ();
     *g_slice = intout[0];
     *g_page = intout[1];
     *a_slice = intout[2];
     *a_page = intout[3];
     *div_fac = intout[4];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	5	# Einträge in intout
contrl+10	contrl[5]	24	VQ_SCAN
contrl+12	contrl[6]	handle	Gerätekennung
intout	intout[0]	g_slice	
intout+2	intout[1]	g_page	
intout+4	intout[2]	a_slice	
intout+6	intout[3]	a_page	
intout+8	intout[4]	div_fac	

Parameter:

g_slice:	Der Druckertreiber unterteilt die Druckseite in mehrere "Scheiben" (engl.: slices), die nacheinander formatiert und gedruckt werden, um den Speicherplatzbedarf zu senken. "g_slice" ist die Anzahl dieser Scheiben.
g_page:	Pixelhöhe einer Scheibe
a_slice:	Höhe einer Textzeile in Pixeln
a_page:	Textzeilen pro Seite
div_fac:	Durch diesen Faktor müssen die anderen Werte ggf. noch geteilt werden.

OUTPUT ALPHA TEXT (VDI 5, Escape 25) (Drucker, Metafile)

Erlaubt die Ausgabe einfachen Textes (*nicht* im Grafikmodus) an der aktuellen Position des Druckkopfes. Grundlegende Steuerfunktionen sind für alle Druckertypen genormt.

Deklaration in C:

```
void v_alpha_text (WORD handle, const char *string)
{
     WORD *tmp;
     tmp = intin;
     while (*tmp++ = *string++);
     contrl[0] = 5;
     contrl[1] = 0;
     contrl[3] = ((int)(tmp-intin)-1);
     contrl[5] = 25;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	n	# Einträge in intin
contrl+8	contrl[4]	5	# Einträge in intout
contrl+10	contrl[5]	25	V_ALPHA_TEXT
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0..n–1]	string[0..n–1]	

Parameter:

string: auszugebender Text. Dabei sind folgende Steuerzeichen genormt:

DC2 0: Fett an
DC2 1: Fett aus
DC2 2: Italic an (Schrägschrift)

DC2 3:	Italic aus
DC2 4:	Unterstrichen an
DC2 5:	Unterstrichen aus
DC2 6:	Superscript an
DC2 7:	Superscript aus
DC2 8:	Subscript an
DC2 9:	Subscript aus
DC2 A:	Briefqualität-Modus an (LQ, NLQ)
DC2 B:	Briefqualität-Modus aus
DC2 C:	Breitschrift an
DC2 D:	Breitschrift aus
DC2 E:	Helle Schrift an
DC2 F:	Helle Schrift aus
DC2 G bis	
DC2 V:	reserviert, werden vom Treiber ignoriert
DC2 W:	Pica-Schrift (10 cpi)
DC2 X:	Elite-Schrift (12 cpi)
DC2 Y:	komprimierter Druck
DC2 Z:	Proportionalschrift

Ferner wird der Formfeed (Seitenvorschub) mit dem ASCII-Wert 12 unterstützt. Der String ist bei C-Binding nullterminiert.

Bemerkungen
Bei "DC2" handelt es sich um den ASCII-Wert 18.

Das OUT-Dateiformat (das von "OUTPUT" für PC-GEM gelesen wird) benutzt zusätzlich zu den genannten Control-Sequenzen das Kommando

```
(ESC) (ESC) GEM,x,y,w,h,D:\PATHNAME\FILENAME.EXT
```

um Grafiken in den Ausdruck einzubinden. Bei ESC handelt es sich um den ASCII-Wert 27. x, y, w und h sind in Zeicheneinheiten relativ zur aktuellen Cursorposition anzugeben.

SELECT PALETTE (VDI 5, Escape 60) (IBM-CGA)

Diese Funktion ermöglicht die Auswahl der Farbpalette auf einer IBM-CGA-Grafikkarte in mittlerer Auflösung.

Deklaration in C:

```
WORD vs_palette (WORD handle, WORD palette)
{
     contrl[0] = 5;
     contrl[1] = 0;
     contrl[3] = 1;
     contrl[5] = 60;
     contrl[6] = handle;
     intin[0] = palette;
     vdi ();
     return intout[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	1	# Einträge in intin
contrl+8	contrl[4]	1	# Einträge in intout
contrl+10	contrl[5]	60	VS_PALETTE
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	palette	
intout	intout[0]	Return-Wert	

Parameter:

palette: Farbpalettenwahl (0: Rot, Grün, Braun (default); 1: Cyan, Magenta, Weiß)
vs_palette(): ausgewählte Palette

Bemerkungen
Vermutlich die überflüssigste Funktionsbeschreibung des Profibuchs.

GENERATE SPECIFIED TONE (VDI 5, Escape 61) (Bildschirm) – nur in PC-GEM ab Version 2.0

Erzeugt einen Ton mit der angegebenen Länge und Tonhöhe.

Deklaration in C:

```
void v_sound (WORD handle, WORD frequency, WORD duration)
{
     contrl[0] = 5;
     contrl[1] = 0;
     contrl[3] = 2;
     contrl[5] = 61;
     contrl[6] = handle;
     intin[0] = frequency;
     intin[1] = duration;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	2	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	61	V_SOUND
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	frequency	
intin+2	intin[1]	duration	

Parameter:

frequency: Tonhöhe in Hertz
duration: Dauer in Timer-Ticks

Bemerkungen
Im ROM-Bildschirmtreiber nicht implementiert.

SET/CLEAR TONE MUTING FLAG (VDI 5, Escape 62) (Bildschirm) – nur in PC-GEM ab Version 2.0

Setzt oder löscht das Tonflag oder liefert dessen Status zurück.

Deklaration in C:

```
WORD vs_mute (WORD handle, WORD action)
{
     contrl[0] = 5;
     contrl[1] = 0;
     contrl[3] = 1;
     contrl[5] = 62;
     contrl[6] = handle;
     intin[0] = action;
     vdi ();
     return intout[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	1	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	62	VS_MUTE
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	action	
intout	intout[0]	Return-Wert	

Parameter:

action:	MUTE_RETURN (–1):	Status abfragen
	MUTE_ENABLE (0):	Tonerzeugung ein
	MUTE_DISABLE (1):	Tonerzeugung aus
vs_mute():	Tonerzeugung an (0) oder aus (ungleich 0)	

Bemerkungen

Im ROM-Bildschirmtreiber nicht implementiert.

SET TABLET AXIS RESOLUTION IN LINES/INCH (VDI 5, Escape 81) (Grafik-Tablett)

Setzt horizontale und vertikale Auflösung des Grafiktabletts.

Deklaration in C:

```
void vt_resolution (WORD handle, WORD xres, WORD yres,
                    WORD *xset, WORD *yset)
{
     contrl[0] = 5;
     contrl[1] = 0;
     contrl[3] = 2;
     contrl[5] = 81;
     contrl[6] = handle;
     intin[0] = xres;
     intin[1] = yres;
     vdi ();
     *xset = intout[0];
     *yset = intout[1];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	2	# Einträge in intin
contrl+8	contrl[4]	2	# Einträge in intout
contrl+10	contrl[5]	81	VT_RESOLUTION
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	xres	
intin+2	intin[1]	yres	
intout	intout[0]	xset	
intout+2	intout[1]	yset	

Parameter:

xres, yres:	Auflösung in Zeilen pro Zoll (lpi)
xset, yset:	gesetzte Auflösung

SET TABLET AXIS RESOLUTION IN LINES (VDI 5, Escape 82) (Grafik-Tablett)

Setzt horizontale und vertikale Auflösung des Grafiktabletts.

Deklaration in C:

```
void vt_axis (WORD handle, WORD xres, WORD yres, WORD *xset,
              WORD *yset)
{
     contrl[0] = 5;
     contrl[1] = 0;
     contrl[3] = 2;
     contrl[5] = 82;
     contrl[6] = handle;
     intin[0] = xres;
     intin[1] = yres;
     vdi ();
     *xset = intout[0];
     *yset = intout[1];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	2	# Einträge in intin
contrl+8	contrl[4]	2	# Einträge in intout
contrl+10	contrl[5]	82	VT_AXIS
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	xres	
intin+2	intin[1]	yres	
intout	intout[0]	xset	
intout+2	intout[1]	yset	

Parameter:

xres, yres:	Auflösung in Zeilen
xset, yset:	gesetzte Auflösung

SET TABLET X AND Y ORIGIN (VDI 5, Escape 83) (Grafik-Tablett)

Setzt den Ursprung des Koordinatensystems.

Deklaration in C:

```
void vt_origin (WORD handle, WORD xorigin, WORD yorigin)
{
     contrl[0] = 5;
     contrl[1] = 0;
     contrl[3] = 2;
     contrl[5] = 83;
     contrl[6] = handle;
     intin[0] = xorigin;
     intin[1] = yorigin;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	2	# Einträge in intin
contrl+8	contrl[4]	2	# Einträge in intout
contrl+10	contrl[5]	83	VT_ORIGIN
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	xorigin	
intin+2	intin[1]	yorigin	

Parameter:

xorigin: X-Koordinate der linken oberen Ecke
yorigin: Y-Koordinate der linken oberen Ecke

RETURN TABLET X AND Y DIMENSIONS (VDI 5, Escape 84) (Grafik-Tablett)

Liefert die Ausmaße des Grafiktabletts in 1/10-Zoll.

Deklaration in C:

```
void vq_tdimensions (WORD handle, WORD *xdimension,
                     WORD *ydimension)
{
     contrl[0] = 5;
     contrl[1] = contrl[3] = 0;
     contrl[5] = 84;
     contrl[6] = handle;
     vdi ();
     *xdimension = intin[0];
     *ydimension = intin[1];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	2	# Einträge in intout
contrl+10	contrl[5]	84	VQ_TDIMENSIONS
contrl+12	contrl[6]	handle	Gerätekennung
intout	intout[0]	xdimension	
intout+2	intout[1]	ydimension	

Parameter:

xdimension: Breite in 1/10 Zoll
ydimension: Höhe in 1/10 Zoll

SET TABLET ALIGNMENT (VDI 5, Escape 85) (Grafik-Tablett)

Dient der Ausrichtung des Koordinatensystems innerhalb eines Ausschnitts des Grafiktabletts.

Deklaration in C:

```
void vt_alignment (WORD handle, WORD dx, WORD dy)
{
     contrl[0] = 5;
     contrl[1] = 0;
     contrl[3] = 2;
     contrl[5] = 85;
     contrl[6] = handle;
     intin[0] = dx;
     intin[1] = dy;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	2	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	85	VT_ALIGNMENT
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	dx	
intin+2	intin[1]	dy	

Parameter:

dx, dy: Offset in X- und Y-Koordinate vom Ursprung.

SET CAMERA FILM TYPE AND EXPOSURE TIME (VDI 5, Escape 91) (Polaroid Palette)

Legt Filmtyp und Belichtungszeit fest.

Deklaration in C:

```
void vsp_film (WORD handle, WORD index, WORD lightness)
{
     contrl[0] = 5;
     contrl[1] = 0;
     contrl[3] = 2;
     contrl[5] = 91;
     contrl[6] = handle;
     intin[0] = index;
     intin[1] = lightness;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	2	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	91	VSP_FILM
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	index	
intin+2	intin[1]	lightness	

Parameter:

index: Nummer des Filmtyps (siehe "vqp_filmname()")

lightness: Belichtungszeit von –3 bis 3 (–3: halbe, 0: normal, 3: doppelte); mit jeder ganzen Zahl mehr wird die Blendenzahl um ein Drittel erhöht.

INQUIRE CAMERA FILM NAME (VDI 5, Escape 92) (Polaroid Palette)

Liefert zu einer Filmnummer den entsprechenden Namen (maximal 24 Zeichen). Bei einer falschen Filmnummer erhält man einen Nullstring als Ergebnis.

Deklaration in C:

```
WORD vqp_filmname (WORD handle, WORD index, char *name)
{
     WORD tmp;
     contrl[0] = 5;
     contrl[1] = 0;
     contrl[3] = 1;
     contrl[5] = 92;
     contrl[6] = handle;
     intin[0] = index;
     vdi ();
     for (tmp=0; tmp<contrl[4]; tmp++)
          name[tmp] = intout[tmp];
     return contrl[4];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	1	# Einträge in intin
contrl+8	contrl[4]	status	# Einträge in intout
contrl+10	contrl[5]	92	VQP_FILMNAME
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	index	
intout	intout[0..24]	name[0..24]	

Parameter:

index:	Nummer des Filmtyps
name:	Name des Films (nullterminiert)
vqp_filmname():	0: falsche Nummer übergeben

DISABLE OR ENABLE FILM EXPOSURE FOR FRAME PREVIEW (VDI 5, Escape 93) (Polaroid Palette)

Bei Kameratypen mit Preview-Möglichkeit kann man mit dieser Funktion die Belichtung abschalten.

Deklaration in C:

```
void vsc_expose (WORD handle, WORD state)
{
     intin[0] = state;
     contrl[0] = 5;
     contrl[1] = 0;
     contrl[3] = 1;
     contrl[5] = 93;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	1	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	93	VSC_EXPOSE
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	state	

Parameter:

state: 0: Belichtung abschalten
sonst: Belichtung einschalten

UPDATE METAFILE EXTENTS (VDI 5, Escape 98) (Metafiles)

Mit dieser Funktion werden die Größen-Informationen im Metafile-Kopf erneuert.

Die Größen-Information kann man benutzen, um schnell die minimalen oder maximalen Ausmaße aller im Metafile abgespeicherten Primitiven zu ermitteln.

Deklaration in C:

```
void v_meta_extents (WORD handle, WORD min_x, WORD min_y,
                     WORD max_x, WORD max_y)
{
     ptsin[0] = min_x;
     ptsin[1] = min_y;
     ptsin[2] = max_x;
     ptsin[3] = max_y;
     contrl[0] = 5;
     contrl[1] = 2;
     contrl[3] = 0;
     contrl[5] = 98;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	2	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	0	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	98	V_META_EXTENTS
contrl+12	contrl[6]	handle	Gerätekennung
ptsin	ptsin[0]	min_x	
ptsin+2	ptsin[1]	min_y	
ptsin+4	ptsin[2]	max_x	
ptsin+6	ptsin[3]	max_y	

Parameter:

min_x: minimaler X-Wert des minimalen umgebenden Rechteckes
min_y: minimaler Y-Wert des minimalen umgebenden Rechteckes
max_x: maximaler X-Wert des minimalen umgebenden Rechteckes
max_y: maximaler Y-Wert des minimalen umgebenden Rechteckes

WRITE METAFILE ITEM (VDI 5, Escape 99) (Metafiles)

Die mit dieser Funktion in ein Metafile geschriebenen Parameter werden mit einem Opcode als benutzerdefinierte Gegenstände abgespeichert. Intin[0] enthält einen benutzerdefinierten Sub-Opcode.

Die Nummern 0 bis 100 sind reserviert, als Sub-Opcode kommen also Nummern ab 101 in Frage.

Diese Funktion wird beispielsweise von "GEM Draw" (Digital Research) benutzt, um zu Gruppen zusammengefaßte Objekte zu markieren.

Deklaration in C:

```
void v_write_meta (WORD handle, WORD num_intin, WORD *a_intin,
                   WORD *num_ptsin, WORD *a_ptsin)
{
     iioff = a_intin;
     pioff = a_ptsin;
     contrl[0] = 5;
     contrl[1] = num_ptsin;
     contrl[3] = num_intin;
     contrl[5] = 99;
     contrl[6] = handle;
     vdi ();
     iioff = intin;
     pioff = ptsin;
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	num_ptsin	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	num_intin	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	99	V_WRITE_META
contrl+12	contrl[6]	handle	Gerätekennung

Parameter:

num_intin:	Anzahl der Werte im Intin-Array
num_ptsin:	Anzahl der Werte im Ptsin-Array
int_in[0]:	benutzerdefinierter Sub-Opcode
intin[1] bis intin[num_intin–1]:	benutzerdefinierte Information
ptsin[0] bis ptsin[num_ptsin–1]:	benutzerdefinierte Information

Bemerkungen
Von "GEM Draw" benutzte Opcodes:

START GROUP (10)	Beginn einer Gruppe
END GROUP (11)	Ende einer Gruppe
SET NO LINE STYLE (49)	Jeglichen Linienstil ausschalten
SET ATTRIBUTE SHADOW ON (50)	Alle bis "SHADOW OFF" folgenden Befehle werden benutzt, um für das erste Objekt danach einen Schatten zu zeichnen.
SET ATTRIBUTE SHADOW OFF (51)	Siehe oben
START DRAW AREA PRIMITIVE (80)	Beginn einer Füllfläche
END DRAW AREA PRIMITIVE (81)	Siehe oben

PHYSICAL PAGE SIZE (VDI 5, Escape 99, Opcode 0) (Metafiles)

Setzt die Seitengröße in 1/10 mm.

Deklaration in C:

```
void vm_pagesize (WORD handle, WORD pgwidth, WORD pgheight)
{
     contrl[0] = 5;
     contrl[1] = intin[0] = 0;
     contrl[3] = 3;
     contrl[5] = 99;
     contrl[6] = handle;
     intin[1] = pgwidth;
     intin[2] = pgheight;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	3	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	99	V_WRITE_META
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	0	VM_PAGESIZE
intin+2	intin[1]	pgwidth	
intin+4	intin[2]	pgheight	

Parameter:

pgwidth: Breite in 1/10 mm
pgheight: Länge in 1/10 mm

COORDINATE WINDOW (VDI 5, Escape 99, Opcode 1) (Metafiles)

Setzt das benutzte Koordinatensystem für die Seite.

Deklaration in C:

```
void vm_coords (WORD handle, WORD llx, WORD lly, WORD urx,
                WORD ury)
{
     contrl[0] = contrl[3] = 5;
     contrl[1] = 0;
     contrl[5] = 99;
     contrl[6] = handle;
     intin[0] = 1;
     intin[1] = llx;
     intin[2] = lly;
     intin[3] = urx;
     intin[4] = ury;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	5	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	99	V_WRITE_META
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	1	VM_COORDS
intin+2	intin[1]	llx	
intin+4	intin[2]	lly	
intin+6	intin[3]	urx	
intin+8	intin[4]	ury	

Parameter:

llx: X-Koordinaten links unten
lly: Y-Koordinaten links unten
urx: X-Koordinaten rechts oben
ury: Y-Koordinaten rechts oben

CHANGE GEM VDI FILE NAME (VDI 5, Escape 100) (Metafiles)

Diese Funktion benennt ein Metafile von "GEMFILE.GEM" in einen anderen Namen unter Beibehaltung der Extension "GEM" um. Ein Pfadname (mit Laufwerksbezeichnung) zur Auffindung des Files kann mit angegeben werden. Sollte "CHANGE GEM VDI FILE NAME" nicht sofort nach Aufruf der Funktion "OPEN WORKSTATION" (VDI 1) benutzt werden, so bleibt der Aufruf wirkungslos. Ein eventuell geöffnetes Metafile wird geschlossen.

Deklaration in C:

```
void vm_filename (WORD handle, const char *filename)
{
     WORD *tmp;
     tmp = intin;
     while (*tmp++ = *filename++);
     contrl[0] = 5;
     contrl[1] = 0;
     contrl[3] = ((int)(tmp-intin)-1);
     contrl[5] = 100;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	n	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	100	VM_FILENAME
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0..n–1]	filename[0..n–1]	

Parameter:

n: Länge des Intin-Arrays (1 bis 74)
filename: Dateispezifikation des Metafiles (muß mit dem ASCII-Wert 0 enden).

SET LINE OFFSET (VDI 5, Escape 101) (ROM-Bildschirmtreiber)

Setzt den Offset in Rasterzeilen zum Beginn des logischen Bildschirms (normalerweise 0). Diese Funktion ist in keiner offiziellen Dokumentation beschrieben – Benutzung auf eigene Gefahr!

Deklaration in C:

```
void v_offset (WORD handle, WORD offset)
{
     intin[0] = offset;
     contrl[0] = 5;
     contrl[1] = 0;
     contrl[3] = 1;
     contrl[5] = 101;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	1	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	101	V_OFFSET
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	offset	

Parameter:

offset: Anzahl der Rasterzeilen unterhalb der oberen Bildschirmkante, von der ab der logische Bildschirm beginnen soll.

INIT SYSTEM FONT (VDI 5, Escape 102) (ROM-Bildschirmtreiber)

Diese Routine installiert den angegebenen Zeichensatz als Systemzeichensatz. Eigene Versuche ergaben, daß die Zeichenbreite konstant acht Pixel betragen und der Zeichensatz im Motorola-Format vorliegen muß (siehe Font-Header)! Diese Funktion ist nirgendwo offiziell dokumentiert – daher Verwendung auf eigene Gefahr!

Deklaration in C:

```
void v_fontinit (WORD handle, WORD fh_high, WORD fh_low)
{
     intin[0] = fh_high;
     intin[1] = fh_low;
     contrl[0] = 5;
     contrl[1] = 0;
     contrl[3] = 2;
     contrl[5] = 102;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	2	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	102	V_FONTINIT
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0/1]	font_hdr	

Parameter:

font_hdr: Zeiger auf den Zeichensatzkopf.

ESCAPE 2000 (VDI 5, Escape 2000) (Atari SLM-Laser-Drucker)

Diese Spezialfunktion für die Atari-SLM-Laserdrucker druckt beliebig viele Kopien der laufenden Seite.

Deklaration in C:

```
void v_escape2000 (WORD handle, WORD times)
{
     intin[0] = times;
     contrl[0] = 5;
     contrl[1] = 0;
     contrl[3] = 1;
     contrl[5] = 2000;
     contrl[6] = handle;
     vdi ();
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	5	Opcode für ESCAPE
contrl+2	contrl[1]	0	# Einträge in ptsin
contrl+4	contrl[2]	0	# Einträge in ptsout
contrl+6	contrl[3]	1	# Einträge in intin
contrl+8	contrl[4]	0	# Einträge in intout
contrl+10	contrl[5]	2000	V_ESCAPE2000
contrl+12	contrl[6]	handle	Gerätekennung
intin	intin[0]	times	

Parameter:

times: Anzahl der zusätzlich zu druckenden Kopien

Kapitel 4: AES-Betriebssystemroutinen

Einleitung

Die AES ("Application Environment Services") sind die obere Schicht des GEM. Sie befassen sich mit all jenen GEM-Bestandteilen, die über elementare Grafikausgaben (und -eingaben) hinausgehen. Es werden ausschließlich VDI- und GEMDOS-Aufrufe benutzt, und so sind die AES sowohl von Grafikhardware (VDI), vom Eingabegerät (VDI) als auch vom Dateisystem (GEMDOS) unabhängig. Die Entwicklung der AES reicht in das Jahr 1984 zurück, als es von Digital Research zunächst für MS-DOS-Rechner entwickelt wurde und später auf die Apple Lisa (unter CP/M 68K) portiert wurde. Diese Version wurde dann anschließend auf den Atari ST (unter GEMDOS) portiert.

Die verschiedenen GEM-Versionen

GEM-Versionsnummern

Als GEM-Versionsnummer wird normalerweise die bei "appl_init()" vom AES (im "Global"-Feld) zurückgelieferte Kennung benutzt. Das VDI hingegen hat eigentlich keine eigene Versionsnummer, zumal das Verhalten der einzelnen VDI-Funktionen hauptsächlich von den benutzten Gerätetreibern (die ja weitestgehend austauschbar sind) bestimmt wird.

GEM 1.x

Diese erste AES-Version (1.x) hatte nicht von ungefähr sehr auffällige Ähnlichkeiten mit dem Betriebssystem des Apple Macintosh. Das äußerte sich nicht nur im Design der Fensterelemente, sondern auch in vielen Eigenschaften des Desktops und anderer Anwendungsprogramme. Damals wurde GEM meist im Zusammenhang mit Testversionen von "GEM-Draw", "GEM-Paint" und "GEM-Write" gezeigt, die in vielen Details den bekannten Mac-Vorbildern "MacDraw", "MacPaint" und "MacWrite" entsprachen.

Dies ist auch die GEM-Version, die schließlich von Atari übernommen und im ST ausgeliefert wurde. Nicht ganz fertige Testversionen der "Vorbildprogramme" werden noch bis heute benutzt. Auf dieser Fassung beruhen auch alle neueren Atari-GEM-Versionen. Atari hat damals nämlich alle Rechte an der bestehenden Version erworben und die Entwicklung selbst fortgeführt. Damit lassen sich auch die immer weiter "klaffenden" Unterschiede zwischen Atari-GEM und PC-GEM erklären.

GEM 2.x

Wie man sich denken kann ("nachher ist man immer schlauer") kam es zu einer juristischen Auseinandersetzung zwischen Apple und Digital Research, bei der es allerdings in erster Linie um das Outfit der Anwendungsprogramme und des Desktops ging.

Die Einigung, die Ataris GEM-Version nicht betraf, sah folgendermaßen aus:

- Einige Fensterelemente wurden so verändert, daß sie nicht mehr ganz so wie Mac-Fenster aussahen (in erster Linie der Titelbalken).

- Das Accessory-Menü wanderte in die gegenüberliegende Bildschirmecke.

- Die sogenannten "Grow-" und "Shrink-Boxes" wurden eliminiert.

- Das Desktop wurde völlig neu programmiert und auf zwei feste Fenster festgelegt. Entgegen landläufiger Meinung war das allerdings nur eine Änderung im Desktop, nicht in den AES (die weiterhin bis zu acht überlappende Fenster erlauben).

Doch es gab nicht nur Einschränkungen, sondern auch Verbesserungen. Als wichtiges Beispiel sei nur erwähnt, daß Accessories eigene Menüleisten anmelden dürfen.

Diese Version erhielt die Versionsnummer 2.0 und wird seit 1987 ausgeliefert. Nur wenig später wurde sie von dem niederländischen Softwarehaus "ABC" zusammen mit "GEM-Draw", "GEM-Paint", "GEM-Graph" und natürlich "GEM-Desktop" auf den Atari portiert. Leider erlangte diese Version nie große Bedeutung und ist heute nicht mehr erhältlich.

Alle für GEM 2.x spezifischen Informationen sind im folgenden mit einem Zusatz wie "nur in PC-GEM ab Version 2.0" versehen.

PC-GEM 3.x

In dieser Version sind nur noch geringfügige Verbesserungen vorgenommen worden (das Menüverhalten läßt sich von "Drop-Down" auf "Pull-Down" umschalten).

Mittlerweile wird PC-GEM nicht mehr von Digital Research selbst weiterentwickelt, und so sind angesichts der Dominanz von Microsoft Windows im PC-Bereich wohl keine größeren Neuerungen mehr zu erwarten. Teile von GEM und Desktop sind allerdings in die grafische Benutzeroberfläche "ViewMax" eingeflossen, die von Digital Research als Zubehör zu "DR-DOS 5.0" für PCs ausgeliefert wird.

Atari-GEM 1.4

Wichtige Änderungen hat Atari erst im GEM 1.4 im TOS 1.04 vorgenommen. Bekannteste Auswirkung ist die stark verbesserte Dateiauswahlbox.

Atari-GEM 3.x

Die Erweiterungen im Atari-GEM 3.x (verstellbare Farben und Muster für die Fensterelemente) hätten diese hohe Versionsnummer eigentlich nicht gerechtfertigt. Offenbar wollten die Programmierer absolut klar machen, daß Atari *nicht* gewillt ist, weiter zu PC-GEM kompatibel zu bleiben oder gar die dort vorgenommenen Änderungen nachzuvollziehen. Eine der Veränderungen im Atari-GEM ist bereits jetzt zum PC-GEM inkompatibel (neue Funktionsnummer bei "wind_set()").

Künftige Weiterentwicklungen

Es ist immer schwierig, in die Zukunft zu schauen. Doch bereits existierende Weiterentwicklungen wie X/GEM (ein erweitertes GEM für DR's Multitasking-System "Flex-OS") oder Multi-GEM (ein "Hack" für Atari-GEM von der Maxon GmbH) zeigen schon jetzt, daß das Marschziel dem sehr nahe kommen wird, was Mac-Benutzer als den "Multi-Finder" kennen: AES, die mehr als ein Hauptprogramm gleichzeitig ausführen können. Auch Ataris eigene Pläne gehen sicherlich in eben diese Richtung.

Wer neue GEM-Programme entwickelt, sollte diese möglichen Weiterentwicklungen immer im Kopf haben!

Die AES und Multitasking

Begriffserklärungen

Die aktuelle Version der AES erlaubt die gleichzeitige Abarbeitung von bis zu acht AES-Prozessen. Das klingt vielleicht zunächst etwas verblüffend, daher erst einmal ein paar Begriffserklärungen:

Auf einem Rechner mit nur einer CPU können natürlich niemals mehrere Programme wirklich gleichzeitig ausgeführt werden. Die Rechenzeit muß also möglichst geschickt auf die verschiedenen Aufgaben verteilt werden.

Bei "pre-emptivem" Multitasking wird jedem Prozeß ein bestimmtes Quantum Zeit zugestanden. Nach Ablauf dieser Frist wird dann automatisch auf den nächsten Prozeß weitergeschaltet. Dabei sind natürlich noch etliche Verfeinerungen wie unterschiedliche Prioritäten oder das "Blocken" von Funktionen (so könnte beispielsweise ein Prozeß so lange blockiert bleiben, bis ein Tastendruck gemeldet wird) möglich. Dieses Multitasking-Konzept wird zum Beispiel in allen Unix-Versionen und verwandten Betriebssystemen benutzt.

Beim "kooperativen" Multitasking wird ein etwas anderer Weg gegangen: Jeder Prozeß erhält quasi die volle Kontrolle über das Rechnersystem, die er dann irgendwann freiwillig wieder abgibt. Damit ist auch schon der Hauptnachteil klar: Ein fehlerhaftes Programm kann das gesamte System zum Erliegen bringen. Andererseits wird möglichst wenig Zeit mit überflüssigen Prozeßwechseln verbracht.

Ein weiteres wichtiges Stichwort heißt "Ereignisorientierung". In ereignisorientierten Systemen teilen die einzelnen Prozesse dem Betriebssystem mit, auf welche Art von Ereignissen (wie dem Verstreichen einer Zeitspanne oder eine Tastatureingabe) sie warten. Mit Hilfe dieser Information kann das Betriebssystem die Prozessorzeit sehr effektiv verteilen.

Digital Research hat für die AES ein kooperatives, ereignisorientiertes Multitasking ausgewählt. Umschaltungen zwischen AES-Prozessen finden immer nur dann statt, wenn ein Prozeß freiwillig die Kontrolle abgibt, sprich einen AES-Aufruf tätigt. Ansonsten verbringen die allermeisten GEM-Programme ihre Zeit mit dem Warten auf ein Ereignis – ob ein Tastendruck, eine Fenstermanipulation oder etwas anderes.

In den aktuellen Versionen des Atari-GEM können insgesamt acht verschiedene AES-Prozesse gleichzeitig ablaufen. Einer davon ist immer das "Hauptprogramm", und sechs weitere "Slots" (also Platzhalter für Prozesse) sind für bis zu sechs Accessories reserviert. Der achte im Bunde ist der "Screen-Manager", der für die Fensterelemente und die Drop-Down-Menüs zuständig ist.

Der Dispatcher

Die Verteilung der Prozessorzeit wird vom Dispatcher übernommen. Um Überblick und Gerechtigkeit zu wahren, legt er zwei Listen an, in denen alle Prozesse, die bereit zum Ablauf sind (READY-LIST), und alle, die auf ein Ereignis warten (NOT-READY-LIST), verzeichnet sind.

Wie geht der Dispatcher nun vor? Zunächst genehmigt er dem Prozeß, der in der READY-Liste ganz vorne steht, einen "Durchgang": Das heißt, er wartet darauf, daß dieser Prozeß irgendwann mal wieder einen AES-Aufruf tätigt. Daher sollte man es sich zur Angewohnheit

machen, in Programmschleifen, die völlig ohne AES-Aufrufe auskommen, den einen oder anderen "evnt_timer()"-Aufruf einzustreuen.

Ist nun wieder der Dispatcher am längeren Hebel, prüft er zunächst, ob eines der Ereignisse, auf das die Prozesse in der NOT-READY-Liste so sehnsüchtig warten, eingetreten ist. In diesem Fall wird der betreffende Prozeß von der NOT-READY-Liste an das Ende der READY-Liste gesetzt.

Weiterhin ist zu betrachten, ob nun möglicherweise der zuletzt aktive Prozeß auf ein Ereignis wartet. In einem solchen Fall wird er an das Ende der NOT-READY-Liste verbannt, andernfalls wird er das Schlußlicht der READY-Liste.

In den aktuellen Atari-AES sind sowohl Anzahl als auch Belegung der Prozeß-Slots von Beginn an fest. Accessories und Screen-Manager werden nie beendet, und das Hauptprogramm wird nach Beendigung sofort durch ein anderes (normalerweise das Desktop) ersetzt. In künftigen AES-Versionen wird es vermutlich möglich sein, mehrere Hauptprogramme gleichzeitig zu starten – Accessories werden dadurch weitestgehend ihre Existenzberechtigung verlieren.

Der Screen Manager

Der Screen Manager ist immer aktiv und überwacht die Position des Mauszeigers, wenn dieser den Arbeitsbereich der Fenster anderer Applikationen verläßt. Die hier in Frage kommenden Flächen sind die Rahmen der Fenster, die "Drop-Down"-Menüs und die Menüleiste.

Beim Berühren des Menübereichs sorgt der Screen Manager selbsttätig dafür, daß der vom Menü belegte Bildschirmausschnitt gesichert und anschließend wiederhergestellt wird (dazu wird der sogenannte "Quarter Screen Buffer" benutzt, auf den wir später zurückkommen).

Auch Manipulationen an den Fensterkontrollen führen nicht zu dauerhaften Veränderungen des Bildspeichers (es werden nur jeweils die Umrisse bewegt). Resultat der Interaktionen mit dem Screen Manager sind die sogenannten "Mitteilungsereignisse", die die zuständige Applikation über die Aktionen des Benutzers informieren.

Einschränkungen

Obwohl die AES viele gute Ansätze enthalten, machen sich doch Herkunft (PC unter MS-DOS) und Alter (1984) stark bemerkbar. Daher auch viele Einschränkungen, die die Praxistauglichkeit des Multitaskings in den aktuellen AES-Versionen stark einschränken.

Zunächst einmal muß man sich der traurigen Wahrheit stellen, daß das GEMDOS rein gar nichts vom Multitasking der AES weiß. Hauptprogramm und alle Accessories müssen sich für alle GEMDOS-Aufrufe (Speicher, Dateikennungen, DTA usw.) den gleichen GEMDOS-Prozeß teilen. Dadurch werden viele Projekte erheblich beschwert, zumal auch das VDI (Workstation öffnen, Zeichensätze laden) und viele Standard-Bibliotheksfunktionen (in C häufig die Benutzung der Standard-I/O-Routinen) Speicher vom GEMDOS anfordern.

Für weitere Kopfschmerzen sorgt ein Problem, das erst in der AES-Version 3.0 beseitigt wurde: GEMDOS-Aufrufe können ja bekanntlich zur Alarmbox des "Critical Error Handler" führen. *Während* die Alarmbox auf dem Bildschirm steht, kann es zu AES-Prozeßwechseln kommen. Wenn der ans Ruder gekommenene Prozeß nun wiederum eine GEMDOS-Funktion aufruft, kommt es zu einem Absturz (merke: GEMDOS ist *nicht* re-entrant – das heißt, GEMDOS-Funktionen können nicht aus GEMDOS-Funktionen heraus aufgerufen werden).

Diese Situation kann beispielsweise dann eintreten, wenn in einem Accessory ein GEMDOS-Aufruf auf "evnt_timer()" folgt (Abhilfe: GEMDOS-Aufrufe mittels der Funktion "wind_update()" klammern, damit ist der AES-Prozeß so lange blockiert, bis die Alarmbox wieder verschwunden ist).

Der Dispatcher selbst ist auch nur eingeschränkt für die Zukunft gerüstet. So sorgt er zwar für die Umschaltung der Prozessor-Register, nicht aber für die der FPU. Daher kann zur Zeit nur ein AES-Prozeß gleichzeitig FPU-Register benutzen (außer man sorgt selbst vor dem Aufruf von AES-Funktionen dafür, daß die Register gesichert werden).

Wie schon erwähnt, wird die Kommunikation zwischen den verschiedenen AES-Prozessen über "Mitteilungen" geregelt. Die AES bedienen sich einer Warteschlange ("Message-Queue"), um noch nicht verschickte Mitteilungen zu speichern. Diese Queue hat in den existierenden AES-Versionen eine fixe Länge, so daß nicht beliebig viele Mitteilungen gleichzeitig "unterwegs" sein können. Daher ist spezielle Vorsicht geboten, wenn ein Prozeß Mitteilungen an sich selbst schickt (ein beliebter Trick zum Auslösen von "Redraws") – ein Überlaufen der Message-Queue (die in den bestehenden AES-Versionen offenbar eine Länge von 16 Einträgen hat) führt zum Blockieren des sendenden Prozesses.

Die Initialisierung der AES

Aufruf nach der BIOS-Initialisierung

In den aktuellen TOS-Versionen sind AES und Desktop zu einem gemeinsamen ausführbaren Programm zusammengefaßt. Dies – und nicht irgendwelche bösen Absichten – ist auch der Grund dafür, daß das Desktop keine echten AES-Aufrufe, sondern nur einfache Unterprogramm-

aufrufe macht. Das BIOS erzeugt nach Abarbeitung des AUTO-Ordners einen neuen GEMDOS-Prozeß und fügt als Startadresse des Text-Segments (also als Einsprungadresse) den Inhalt der Systemvariablen "exec_os" ein. Dort wird dann die Ausführung fortgesetzt.

AES-Environment

Zu Beginn wird das GEMDOS-Environment in einen eigenen Puffer fixer, beschränkter Länge kopiert und dort später von der Shell-Bibliothek ("shel_envrn()" und "shel_find()") benutzt.

Laden der Accessories

An dieser Stelle wird ab Atari-GEM 3.0 ein neuer GEMDOS-Prozeß erzeugt, der später bei einem Auflösungswechsel wieder terminiert wird. Nur so ist es möglich, beim Wechsel der Auflösung wirklich allen von Accessories belegten Speicher wieder freizugeben. Unter älteren GEM-Versionen blieben von Accessories allozierte Speicherblöcke belegt, was nicht nur zur Reduzierung, sondern auch zur starken Zersplitterung des freien Speichers führte.

Bei der Lokalisierung der Accessories legen die AES ein sehr verwunderliches Verhalten an den Tag. Wissen sollte man, daß die AES die Accessories *immer* von Laufwerk C: laden, falls dieses existiert und dort mindestens eine Datei mit der Namenserweiterung "ACC" gefunden wird. Ähnliches gilt für das Laden der "DESKTOP.INF"- bzw. "NEWDESK.INF"-Datei. Eine wirklich portable Methode, mit der man die AES dazu bewegen könnte, die Accessories vom zum Zeitpunkt der Initialisierung aktuellen oder dem in "_bootdev" angegebenen Laufwerk zu laden, ist *nicht* bekannt.

Es folgt der eigentliche Ladevorgang. Jede gefundene Accessory-Datei (in den aktuellen AES-Versionen: maximal sechs) wird mit "Pexec()" (Modus 3) geladen und reloziert. Damit werden die Einstellungen im Programmkopf (siehe GEMDOS-Kapitel) in Hinsicht auf das *Laden* des Prozesses sehr wohl beachtet. Anschließend wird die vom "Pexec()"-Aufruf erzeugte TPA mittels "Mshrink()" auf die "normale" Größe (die Summe der Längen der drei Programmsegmente zuzüglich 256 Bytes für die Basepage) geschrumpft.

Initialisierung der VDI-Workstation

Anschließend öffnen die AES eine physikalische VDI-Workstation für den Bildschirm. Die gewünschte Bildschirmauflösung wird dabei als Gerätenummer an "v_opnwk()" übergeben (sehr wichtig beim Auflösungswechsel). Später gestartete Programme dürfen nur noch

virtuelle Bildschirm-Workstations öffnen und müssen dazu mittels "graf_handle()" die Kennung der von den AES benutzten physikalischen Workstation erfragen.

Hinweis: Alle zur Zeit bekannten AES-Versionen stürzen ab, wenn der "v_opnwk()"-Aufruf zu einem Fehler führt.

Damit sind insbesondere die VDI-Eingabefunktionen (die nur auf einer physikalischen Workstation benutzt werden dürfen) für die AES reserviert. Die AES benutzen das Rasterkoordinatensystem des VDI. Der Ursprung liegt damit in der linken oberen Ecke des Bildschirms. Die vom VDI gelieferten Angaben werden von den AES sauber verarbeitet. Nur deshalb ist es auch möglich, eigene VDI-Gerätetreiber für zusätzliche Grafikkarten zu installieren und dann AES und Desktop darauf laufen zu lassen.

Der "Quarter Screen Buffer"

Dieser Speicherbereich wird vom Screen Manager benötigt, um beim Herunterklappen von Drop-Down-Menüs den Inhalt des Menühintergrunds zu retten. Auch bei der Anzeige von Alarmboxen kommt der "QSB" (so die gebräuchliche Abkürzung) zum Einsatz. Normalerweise sollte seine Größe von der Anzahl der Farbebenen und der Größe des Systemzeichensatzes, nicht aber von der Gesamtgröße des Bildschirms abhängen. Eine gute Formel wäre zum Beispiel:

500 (Zeichen) * Platzbedarf eines Zeichens * Farbebenen

Damit käme man in der Auflösung "ST-Hoch" genau auf den Wert 8000 (also ein Viertel des Bildspeichers). Leider sind die AES in der Praxis nicht so clever. Hier eine Übersicht über die verschiedenen zur Zeit benutzten Algorithmen zur Bestimmung der Puffergröße:

GEM-Version	Methode zum Setzen des QSB
1.0, 1.2	statisch, 8000 Bytes
1.4	dynamisch, ein Viertel des Bildspeichers
3.0	dynamisch, die Hälfte des Bildspeichers

Die GEM-Versionen 1.0 und 1.2 (also bis einschließlich TOS 1.02) sind mithin nicht für Farbkarten vorbereitet – einer unter mehreren Gründen, warum man selbst bei Benutzung eines speziellen VDI-Treibers unter diesen GEM-Versionen Farbgrafikkarten nicht einsetzen kann.

Das bei GEM 1.4 benutzte Verfahren war eigentlich sehr sinnvoll, konnte aber wegen der "TT-Niedrig"-Auflösung nicht beibehalten werden. Rechenexempel: Die Bildspeichergröße

beträgt 153600 Bytes. Ein Viertel davon sind 38400 Bytes, die bei acht Farbebenen und 16 Bytes pro Zeichen Platz für nur 300 Zeichen bieten. Daher ist man in GEM 3.0 dazu übergegangen, gleich die Hälfte einer Bildspeicherlänge zu reservieren. Das funktioniert zwar, ist jedoch eine immense Speicherplatzverschwendung (bei einer Farbgrafikkarte mit 1024 * 768 Punkten in 256 Farben: über 390 KByte Puffer!). Hier sollte Atari also künftig eine flexiblere Formel (wie die oben vorgeschlagene) einsetzen.

Startupcode für Accessories

Accessories werden zwar wie echte GEMDOS-Prozesse geladen, damit hört die Ähnlichkeit aber schon auf. Die vom "Pexec()"-Aufruf angelegte Basepage ist nicht vollständig ausgefüllt, und anders als bei einem normalen GEMDOS-Programm wird beim Aufruf der Basepage-Zeiger nicht auf dem Stack, sondern in Adreßregister A0 übergeben. Darüber hinaus muß unbedingt ein eigener Stack angelegt werden, da der Stackpointer beim Aufruf normalerweise auf Null initialisiert ist (genaugenommen wird er mit einem zufälligen Wert initialisiert, der eben praktisch immer Null ist). Zusammenfassung:

- Wenn bei Programmstart A0 den Wert Null hat, dann handelt es sich um einen normalen Programmstart (wie im GEMDOS-Kapitel beschrieben).

- Anderenfalls handelt es sich um ein Accessory, und A0 enthält einen Zeiger auf die unvollständig ausgefüllte Basepage. Die TPA ist bereits passend geschrumpft (auf die Summe von Basepagegröße und der Längen der drei Programmsegmente). Ein Stack muß erst noch angelegt werden.

Mit diesen Informationen ist es kein Problem, den Startup-Code für ein Programm so zu gestalten, daß er selbständig erkennt, wie das Programm gestartet worden ist, und entsprechend die Initialisierung fortsetzt. Bei den meisten neueren C-Compilern wird im Startupcode automatisch die externe Variable "_app" initialisiert, die genau dann gleich Null ist, wenn das Programm als Accessory gestartet worden ist. Damit kann man Programme so entwickeln, daß sie sowohl als Accessory als auch als normales Programm eingesetzt werden können.

Die Applikations-Bibliothek

Dieser Teil der AES beschäftigt sich mit allem, was mit der Installation und Abmeldung von Applikationen und der Kommunikation zwischen Applikationen zu tun hat.

Dies ist wegen der – wenn auch eingeschränkten – Multitasking-Fähigkeiten von GEM dringend nötig. Jede GEM-Applikation sollte sich mit "appl_init()" ordnungsgemäß anmel-

den. Dabei erhält sie eine eindeutige Kennung, meist als "ap_id" bezeichnet, die sie von allen anderen, gleichzeitig im Speicher befindlichen GEM-Prozessen unterscheidet. Daneben wird das GLOBAL-Feld (siehe auch unter Parameterübergabe) initialisiert, das wie folgt aussieht:

WORD global[15];

global[0] (ap_version):	Hier steht die AES-Versionsnummer. Zur Zeit mögliche Werte für Atari-GEM sind 0x0100 (GEM 1.0), 0x0120 (GEM 1.2), 0x0140 (GEM 1.4) und 0x0300 (GEM 3.0).
global[1] (ap_count):	Maximalzahl von gleichzeitig unterstützten Applikationen (wird von den AES eingetragen) und ist zur Zeit "1".
global[2] (ap_id):	Kennung der Applikation (wird von den AES eingetragen).
global[3,4] (ap_pprivate):	Irgendwelche Informationen, die nur für die Applikation von Bedeutung sind und auch von ihr gesetzt und gelesen werden. Unter PC-GEM: "ob_spec" von Fenster 0 (kann benutzt werden, um Standardhintergrundfarbe und -muster abzufragen).
global[5,6] (ap_ptree):	Zeiger auf eine Liste von Zeigern auf die Objektbäume der Applikation. Sollte zunächst auf 0 initialisiert werden und wird dann von "rsrc_load()" gesetzt.
global[7,8] (ap_pmem):	Anfangsadresse des für die Resource-Datei reservierten Speichers (nicht von Atari, wohl aber von Digital Research dokumentiert).
global[9] (ap_lmem):	Länge des reservierten Speichers (nicht von Atari, wohl aber von Digital Research dokumentiert).
global[10] (ap_nplanes):	Anzahl der Farbebenen (nicht von Atari, wohl aber von Digital Research dokumentiert).
global[11,12]:	Reserviert.
global[13] (ap_bvdisk):	Im PC-GEM ab Version 2.0: Bitvektor mit den auf dem Desktop angemeldeten Laufwerken (Bit 15: "A" etc). Im Atari-GEM: reserviert.
global[14] (ap_bvhard):	Im PC-GEM ab Version 2.0: Bitvektor, der angibt, welche der in "ap_bvdisk" angegebenen Laufwerke als Festplatte betrachtet werden sollen. Im Atari-GEM: reserviert.

Die Ereignis-Bibliothek

Einleitung

Bevor wir auf die verschiedenen Typen von Ereignissen eingehen, soll hier noch einmal auf die entscheidende Bedeutung dieser Funktionen für das Multitasking unter GEM eingegangen

werden. Der Grundgedanke ist, daß Programme, die auf irgendein Ereignis warten, dies nicht dadurch realisieren, daß sie in einer großen Schleife alle möglichen Ereignisse durchchecken, sondern indem sie den AES mitteilen, worauf sie warten und sich dann vorläufig zur Ruhe begeben. In dieser Wartezeit können die AES die gesamte Prozessorzeit anderen Prozessen zuweisen. Die Applikation wird wieder aktiviert, wenn das erwartete Ereignis dann eingetreten ist.

Arten von Ereignissen (events)

Tastatur-Ereignis:	Es wird ein Tastendruck des Benutzers erwartet.
Mausknopf-Ereignis:	Es wird auf einen auszuwählenden Zustand der Maustasten gewartet.
Maus-Ereignis:	Es wird darauf gewartet, daß der Mauszeiger einen rechteckigen Teil des Bildschirms betritt oder verläßt.
Mitteilungs-Ereignis:	Die Applikation wartet auf eine Nachricht von einem anderen Prozeß (im allgemeinen dem Screen Manager, der das Anklicken von Menüs oder Veränderungen der Fenster mitteilt).
Timer-Ereignis:	Es wird auf das Verstreichen einer bestimmten Zeitspanne gewartet (auf gar keinen Fall statt dessen eine handgestrickte Warteschleife benutzen).

Zusätzlich kann man auf eine beliebige Kombination solcher Ereignisse warten (Multi-Ereignis).

Mitteilungs-Ereignisse

Mitteilungs-Ereignisse dienen zur Kommunikation zwischen verschiedenen AES-Prozessen. In den allermeisten Fällen handelt es sich um Nachrichten, die der Screen Manager an ein Accessory oder die Hauptapplikation schickt. Die AES verschicken Mitteilungen in Paketen zu je 16 Bytes, deren Format so aussieht:

```
WORD mbuf[8];
```

mbuf[0]:	Nachrichtennummer
mbuf[1]:	Nummer (ap_id) des Absenders

mbuf[2]: Anzahl der Bytes, die noch nachträglich per "appl_read()" gelesen werden müssen.

Dies ist allerdings nur eine Konvention, die eigentlich nirgendwo erzwungen wird. Nur bei Redraw-Aufforderungen (die anhand der Nachrichtennummer erkannt werden) nehmen sich die AES die Freiheit, mehrere Mitteilungen zu einer einzigen zu sammeln.

"mbuf[2]" ist zur Zeit bei allen vom Screen Manager verschickten Mitteilungen Null (da sie immer nur 16 Bytes umfassen).

Laut Digital Research sollte man also ungefähr so vorgehen:

```
evnt_mesag (mbuf);

if (mbuf[2] > 0)
{
     appl_read (ap_id, mbuf[2], puffer);
}
```

Mitteilungen mit unbekannten Nachrichtennummern müssen ignoriert werden. Schließlich könnte der Screen Manager eines Tages erweitert werden oder mal eine benutzerdefinierte Mitteilung eines anderen Programms eintreffen.

Für die einzelnen Ereignistypen hat der Rest des Nachrichtenpuffers jeweils unterschiedliche Bedeutungen:

MN_SELECTED (10)

Ein Eintrag in einem Drop-Down-Menü ist angeklickt worden:

mbuf[3]: Objektnummer des Menütitels
mbuf[4]: Objektnummer des Menüeintrags

WM_REDRAW (20)

Ein Teil des Arbeitsbereiches eines Fensters muß neu gezeichnet werden:

mbuf[3]: Kennung des wiederherzustellenden Fensters
mbuf[4]: X-Koordinate des wiederherzustellenden Fensterausschnitts
mbuf[5]: Y-Koordinate des wiederherzustellenden Fensterausschnitts
mbuf[6]: Breite des wiederherzustellenden Fensterausschnitts
mbuf[7]: Höhe des wiederherzustellenden Fensterausschnitts

Bemerkungen
Mehrere aufeinanderfolgende "WM_REDRAW"-Mitteilungen werden von den AES zu einer einzigen Mitteilung zusammengefaßt.

WM_TOPPED (21)

Dieses Fenster wird zum aktuellen Fenster.

mbuf[3]: Kennung des Fensters

WM_CLOSED (22)

Das Schließfeld des Fensters ist angeklickt worden.

mbuf[3]: Kennung des Fensters

WM_FULLED (23)

Die Full-Box (volle Bildschirmgröße) des Fensters ist angeklickt worden.

mbuf[3]: Kennung des Fensters

WM_ARROWED (24)

Einer der Pfeile ist angeklickt worden.

mbuf[3]: Kennung des Fensters
mbuf[4]: Art der geforderten Veränderung:

WA_UPPAGE (0):	Seite aufwärts (Balken oberhalb des Schiebers angeklickt)
WA_DNPAGE (1):	Seite abwärts (Balken unterhalb des Schiebers angeklickt)
WA_UPLINE (2):	Zeile aufwärts (Pfeil nach oben angeklickt)
WA_DNLINE (3):	Zeile abwärts (Pfeil nach unten angeklickt)
WA_LFPAGE (4):	Seite links (Balken links des Schiebers angeklickt)
WA_RTPAGE (5):	Seite rechts (Balken rechts des Schiebers angeklickt)
WA_LFLINE (6):	Spalte links (Pfeil nach links angeklickt)
WA_RTLINE (7):	Spalte rechts (Pfeil nach rechts angeklickt)

WM_HSLID (25)

Der horizontale Schieber eines Fensters wurde bewegt.

mbuf[3]: Kennung des Fensters
mbuf[4]: von 0 (ganz links) bis 1000 (ganz rechts)

WM_VSLID (26)

Der vertikale Schieber eines Fensters wurde bewegt.

mbuf[3]: Kennung des Fensters
mbuf[4]: von 0 (ganz oben) bis 1000 (ganz unten)

WM_SIZED (27)

Die Größe eines Fensters wurde geändert (dabei sind die Außenmaße gemeint).

mbuf[3]: Kennung des Fensters
mbuf[4]: X-Koordinate
mbuf[5]: Y-Koordinate
mbuf[6]: neue Breite
mbuf[7]: neue Höhe

WM_MOVED (28)

Die Position eines Fensters wurde geändert (wiederum Angabe der Außenmaße).

mbuf[3]: Kennung des Fensters
mbuf[4]: Neue X-Koordinate
mbuf[5]: Neue Y-Koordinate
mbuf[6]: Breite (sollte gleichgeblieben sein)
mbuf[7]: Höhe (sollte gleichgeblieben sein)

WM_NEWTOP (29)

Diese Mitteilung hatte wohl eine ähnliche Funktion wie "WM_TOPPED", ist aber mittlerweile nicht mehr dokumentiert und scheint auch nicht aufzutreten.

WM_UNTOPPED (30) – nur in PC-GEM ab Version 2.0

Das betreffende Fenster wird gerade inaktiv. Diese Nachricht kann man benutzen, um vorher den Fensterinhalt irgendwo zu sichern!

mbuf[3]: Kennung des Fensters

AC_OPEN (40)

Der Eintrag des Accessories im ersten Drop-Down-Menü wurde angeklickt.

mbuf[4]: Kennung (menu_id) des Menü-Eintrags

Bemerkungen
Dies scheint ein echter Fehler im Atari-GEM zu sein, da in sämtlichen PC-Dokumentationen mbuf[3] angegeben wird – es bietet sich an, diese Information gänzlich zu ignorieren, da man die AC_OPEN-Nachricht sowieso nur für das eigene Accessory empfängt. Mehr als einen Menüeintrag (wie die erste Version des Kontrollfeld-Accessories) sollte man aus Rücksicht auf andere Accessories sowieso nicht verwenden.

AC_CLOSE (41)

Ein Accessory wurde geschlossen. Damit ist der Start oder das Ende einer Haupt-Applikation gemeint. Das Accessory braucht seine Fenster nicht selbst zu schließen!

mbuf[3]: Kennung (menu_id) des Menü-Eintrags

Bemerkungen
Erst ab Atari-GEM 3.0 darf man sicher sein, daß diese Mitteilung tatsächlich vor dem endgültigen Ende (also dem Verlust von Speicherblöcken und Dateikennungen) des Hauptprogramms eintrifft.

CT_UPDATE (50), CT_MOVE (51), CT_NEWTOP (52)

Von Digital Research so angegeben; Funktion unbekannt.

CT_KEY (53)

Diese Mitteilungsnummer wird von dem modularen Kontrollfeld "XCONTROL" zur Meldung von Tasteneingaben benutzt (Genaueres dazu bei der Dokumentation von "XCONTROL").

Benutzerdefinierte Mitteilungen

Neben den vordefinierten Mitteilungen kann man natürlich auch noch selbst definierte Nachrichten einsetzen – zum Beispiel, um ein Accessory und ein Hauptprogramm miteinander kommunizieren zu lassen. Digital Research schlägt für solche Nachrichten Mitteilungsnummern jenseits von 1024 vor.

Mittlerweile sind viele flexible Protokolle auf Basis der AES-Events festgelegt worden. Exemplarisch sei das "AV-Protokoll" genannt, das im alternativen Desktop "Gemini" für die Kommunikation zwischen Desktop und Accessories zuständig ist und auch leicht auf andere Programme angewandt werden kann. Einige nützliche Eigenschaften:

- Accessory-Fenster können in die Fensterverwaltung des Hauptprogramms aufgenommen werden (zum Beispiel für "Fenster wechseln").

- Accessories können Tastendrücke an das Hauptprogramm weiterleiten.

- Accessories können Statusinformationen durch das Hauptprogramm laden und sichern lassen.

- Accessories werden darüber informiert, wenn ein Bildsymbol in das Fenster des Accessories gezogen wurde.

- Accessories können Gemini-Fenster öffnen und Programme starten lassen.

Wie man sieht, bietet das "AV-Protokoll" allerlei nützliche Dinge. Eine vollständige Dokumentation finden Sie in der Gemini-Dokumentation.

Die Menü-Bibliothek

Zu den Grundlagen der Drop-Down-Menüs braucht nicht viel gesagt zu werden – jeder kennt sie aus Desktop und vielen Anwendungsprogrammen.

Anders steht es mit vielen Details: Welche Standardmenüs sollte es geben, wie zeigt man Tastatur-Shortcuts an? Die Antworten zu diesen Fragen finden Sie im Kapitel "Richtlinien zur Benutzerführung und Programmierung".

Die aktuellen AES-Versionen auf dem Atari erlauben nur eine Menüleiste – Accessories müssen also andere Wege der Benutzerführung gehen. Beim PC-GEM werden die Menüleisten automatisch umgeschaltet: je nachdem, welchem AES-Prozeß das aktive Fenster gehört. Man darf wohl davon ausgehen, daß es unter einem künftigen, multitaskingfähigen AES genauso funktionieren wird.

Noch eine Bemerkung zur Namensgebung: Drop-Down-Menüs sind etwas anderes als die "Pull-Down"-Menüs, wie sie zum Beispiel auf dem Macintosh benutzt werden. Erste "fallen herunter", wenn man sie mit dem Mauszeiger berührt. "Pull-Down"-Menüs hingegen werden erst ausgeklappt, wenn man mit *gedrückter* Maustaste in den Menübereich fährt.

"Drop-Down"-Menüs haben den Nachteil, daß man sie oft unabsichtlich aktiviert und erst durch einen Mausklick wieder los wird. Mac-artige Menüs hingegen sind meist für Anfänger schwerer zu bedienen, weil sie die Koordination von Maustaste und Mausbewegung erfordern. Ab PC-GEM Version 3 kann man das Verhalten der Menüs mittels "menu_click()" einstellen.

Wenden wir uns nun der interessanten Frage zu, was denn so ein Menü-Objektbaum (den man mit "menu_bar()" auf den Bildschirm bringt) eigentlich ist. Dazu schauen wir uns einmal die Objektstruktur an, die ein Resource-Construction-Programm erzeugt, wenn man eine MENÜ-Struktur entwirft.

Der Objektbaum besteht aus insgesamt 17 Objekten (nähere Informationen zu Objekten übrigens im Kapitel über die Objekt-Bibliotheken...):

Objekt 0: Äußere Box (IBOX), die den gesamten Baum umfaßt

Objekt 1: Unsichtbare Box für den Menü-Balken (enthält Objekt 2)

Objekt 2: Solide weiße Box, die alle Menü-Titel umfaßt (enthält Objekt 3 und 4)

Objekt 3: Titel "Desk"

Objekt 4: Titel "File"

Objekt 5: Unsichtbare Box, liegt genau unterhalb des Menü-Balkens (enthält Objekte 6 und 15)

Objekt 6: Box für das Desk-Menü (enthält als Objekte die einzelnen Einträge 7–14)

Objekt 7: Eintrag "Your message here..."

Objekt 8: Eintrag "----------"

Object 9: Eintrag "Desk Accessory 1 "

Object 10: Eintrag "Desk Accessory 2 "

Object 11: Eintrag "Desk Accessory 3 "

Object 12: Eintrag "Desk Accessory 4 "

Object 13: Eintrag "Desk Accessory 5 "

Object 14: Eintrag "Desk Accessory 6 "

Objekt 15: Box für das File-Menü (enthält Objekt 16)

Objekt 16: Eintrag "Quit "

Schließlich noch ein Hinweis auf künftige Weiterentwicklungsmöglichkeiten: Wann immer möglich, sollte man die Funktionen der Menü-Bibliothek benutzen, anstatt direkt auf die Objekte zuzugreifen. Die momentan benutzte Struktur schränkt die Anzahl der Accessory-Einträge auf sechs ein, so daß unter Umständen eines Tages eine Umstellung der internen Repräsentation der Menüs vorgenommen werden könnte.

Die Objekt-Bibliothek

Das Konzept

Objekte sind das grundlegende Element der AES. Dateiauswahlbox, Alarmboxen, Fenster, Drop-Down-Menüs – alles ist intern aus Objekten zusammengesetzt.

Ein "object" ist nichts anderes als eine Datenstruktur, die ein grafisches Objekt (wie eine Box, einen Text oder einen Knopf) beschreibt. Dazu gehören in erster Linie Typ (Knopf, Texteingabefeld...), Position und verschiedene Attribute (zum Beispiel das Füllmuster eines Box-Objekts).

Die einzelnen Objekte sind in einer *visuellen Hierarchie* geordnet. Mit Hierarchie ist in diesem Zusammenhang gemeint, daß zu jedem Objekt genau ein übergeordnetes *Parent*-Objekt gehört (außer es handelt sich bereits um das höchste Objekt). "Visuell" steht dafür, daß sich die Hierarchie nicht nur in der Datenstruktur, sondern auch in der Repräsentation auf dem Bildschirm wiederspiegelt.

So ist in einer einfachen Alarmbox jedes Objekt (also Texte, Knöpfe und das Bildsymbol) ein *Child*-Objekt der Hintergrundbox. Dieses nennt man oft auch das Wurzelobjekt. Die Hierarchieebenen können natürlich auch noch weitergehen. Oft sieht man mehrere Knöpfe, die gemeinsam in einem umgebenden Rahmen zusammengefaßt sind.

Vom Modell des objekt-orientierten Programmierens kennt man die Idee, daß ein Parent-Objekt an die ihm untergeordneten Objekte Eigenschaften *vererbt*. Bei den AES-Objekten handelt es sich freilich nur um den Bezugspunkt der Positionsangabe. Diese ist also immer als *Offset* zur Position des übergeordneten Objekts zu verstehen. Nur beim Wurzelobjekt handelt es sich um eine absolute Bildschirmposition.

Jedes Objekt bedeckt ein genau definiertes Rechteck (dessen Breite und Höhe in der Objektstruktur angegeben ist). Sinn und Zweck der visuellen Hierarchie ist, daß für jedes Objekt das zugehörige Rechteck vollständig in dem des Parent-Objekts enthalten ist. Die meisten Resource-Construction-Programme fördern diese Gliederung, indem sie vor Überlap-

pungen warnen. Zur Veranschaulichung wollen wir hier den Algorithmus darstellen, den "objc_offset()" benutzt, um die absolute Bildschirmposition eines Objekts zu ermitteln:

```
WORD
objc_offset (OBJECT *Tree, WORD ObNum, WORD *Xoff, WORD *Yoff)
{
     WORD sumx = 0, sumy = 0;          /* Auf 0 initialisieren */

     do
     {
          sumx += Tree[ObNum].ob_x;    /* Position aufaddieren */
          sumy += Tree[ObNum].ob_y;
          ObNum = objc_get_parent (Tree, ObNum);
     } while (ObNum != -1);            /* -1: Wurzel erreicht */

     *Xoff = sumx;
     *Yoff = sumy;

     return TRUE;
}
```

Es werden also die X- und Y-Positionen des betreffenden Objekts und aller übergeordneter Objekte aufsummiert.

Auch "objc_draw()" macht sich die Gliederung eines Objektbaums zunutze. Es beginnt bei dem Wurzelobjekt und steigt dann rekursiv immer tiefer die Objekthierarchie hinunter. Folge ist der allseits bekannte Effekt, daß die Objekte der untersten Ebene zuletzt und damit "zuoberst" gezeichnet werden.

Eine besonders aufwendige *und* zeitkritische Aufgabe ist es, das oberste, an einer bestimmten Bildschirmkoordinate liegende Objekt zu finden. Dieses Problem müssen die AES bei jedem einzelnen Mausklick in eine Dialogbox bearbeiten.

Die hierarchische Gliederung macht es einfach: Ein Unterbaum eines Objekts braucht nur dann untersucht zu werden, wenn die Koordinaten innerhalb des Rechtecks des Parent-Objekts liegen. Auf diese Weise werden die Suchzeiten drastisch reduziert (für Mathematiker: von $O(n)$ auf $O(\log n)$).

Wenn man jetzt noch weiß, daß die Fenster intern als Objektbaum dargestellt werden und ein Aufruf von "wind_find()" ziemlich direkt zu einem Aufruf von "objc_find()" führt, wird dieses Objekt-Modell wohl auch recht einleuchtend erscheinen.

Die Implementation

Baumartige Datenstrukturen sind zwar sehr flexibel, dafür aber im Vergleich zu einfachen Feldern etwas schwerfällig. Auf modernen Workstations würde man vielleicht tatsächlich einen "echten" Baum benutzen. 1984 galt es jedoch, möglichst wenig Platz zu verbrauchen und so effizient wie nur möglich zu sein.

Objektbäume werden deshalb als Felder (Arrays) von einzelnen Objektstrukturen dargestellt. Dadurch kann bei Kenntnis der Adresse des Wurzelobjekts direkt auf jedes Objekt des Baums zugegriffen werden, ohne erst lang verzeigerte Strukturen durchlaufen zu müssen. Die Verkettung der einzelnen Objekte erfolgt nicht über echte Zeiger, sondern jeweils über deren Nummer im Feld. Nachteil dieses Konzepts ist, daß bestehende Bäume nur durch Umkopieren vergrößert werden können. Die Gliederung erfolgt über die drei speziellen Einträge in der Objekt-Struktur:

ob_next: Nummer des folgenden Objekts gleicher Ebene oder – falls es das letzte Element in der Ebene ist – des Parent-Objekts.

ob_head: Nummer des ersten Kinds des Objekts.

ob_tail: Nummer des letzten Kinds des Objekts.

Zeiger, die nirgendwohin weisen (zum Beispiel, weil es keine tiefere Ebene mehr gibt), haben den Wert "–1" (gemeinhin NIL genannt).

Bevor wir uns nun in alle Spielarten von Objekten vertiefen, sei darauf verwiesen, daß man Objektbäume (im Atari-Jargon auch gerne Resources genannt) üblicherweise nicht "zu Fuß" im Speicher zusammensetzt (das geht natürlich auch), sondern mit einem "Resource Construction Set" (RCS) entwirft und in einer Resource-Datei ablegt, die das Programm dann mittels "rsrc_load()" lädt. Ein sehr leistungsfähiges RCS ist das Programm "Interface", das von der Firma Shift vertrieben wird.

Die Objekt-Struktur

Ein Objekt hat folgende Struktur:

```
typedef struct
{
    WORD    ob_next;    /* Nummer des nächsten Objekts */
    WORD    ob_head;    /* Nummer des ersten Kinds */
    WORD    ob_tail;    /* Nummer des letzten Kinds */
```

```
    UWORD     ob_type;      /* Art des Objekts */
    UWORD     ob_flags;     /* Verschiedene Flags */
    UWORD     ob_state;     /* Status des Objekts */
    void      *ob_spec;     /* typabhängig */
    WORD      ob_x;         /* X-Position (rel. zum Parent-
                               Objekt) */
    WORD      ob_y;         /* Y-Position (rel. zum Parent-
                               Objekt) */
    WORD      ob_width;     /* Breite */
    WORD      ob_height;    /* Höhe */
} OBJECT;
```

Das Strukturelement "ob_spec" ist 32 Bits groß.

Seine Bedeutung hängt ganz von "ob_type" des Objekts ab. Für C-Programmierer erscheint daher die Deklaration einer "Union" als sinnvoll, damit man ohne die lästige Anwendung von "typecasts" zugreifen kann:

```
/* Deklaration von "OBSPEC" als Union, angelehnt an die Definition
   im AES.H des Turbo-C-Compilers */

typedef union obspecptr
{
    LONG                index;
    union obspecptr     *indirect;
    bfobspec            obspec;       /* siehe unten */
    TEDINFO             *tedinfo;
    ICONBLK             *iconblk;
    BITBLK              *bitblk;
    USERBLK             *userblk;
    char                *free_string;
} OBSPEC;
```

Die Deklaration von "ob_spec" in der OBJECT-Struktur ändert sich dadurch zu:

```
OBSPEC ob_spec;
```

Zum Zugriff auf eine TEDINFO-Struktur würde man also

```
Tree[ob].ob_spec.tedinfo
```

anstelle von

```
(TEDINFO *)Tree[ob].ob_spec
```

schreiben. Ähnliche Deklarationen sind auch in Pascal oder in Modula-2 denkbar.

Zu besprechen bleiben also noch die Strukturelemente "ob_type", "ob_flags", "ob_state" und "ob_spec":

Objekttypen (ob_type)

Das obere Byte des Objekttyps wird von den AES komplett ignoriert und steht daher für eigene Anwendungen zur Verfügung (speziell als Zusatzinformation für selbstdefinierte Objekttypen, wie man sie mittels der USERDEF-Struktur erzeugen kann).

Wenn man in eigenen Programmen Entscheidungen anhand des Objekttyps fällt, sollte man also *immer* die obersten acht Bits ausmaskieren!

G_BOX (20)	Rechteckiger Kasten mit diversen Attributen. "ob_spec" enthält verschiedene Informationen über Rahmenstärke, Farbe und ähnliches.
G_TEXT (21)	Grafik-Text. "ob_spec" zeigt auf eine TEDINFO-Struktur.
G_BOXTEXT (22)	Rechteckiger Kasten mit Grafik-Text. "ob_spec" zeigt auf eine TEDINFO-Struktur.
G_IMAGE (23)	Ein einfaches Bild. "ob_spec" zeigt auf eine BITBLK-Struktur.
G_USERDEF (24)	Dieser Objekttyp erlaubt es, eigene Funktionen zur Ausgabe zu definieren. Die AES kümmern sich dabei auf Wunsch um die Darstellung der verschiedenen Objektzustände (siehe "ob_state"). "ob_spec" zeigt auf eine USERBLK-Struktur, die insbesondere einen Zeiger auf eine Ausgabefunktion enthält. Dieser Objekttyp wurde ursprünglich als "G_PROGDEF" bezeichnet.
G_IBOX (25)	Unsichtbares Rechteck (ohne Inneres). Nur wenn die Umrandung nicht die Dicke Null hat, kann man es sehen. "ob_spec" enthält weitere Informationen über das Aussehen.
G_BUTTON (26)	Zentrierter Text in einem Rechteck. "ob_spec" zeigt auf eine Zeichenkette mit dem Text, der in dem Knopf erscheinen soll.

G_BOXCHAR (27)	Rechteck, das einen einzigen Buchstaben enthält. In "ob_spec" wird nicht nur das Aussehen der Umrandung, sondern auch das Zeichen definiert (Detail am Rande: Die "Eckelemente" der AES-Fenster werden intern mit diesem Objekttyp erzeugt).
G_STRING (28)	Zeichenkette. "ob_spec" zeigt auf den entsprechenden String im Speicher.
G_FTEXT (29)	Formatierter Grafik-Text. "ob_spec" zeigt auf eine TEDINFO-Struktur.
G_FBOXTEXT (30)	Rechteck mit formatiertem Grafik-Text. "ob_spec" zeigt auf eine TEDINFO-Struktur.
G_ICON (31)	Icon. "ob_spec" zeigt auf eine ICONBLK-Struktur.
G_TITLE (32)	Titel eines Drop-Down-Menüs. "ob_spec" zeigt auf eine Zeichenkette im Speicher.

Viele Bits im "ob_spec"

Für G_BOX, G_IBOX und G_BOXCHAR zeigt "ob_spec" nicht auf eine andere Datenstruktur, sondern enthält weitere Informationen zum Aussehen des Objekts:

Bit 24..31:	Darzustellendes Zeichen (nur bei G_BOXCHAR, ansonsten unbenutzt)	
Bit 16..23:	0:	Rahmendicke 0 (kein Rahmen)
	1 bis 128:	Der Rand liegt 1 bis 128 Pixel im Inneren des Objekts
	−1 bis −127:	Der Rand liegt 1 bis 127 Pixel außerhalb des Objekts
Bit 12..15:	Rahmenfarbe (0..15)	
Bit 8..11:	Textfarbe (0..15)	
Bit 7:	Text transparent (0) oder deckend (1)	
Bit 4..6:	IP_HOLLOW (0): hohl	
	IP_1PATT (1): ansteigende Intensität	
	IP_2PATT (2)	
	IP_3PATT (3)	

IP_4PATT (4)
IP_5PATT (5)
IP_6PATT (6)
IP_SOLID (7): solide Fläche

Bit 0..3: Innenfarbe (0..15)

Zum vereinfachten Zugriff bietet sich die Definition eines C-Bitfelds an (Voraussetzung ist dabei allerdings, daß der verwendete C-Compiler die Bits in der gleichen "Richtung" anordnet wie das hier benutzte Turbo-C):

```
typedef struct
{
     unsigned character       : 8;
     signed framesize         : 8;
     unsigned framecol        : 4;
     unsigned textcol         : 4;
     unsigned textmode        : 1;
     unsigned fillpattern     : 3;
     unsigned interiorcol     : 4;
} bfobspec;
```

Vorteil: Zum Zugriff auf einzelne Teile des Bitfelds kann man einfach den Namen des Feldelements angeben – ein aufwendiges Shiften und Ausmaskieren von Bits entfällt. *Beispiel*: Die Textfarbe erhält man mit:

```
Tree[ob].ob_spec.obspec.textcol
```

Objekt-Flags (ob_flags)

Für jedes Flag wird ein Bit in "ob_flags" benutzt. Außerdem ist meist noch die Konstante "NONE" (für keine Flags, also 0) vordefiniert.

SELECTABLE (0x0001) Der Benutzer kann das Objekt durch Anklicken selektieren.

DEFAULT (0x0002) Hebt normalerweise das Objekt optisch hervor und bedeutet, daß das Drücken von RETURN mit dem Anklicken dieses Objekts gleichgesetzt wird (bei Aufruf von "form_do()"). Innerhalb eines Dialoges sollte man nur bei einem Objekt DEFAULT setzen.

EXIT (0x0004) "form_do()" bricht ab, wenn der Benutzer das Objekt anklickt. Man beachte, daß die AES unter "Anklicken" das Niederdrücken und Loslassen einer Maustaste verstehen!

EDITABLE (0x0008) Das Objekt ist editierbar.

RBUTTON (0x0010) Das Objekt ist ein sogenannter Radioknopf (radio button). Dies ist eine Metapher aus der Zeit der Rundfunkgeräte, als bei Anwahl einer Station jeweils die anderen Tasten wieder herausgesprangen. Auf die AES übertragen bedeutet dies, daß von allen Objekten, bei denen dieses Flag gesetzt ist und die in der gleichen Hierarchieebene des Objektbaums stehen, nur eines ausgewählt sein kann. Wird ein anderes angeklickt, werden alle anderen automatisch deselektiert.

LASTOB (0x0020) Letztes Objekt im Objektbaum-Feld. Wird zumindest bei "form_do()" beim Durchsuchen des Baums nach DEFAULT- und EDIT-Objekten benötigt.

TOUCHEXIT (0x0040) Diese Bezeichnung ist ein wenig verwirrend und sollte besser "CLICKEXIT" lauten. Wird bei "form_do()" ein solches Objekt mit gedrückter Maustaste berührt (die Maustaste muß also nicht erst wieder losgelassen werden), bricht die Funktion ab (und liefert die Objektnummer des berührten Objekts).

HIDETREE (0x0080) Versteckt Objekt und alle Unterobjekte. Das Objekt wird also nicht nur nicht gezeichnet, sondern ist auch logisch nicht vorhanden (wichtig für "objc_find()"!). Editierbare Objekte kann man übrigens auch ausfüllen, wenn sie versteckt sind... (prima für Passwordeingaben...)

INDIRECT (0x0100) Zeigt an, daß in "ob_spec" ein Zeiger auf den eigentlichen "ob_spec" steht.

Alle weiteren Bits sind reserviert und sollten laut Digital Research auf Null gesetzt werden.

Objekt-Status (ob_state)

Während die "ob_flags" in erster Linie Informationen über die *Funktion* eines Objekts enthalten, beschreibt der Objekt-Status den *Zustand* des Objekts (oft ist auch die Konstante NORMAL (0) als "Normalzustand" vordefiniert).

SELECTED (0x0001)	Das Objekt wird hervorgehoben, indem es invertiert dargestellt wird. Dieses Bit ist übrigens immer bei dem Objekt gesetzt, das das Verlassen eines Dialoges verursacht hat, und sollte deshalb beizeiten wieder gelöscht werden.
CROSSED (0x0002)	Im Inneren des Objekts (nur für BOX-Objekte) wird ein Kreuz gezeichnet.
CHECKED (0x0004)	Das Objekt wird mit einem Häkchen gezeichnet (wird beispielsweise bei Einträgen in Drop-Down-Menüs verwendet).
DISABLED (0x0008)	Das Objekt wird grau schattiert gezeichnet (normalerweise bei Text). Angezeigt wird damit, daß es nicht selektiert werden kann.
OUTLINED (0x0010)	Um das Objekt wird eine zusätzliche Umrandung gezeichnet.
SHADOWED (0x0020)	Zusätzlich wird rechts unter dem Objekt ein Schatten gezeichnet (meistens bei BOX-Objekten).
WHITEBAK (0x0080)	Die Icon-Maske wird nicht mitgezeichnet. Kann die Ausgabe beispielsweise dann beschleunigen, wenn der Hintergrund sowieso schon weiß ist (nur für PC-GEM dokumentiert!)
DRAW3D (0x0040)	Nur für ICONBLK-Strukturen und auch nur ab PC-GEM 2.0 wird ein dreidimensionaler Effekt erzeugt, indem das Icon dreimal knapp schräg übereinander gezeichnet wird.

Objektfarben

Folgende Objektfarben sind vordefiniert (das hängt natürlich von der gewählten Bildauflösung und den Einstellungen des Benutzers ab):

WHITE (0)	Weiß
BLACK (1)	Schwarz
RED (2)	Rot
GREEN (3)	Grün
BLUE (4)	Blau
CYAN (5)	Cyan
YELLOW (6)	Gelb
MAGENTA (7)	Magenta

LWHITE (8)	Hellgrau
LBLACK (9)	Dunkelgrau
LRED (10)	Hellrot
LGREEN (11)	Hellgrün
LBLUE (12)	Hellblau
LCYAN (13)	Hellcyan
LYELLOW (14)	Hellgelb
LMAGENTA (15)	Hellmagenta

Die Textinformations-Struktur (TEDINFO)

Ein TEDINFO hat folgende Struktur:

```
typedef struct
{
     char *te_ptext;      /* Zeiger auf Text */
     char *te_ptmplt;     /* Zeiger auf Textmaske */
     char *te_pvalid;     /* Zeiger auf Texttypmaske */
     WORD te_font;        /* Zeichensatz */
     WORD te_resvd1;      /* reserviert */
     WORD te_just;        /* Justifikation */
     WORD te_color;       /* Farbe des betreffenden Rechtecks */
     WORD te_resvd2;      /* reserviert */
     WORD te_thickness;   /* Rahmen */
     WORD te_txtlen;      /* Länge des Textes */
     WORD te_tmplen;      /* Länge der Textmaske */
} TEDINFO;
```

Diese Datenstruktur wird von allen Objekten benutzt, die die Eingabe von Text zulassen.

Die einzelnen Strukturelemente:

te_ptext — Zeiger auf die eigentliche Zeichenkette (die durch den Editiervorgang auch verändert wird). Ist das erste Zeichen ein "@" ("Schneckennudel" oder "Klammeraffe"), werden alle folgenden Zeichen als Platzhalter angesehen, und der zunächst ausgegebene Text besteht aus Leerzeichen ("@2345678" liefert beispielsweise einen acht Leerzeichen langen String, der Klammeraffe wird also mitgezählt). Folge: das Zeichen "@" kann niemals am Anfang eines Edit-Felds stehen! Dieser String muß genauso lang sein, wie in "te_ptmplt" gültige Stellen vorhanden sind.

te_ptmplt Dieser String enthält nicht nur den Teil der Textinformation, der immer gleich ist, sondern auch eine Maske für die Zeichen, die verändert werden dürfen. Dafür setzt man den Unterstrich "_" ein, der allerdings beim Editieren der Texte mit einem RCS meist durch die Tilde "~" ersetzt wird.

te_pvalid String, der für jedes Zeichen in "te_ptext" eine Zeichenkonstante enthält, die über die Gültigkeit verschiedener Zeichen an dieser Stringposition Auskunft gibt:

"9": nur Ziffern
"A": nur Großbuchstaben und Leerzeichen
"a": nur Buchstaben und Leerzeichen
"N": Großbuchstaben, Ziffern und Leerzeichen
"n": Buchstaben, Ziffern und Leerzeichen
"F": alle Zeichen, die zu einem Dateinamen gehören
"P": alle Zeichen, die zu einem Pfadnamen gehören
"p": wie "P", allerdings ohne "?" und "*"
"X": alle Zeichen

Bemerkungen
Unter TOS 1.00 führt die Eingabe eines Unterstrichs in ein nur für Ziffern zugelassenes Edit-Feld zu einem Programmabsturz!

Die von "F", "P" und "p" zugelassenen Zeichen decken sich leider nicht mit der GEMDOS-Definition eines legalen Datei- bzw. Pfadnamens.

te_pfont IBM (3): normaler Zeichensatz (verschieden je nach Bildauflösung)
SMALL (5): 6*6-Systemzeichensatz

Bemerkungen
Alle bekannten AES-Versionen kommen bei "objc_edit()" ausschließlich mit acht Pixel breiten Zeichensätzen zurecht!

te_just TE_LEFT (0): linksbündig
TE_RIGHT (1): rechtsbündig
TE_CNTR (2): zentriert

te_color Für die Farbe des begrenzenden Rechtecks sind die Bits folgendermaßen belegt:

Bit 12..15: Rahmenfarbe (0..15)
Bit 8..11: Textfarbe (0..15)

Bit 7: Text transparent (0) oder deckend (1)

Bit 4..6:

IP_HOLLOW (0):	hohl
IP_1PATT (1):	ansteigende Intensität
IP_2PATT (2)	
IP_3PATT (3)	
IP_4PATT (4)	
IP_5PATT (5)	
IP_6PATT (6)	
IP_SOLID (7):	solide Fläche

Bit 0..3: Innenfarbe (0..15)

te_thickness Für den Rahmen sind folgende Werte gültig:

0:	Rahmendicke 0 (kein Rahmen)
1 bis 128:	Der Rand liegt 1 bis 128 Pixel im Inneren des Objekts.
–1 bis –127:	Der Rand liegt 1 bis 127 Pixel außerhalb des Objekts.

Zur Erläuterung der etwas komplizierten TEDINFO-Struktur ein Beispiel:

```
te_ptext:    "301164"
te_pvalid:   "999999"
te_ptmplt:   "Bitte Datum angeben: __.__.19__"
```

Zunächst fügen die AES den Inhalt von "te_ptext" in die mit Unterstrichen markierten Positionen von "te_ptmplt" ein:

```
"Bitte Datum angeben: 30.11.1964"
```

Die Einfügemarke steht zunächst hinter dem letzten Zeichen von "te_ptext", also in diesem Fall am Ende der Zeichenkette. Der Benutzer kann nun neben den Zifferntasten (wegen "te_pvalid") innerhalb des Textes mit BACKSPACE, DELETE, den Pfeiltasten und ESC (löscht die Eingabe) editieren. Nehmen wir als Eingabe "<ESC>231265" an.

Auf dem Bildschirm sieht es dann folgendermaßen aus:

```
"Bitte Datum angeben: 23.12.1965"
```

Wird nun der Dialog abgebrochen, dann enthält "te_ptext" "231265" – also genau die Eingabe des Benutzers –, bereinigt von allen festen Zeichen.

Die Icon-Struktur (ICONBLK)

Was ein Icon ist, braucht hier sicherlich nicht mehr erklärt zu werden. Wie wird es jedoch im Speicher dargestellt?

```
typedef struct
{
     UWORD     *ib_pmask;     /* Zeiger auf Icon-Maske */
     UWORD     *ib_pdata;     /* Zeiger auf Icon-Daten */
     char      *ib_ptext;     /* Zeiger auf Icon-Text */
     UWORD     ib_char;       /* Zeichen, das im Icon erscheinen
                                 soll, und Vorder- und Hintergrund-
                                 farbe des Icons */
     UWORD     ib_xchar;      /* dessen X-Position */
     UWORD     ib_ychar;      /* dessen Y-Position */
     UWORD     ib_xicon;      /* X-Position des Icons */
     UWORD     ib_yicon;      /* Y-Position des Icons */
     UWORD     ib_wicon;      /* Breite des Icons */
     UWORD     ib_hicon;      /* Höhe des Icons */
     WORD      ib_xtext;      /* X-Position der Beschriftung */
     WORD      ib_ytext;      /* Y-Position der Beschriftung */
     UWORD     ib_wtext;      /* Textbreite in Pixel */
     UWORD     ib_htext;      /* Texthöhe in Pixel */
     UWORD     ib_resvd;      /* reserviert */
} ICONBLK;
```

Die einzelnen Elemente der Struktur:

ib_pmask — Ein Zeiger auf ein Feld von 16-Bit-Werten, in denen das Bit-Image der Icon-Maske abgelegt ist. Die Icon-Maske legt fest, an welchen Stellen das Icon überhaupt gezeichnet werden soll und welche Pixel "transparent" bleiben sollen. Erzielt wird dieser Effekt dadurch, daß der Iconhintergrund zunächst mit den Bits der Maske "undiert" und dann mit den Icondaten "oderiert" wird.

ib_pdata — Zeiger auf das eigentliche Bild des Icons

ip_ptext — Zeiger auf den Icontext

ib_char
- Bit 15..12: Vordergrundfarbe des Icons
- Bit 11..8: Hintergrundfarbe des Icons
- Bit 7..0: Das Zeichen, das im Icon erscheinen soll (wie der Laufwerkbuchstabe in den Desktop-Icons)

ib_xchar,
ib_ychar — Position des Zeichens innerhalb des Icons

ib_xicon,
ib_yicon — Position des Icons auf dem Bildschirm. Bei Icons, die sich innerhalb eines Objektbaumes befinden, handelt es sich um die relative Position zum Mutterobjekt.

ib_wicon,
ib_hicon — Breite (muß ein Vielfaches von 16 sein) und Höhe des Icons

ib_xtext,
ib_ytext — Position der Textzeile innerhalb des Icons

ib_wtext,
ib_htext — Breite und Höhe des Icon-Textes

ib_resv — Unbenutzt, wird allerdings bei den meisten RCS-Programmen beim Schreiben in die Resourcedatei aufgenommen.

Die Bit-Image-Struktur (BITBLK)

Ein Bit-Image kann man immer dort benutzen, wo ein nicht anwählbares Icon stehen könnte. Der entscheidende Unterschied ist nämlich, daß es zu einem Bit Image keine Maske gibt.

Die Struktur:

```
typedef struct
{
     UWORD     *bi_pdata;     /* Zeiger auf Image */
     UWORD     bi_wb;         /* Breite in Bytes */
     UWORD     bi_hl;         /* Höhe in Pixelzeilen */
     WORD      bi_x;          /* X-Position */
     WORD      bi_y;          /* Y-Position */
     WORD      bi_color;      /* Farbe */
} BITBLK;
```

Die einzelnen Strukturelemente:

bi_pdata — Zeiger auf Wort-Feld, das die Bilddaten enthält (siehe unter ib_pdata)

bi_wb	Breite des Images in Bytes (muß durch 2 teilbar sein)
bi_hl	Höhe des Images in Zeilen
bi_x,bi_y	Position des Images
bi_color	AES-Farbcode für das Image

Die Application-Block-Struktur (USERBLK)

Schon mal über "fehlende" Objektformen geärgert? Der Objekttyp "G_USERDEF" erlaubt die Definition beliebiger neuer Objektarten. "ob_spec" zeigt dabei auf eine USERBLK-Struktur:

```
typedef struct
{
     WORD (*ub_code)(PARMBLK *);    /* Zeiger auf eigene Funktion */
     LONG ub_parm;                  /* Ein optionaler Parameter */
} USERBLK;
```

Dabei ist "ub_code" ein Zeiger auf eine eigene Funktion, die die Darstellung des Objekts voll übernimmt. Als Parameter erhält sie einen Zeiger auf eine PARMBLK-Struktur:

Die Parameter-Block-Struktur (PARMBLK)

```
typedef struct
{
     OBJECT    *pb_tree;       /* Zeiger auf Objektbaum */
     WORD      pb_obj;         /* Objektnummer */
     WORD      pb_prevstate;   /* Vorheriger Status */
     WORD      pb_currstate;   /* Neuer Status */
     WORD      pb_x;           /* X-Position des Objekts */
     WORD      pb_y;           /* Y-Position des Objekts */
     WORD      pb_w;           /* Breite */
     WORD      pb_h;           /* Höhe */
     WORD      pb_xc;     /* X-Position des Begrenzungsrechtecks */
     WORD      pb_yc;     /* Y-Position des Begrenzungsrechtecks */
     WORD      pb_wc;     /* Breite des Begrenzungsrechtecks */
     WORD      pb_hc;          /* Höhe des Begrenzungsrechtecks */
     LONG      pb_parm;        /* Parameter aus USERBLK */
} PARMBLK;
```

In dieser Struktur werden der in der USERBLK-Struktur angegebenen Funktion eine Fülle von Informationen über das Objekt geliefert:

pb_tree	Anfangsadresse des betreffenden Objektbaums
pb_obj	Objektnummer des betreffenden Objekts
pb_prevstate	Status des Objekts vor der aktuellen Änderung
pb_currstate	Neuer Status des Objekts (nur wenn "pb_currstate" und "pb_prevstate" identisch sind, muß das Objekt neu gezeichnet werden. Anderenfalls reicht ein "Update" des Objektstatus).
pb_x,pb_y, pb_w,pb_h	Position und Maße des Objekts in absoluten Bildschirmkoordinaten
pb_xc,pb_yc, pb_wc,pb_hy	Position und Größe des aktuellen (bei "objc_draw()" angegebenen) Begrenzungsrechtecks (clipping rectangle). Wenn kein Clipping eingeschaltet ist (zum Beispiel bei Objekten in Drop-Down-Menüs), dann sind alle Werte Null.
pb_parm	Der optionale, in der USERBLK-Struktur angegebene Parameter

Zu beachten sind folgende Punkte:

- Die eigene Funktion muß im Datenregister D0 den AES zurückliefern, welche Aspekte des Objektstatus noch aktualisiert werden müssen. Damit ist es nicht unbedingt nötig, in der eigenen Ausgabefunktion den Code zum Invertieren des Objekts auszuprogrammieren.

  Im allgemeinen wird man einige Bits des Objektstatus selbst bearbeiten wollen und andere den AES überlassen (siehe dazu das Beispiellisting in "Richtlinien zur Benutzerführung und Programmierung").

- Die Funktion erhält den PARMBLK-Zeiger auf dem Stack und muß daher unter Turbo-C als "cdecl" deklariert sein.

- Ein vollständiges Neuzeichnen des Objekts ist nur dann nötig, wenn "pb_prevstate" und "pb_currstate" gleich sind. Anderenfalls hat sich nur der Objektstatus geändert (zum Beispiel durch Anklicken).

- Die eigene Funktion wird de facto als Unterprogramm der AES ausgeführt. Daher sollte man in Hinsicht auf Stackbenutzung vorsichtig sein. Außerdem darf man natürlich keine weiteren AES-Aufrufe machen (die AES sind *nicht* re-entrant!). Aufrufe der VDI-Eingabefunktionen hingegen sind an *dieser* Stelle erlaubt!

- "pb_parm" dient dazu, der eigenen Funktion weitere Informationen – wie etwa einen Zeiger auf einen String – mit auf den Weg zu geben (vergleichbar mit dem "ob_spec" in der Objekt-Struktur, der ja bei "G_USERDEF"-Objekten bereits durch den USERBLK-Zeiger belegt ist).

- Man sollte sich nie zu weit von der ursprünglichen Optik von GEM entfernen. Abgerundete Rechtecke oder kursive Texte passen sicherlich *nicht* in das normale Erscheinungsbild einer GEM-Applikation!

Die Formular-Bibliothek

Einleitung

Die AES-Formular-Bibliothek nimmt sich der Verwaltung von Dialogboxen (also der Computerform von Formularen) an. Sie bedient sich dabei der Objekt-Bibliothek, die für die Darstellung der einzelnen Objekte und die Verwaltung von Tasteneingaben in Edit-Felder verantwortlich zeichnet.

Direkt zur Verwaltung der Eingaben in Dialogboxen ist "form_do()" bestimmt. "form_do" wiederum bedient sich dazu der Event-Funktionen (um Mausbewegungen und Tastendrücke zu empfangen), der Objekt-Funktionen (zum Beispiel für die Edit-Felder) sowie der Formular-Funktionen "form_keybd()" und "form_button()".

Details können Sie dem weiter unten angegebenen Quelltext von "form_do()" entnehmen. Für das eigentliche *Zeichnen* der Dialogbox ist nicht die Formular-Bibliothek, sondern "objc_draw()" zuständig.

"form_dial()" und "form_center()" sind spezielle Hilfsroutinen für Dialogboxen, die zur Vor- und Nachbereitung benutzt werden können. "form_dial()" sorgt für die "Reservierung" eines Bildschirmausschnitts (zur Zeit eine "leere" Operation, kann sich aber durchaus künftig ändern), das Zeichnen von wachsenden und schrumpfenden Rechtecken, um bei bestimmten Operationen mehr Feedback zu bieten, sowie die "Freigabe" des belegten Bildschirmbereichs (was zur Zeit Redraw-Nachrichten an die betroffenen Fenster auslöst). Mit "form_center()" wird die Dialogbox auf dem Arbeitsbereich des Hintergrundfensters (Fenster 0) zentriert.

"form_alert()" und "form_error()" schließlich sind "Frontends", die auf möglichst einfache Art und Weise Alarmboxen erzeugen, diese anzeigen und auch für die Wiederherstellung des belegten Bildausschnitts sorgen.

Nähere Informationen darüber, auf welche Art und Weise man Dialogboxen entwerfen und programmieren sollte, entnehmen Sie bitte dem Kapitel "Richtlinien zur Benutzerführung und Programmierung".

Alarm-Boxen

Alarm-Boxen sind eine spezielle Form von Dialogen, bei denen man auf fünf Textzeilen zu maximal 30 Zeichen, ein vordefiniertes Bildsymbol und drei Knöpfe zu maximal zehn Zeichen begrenzt ist. Für die Iconnummern gilt:

0: Kein Icon

1: Ausrufezeichen (Gefahr!)

2: Fragezeichen (Nachfrage vor möglicherweise gefährlichen Aktionen)

3: Stopschild (nun ist es zu spät – der Fehler ist bereits aufgetreten, und der Benutzer muß darüber informiert werden)

Die Daten für eine Alarm-Box werden in einem String folgenden Formats übergeben:

```
"[ >Icon-Nummer< ][ >Text< ][ >Knöpfe< ]"
```

Dabei ist >Icon-Nummer< eine Zahl zwischen 0 und 3 (als ASCII-Zeichen), >Text< eine oder mehrere Textzeilen (voneinander durch "|" getrennt) und >Knöpfe< einer oder mehrere Knopfnamen (ebenso voneinander getrennt).

Ein Beispiel:

```
"[3][Die Datei 'PBUCH.DOC' konnte|nicht geschrieben werden!][ OK ]"
```

Man beachte die Leerzeichen links und rechts von "OK": Sie dienen dazu, dem Knopf eine sinnvolle Mindestbreite zu verpassen.

Eine spezielle Spielart von Alarm-Boxen sind die Fehler-Boxen, die man mittels "form_error()" erzeugen kann. Bei ihnen ist zu beachten, daß

- der Parameter aus Gründen der Kompatibilität zu PC-GEM eine MS-DOS-Fehlernummer sein muß
- daß die Texte per definitionem in der gleichen Landessprache wie das benutzte TOS erscheinen und daß
- es nicht für alle GEMDOS-Fehlernummern einen passenden Text gibt.

Ihr Einsatz bietet sich daher nur bei kleineren Programmen, die keiner weiteren Internationalisierung bedürfen, an.

form_keybd() und form_button()

Diese beiden Funktionen sind eine Sache für sich, da DR sie wissentlich oder unwissentlich in der ursprünglichen GEM-Dokumentation weggelassen hat. Allerdings sind sie sehr wohl vorhanden und funktionieren auch so, wie sie sollen.

Um sie für das DR-Entwicklungspaket zu installieren, muß man folgendermaßen vorgehen: Die betreffenden Funktionen definiert man genau so, wie in der AES-Referenz angegeben. Zusätzlich setzt man vor den ersten Aufruf im Programm die Zeile

```
ctrl_cnts[135]=3; ctrl_cnts[136]=3; ctrl_cnts[137]=1;
ctrl_cnts[138]=2; ctrl_cnts[139]=2; ctrl_cnts[140]=1;
```

Statt dessen kann man auch mit einem Debugger die entsprechende Stelle in der Datei "apstart.o" ändern. Bei moderneren Entwicklungssystemen sind diese Funktionen meistens schon vorhanden.

Beide Funktionen werden intern von "form_do()" zur Bearbeitung von Formularen benutzt. Dabei kann man mit "form_button()" Mausknopf-Tastendrücke simulieren und mit "form_keybd()" die Gültigkeit eines eingegebenen Zeichens an einer bestimmten Position eines editierbaren Textfeldes feststellen.

Wie das genau vor sich geht, kann man am besten anhand des Quelltextes von "form_do()" verstehen:

```
#define FMD_BACKWARD  -1
#define FMD_FORWARD   -2
#define FMD_DEFLT     -3

#define TRUE 1
```

```
/* Objekt bestimmten Typs suchen */
WORD find_object (OBJECT *tree, WORD start_object, WORD which)
{
     WORD      object, flag, theflag, increment;

     object = 0;
     flag = EDITABLE;
     increment = 1;

     switch (which)
     {
          case FMD_BACKWARD:
               increment = -1;          /* kein Break! */
          case FMD_FORWARD:
               object = start_object + increment;
               break;
          case FMD_DEFLT:
               flag = DEFAULT;
               break;
     }

     while (object >= 0)
     {
          theflag = tree[object].ob_flags;

          if (theflag & flag)
               return (object);
          if (theflag & LASTOB)
               object = -1;             /* Abbruch */
          else
               object += increment;
     }

     return start_object;
}

/* Cursor auf Startfeld */
WORD ini_field (OBJECT *tree, WORD start_field)
{
     if (start_field == 0)
          start_field = find_object (tree, 0, FMD_FORWARD);
```

```
    return start_field;
}

WORD form_do (OBJECT *tree, WORD start_field)
{
    WORD edit_object, next_object, which, cont;
    WORD idx, mx, my, mb, ks, kr, br;

    wind_update (BEG_UPDATE);

    next_object = ini_field (tree, start_field);
    edit_object = 0;

    cont = TRUE;

    while (cont)
    {
        /* ggfs. Cursor positionieren */

        if ((next_object != 0) && (edit_object != next_object))
        {
            edit_object = next_object;
            next_object = 0;
            objc_edit (tree, edit_object, 0, &idx, ED_INIT);
        }
        /* Maus- oder Tastatur-Event erwarten */

        which = evnt_multi (MU_KEYBD|MU_BUTTON, 0x02, 0x01,
            0x01,  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0x0L,
            0, 0, &mx, &my, &mb, &ks, &kr, &br);

        if (which & MU_KEYBD)
        {
            cont = form_keybd (tree, edit_object, next_object,
                kr, &next_object, &kr);

            /* kr != 0 -> weiterverarbeiten! */

            if (kr)
              objc_edit (tree, edit_object, kr, &idx, ED_CHAR);
        }
```

```
        if (which & MU_BUTTON)
        {
            next_object = objc_find (tree, 0, MAX_DEPTH, mx, my);

            if (next_object == -1)
            {
                Bconout (2, 7);      /* Ping! */
                next_object = 0;
            }
             else
                cont = form_button (tree, next_object, br,
                       &next_object);
        }

        /* ggf. Cursor ausschalten */

        if (!cont || ((next_object != 0) &&
             (next_object != edit_object)))
        {
             objc_edit (tree, edit_object, 0, &idx, ED_END);
        }
    }

    wind_update (END_UPDATE);

    return next_object;
}
```

Die Grafik-Bibliothek

Diese Sammlung von Funktionen ist etwas zusammengewürfelt und bietet folgendes:

- Darstellung von Umrandungen in jeder Variante (schrumpfende, sich bewegende...). In der PC-Version von GEM sind diese Funktionen dem in der Einleitung erwähnten Rechtsstreit zwischen Digital Research und Apple zum Opfer gefallen.

- Änderung des Mauszeigers

- Abfrage des aktuellen VDI-Handles und anderer Informationen über die von den AES benutzte VDI-Workstation

- Abfrage von Position und Zustand der Maus und der "Umschalt-Tasten" Shift, Control und Alternate.

Die Scrap-Bibliothek

Viele Atari-Anwender haben sich wahrscheinlich schon einmal gefragt, warum bei GEM in keiner Weise die beim Apple Macintosh so beliebte "Zwischenablage" verwirklicht ist. Die Antwort könnte glücklich und traurig zugleich stimmen: Die dafür notwendigen Funktionen sind nämlich allesamt vorhanden – allerdings werden sie nur von einer kleinen Zahl von Programmen richtig benutzt.

Einige der positiven Ausnahmen sind "SciGraph" (SciLab GmbH), "Clipboard" (ein Public-Domain-Programm von Dieter und Jürgen Geiß), "Gemini" (der alternative Desktop) und – mit Einschränkungen – "Wordplus".

Die AES-Zwischenablage ist nichts anderes als ein ganz normales Verzeichnis – normalerweise auf einer Festplatte. Die Funktionen "scrp_read()" und "scrp_write()" dienen lediglich dazu, den Namen dieses Verzeichnisses zu setzen bzw. abzufragen. Leider wird im Atari-AES kein Verzeichnis vorinitialisiert. Daher sollte man in eigenen Programmen etwa folgendermaßen vorgehen:

1. Verzeichnispfad mit "scrp_read()" abfragen. Wenn das Ergebnis keine leere Zeichenkette ist, dann ist man schon fertig – ein anderes Programm hat bereits den Pfad gesetzt. Normalerweise ist der Pfad mit "\"- abgeschlossen, sonst muß man es selbst anhängen.

2. Entscheiden, wo das Clipboard angelegt werden soll. Vorschlag von Atari: Falls Laufwerk C: existiert ("Dsetdrv()" benutzen!), das Verzeichnis "C:\CLIPBRD\" benutzen. Andernfalls "A:\CLIPBRD\" nehmen oder den Benutzer fragen. Bei notwendigem Anlegen des Verzeichnisses sollte man nicht vergessen, mögliche Fehler korrekt auszuwerten (z. B. bei schreibgeschützten Medien).

   Anderenfalls gilt es, einen anderen Ort zu finden. Normalerweise versucht man es zunächst mit dem Verzeichnis "CLIPBRD" im Wurzelverzeichnis des Bootlaufwerks (siehe Systemvariable "_bootdev"). Der vollständige Pfad wäre dann also "C:\CLIPBRD\".

3. Sicherstellen, daß das ausgewählte Verzeichnis auch tatsächlich funktioniert und beschreibbar ist.

Ebenso wie beim Macintosh-Clipboard kann man auch unter GEM immer nur *ein* Datenobjekt auf einmal ablegen.

Dabei sind allerdings mehrere Formate möglich. Um Daten im Clipboard abzulegen, muß man also:

1. *alle* Clipboard-Dateien, die auf die Maske "SCRAP.*" passen, löschen.

2. die abzulegenden Daten in einem oder mehreren Formaten sichern. Der Dateiname ist dabei *immer* "SCRAP", die Namenserweiterung hängt vom gewählten Format ab. Nach Möglichkeit sollte *immer* eines der folgenden Standardformate dabei sein:

   TXT ASCII-Textdatei, jede Zeile mit CR/LF abgeschlossen

   GEM GEM-Metadatei (siehe VDI-Dokumentation)

   IMG GEM-Rasterbild (siehe Anhang)

   Zusätzlich kann man eines oder mehrere zusätzliche Formate unterstützen. Der Empfänger hat dann die Möglichkeit, das Format mit den "meisten" Informationen zu benutzen. Weitere gebräuchliche Formate sind:

   ASC ASCII-Textdatei, jeder Absatz mit CR/LF abgeschlossen

   CSV ASCII-Datei mit durch Kommata getrennten Zahlen

   DIF Export-Datei von Tabellenkalkulationen

   1WP Wordplus-Format

   RTF Microsoft Rich Text Format

   EPS Encapsulated PostScript

   TEX TeX

   CVG Calamus Vektorgrafik-Format

Das empfangende Programm sollte zunächst überprüfen, welche der vorhandenen Dateien die meisten Informationen enthält, und dann diese Datei nehmen.

Nochmals zur Erinnerung: Jede der Dateien enthält prinzipiell die gleichen Informationen, nur eben in einem anderen Format! Wordplus zum Beispiel importiert "SCRAP.TXT" nur dann, wenn "SCRAP.1WP" nicht gefunden werden konnte.

Beim Import von Textdaten aus dem Clipboard sollte man übrigens auch mit solchen Dateien zurechtkommen, bei denen das "Carriage Return" am Zeilenende fehlt und nur ein "Linefeed" als Zeilentrenner benutzt wird – viele UNIX-basierten Programme legen solche Textdateien an!

Wer schon einmal an einem Macintosh gearbeitet hat, wird auch das "Album"-Accessory kennen, das anscheinend *mehrere* Datensätze im Clipboard verwaltet. Auch das könnte man unter GEM leicht verwirklichen: Man braucht nur ein Accessory, das eine kleine "Datenbank" für Dateien in verschiedenen Formaten bereitstellt und Daten aus dem Clipboard importieren und in das Clipboard exportieren kann.

Für die aktuelle GEM-Realität ist das AES-Clipboard zwar brauchbar, aber teilweise nur schwierig zu benutzen. Die wichtigsten Einschränkungen sind:

- Es gibt keine Standardnachricht, mit der anderen Prozessen mitgeteilt werden könnte, daß sich der Inhalt des Clipboards geändert hat.

- Eine direkte Kommunikation wie beim "DDE" (Dynamic Data Exchange) unter Microsoft Windows ist nicht möglich.

Beide Probleme werden mit einiger Sicherheit in einer multitaskingfähigen GEM-Version beseitigt werden.

Die Dateiauswahl-Bibliothek

Die Dateiauswahlbox braucht kaum noch genau erklärt zu werden – jedem, der schon einmal eine GEM-Anwendung benutzt hat, ist sie bekannt. Im GEM 1.0 war sie noch ob ihrer Unzuverlässigkeit (Abstürze, Limit für die Anzahl der Dateien) und der unpraktischen Bedienung (Laufwerkswahl ausschließlich per Tastatur) sehr unbeliebt. Seit GEM 1.4 ist sie nun aber nicht nur relativ stabil, sondern auch komfortabel zu bedienen.

Dennoch seien noch einmal kurz die Vorteile aufgezählt:

- Der Anwender kann sehen, welche Dateien es gibt, und läuft nicht Gefahr, sich bei der Eingabe des Namens zu vertippen.

- Die Dateiauswahl funktioniert in allen Programmen auf gleiche Art und Weise.

- Residente Programme können den Betriebssystemaufruf abfangen und eine eigene, leistungsfähigere Routine installieren, die dann von allen Programmen benutzt wird.

- Minimaler Aufwand beim aufrufenden Programm.

- Das eigene Programm braucht sich nicht um die Details des verwendeten Dateisystems zu kümmern.

Doch es gibt auch Nachteile, die nicht verschwiegen werden sollen:

- Vor GEM 1.4 relativ fehleranfällig bei kritischen BIOS-Fehlern.

- Vor GEM 1.4 keine Möglichkeit, einen erklärenden Text in die Dialogbox aufzunehmen (siehe "fsel_exinput()").

- Es werden nicht alle für Dateinamen erlaubten Zeichen akzeptiert (siehe GEMDOS-Einleitung). Andererseits *können* einige Zeichen eingegeben werden, die eigentlich nicht in Dateinamen auftreten dürfen (wie zum Beispiel nationale Sonderzeichen).

- Es können nicht alle vorhandenen Laufwerke direkt angewählt werden (unter Meta-DOS kann es bis zu 26 Laufwerke geben).

- Eigene Verbesserungen bei der Dialogbehandlung (Tastaturbedienung, automatisches Sichern des Bildschirmhintergrunds) können nicht übertragen werden.

- In der Pfadangabe kann nur eine Suchmaske anstelle einer Liste von Suchmasken wie unter PC-GEM angegeben werden.

Wer es aus den oben genannten Gründen für nötig hält, statt dessen eine eigene Routine zu benutzen, sollte folgende Ratschläge beherzigen:

- Als Option die normale Systemfunktion anbieten.

- Fest eingefahrene Gewohnheiten der Benutzer nicht unnötig stören!

Die Fenster-Bibliothek

Einleitung

Mit Fenstern dürfte jeder Atari-Programmierer vertraut sein – schließlich kennt man sie aus Desktop und vielen anderen GEM-Applikationen. Doch bevor wir uns den technischen Details zuwenden, soll erst einmal die gar nicht so dumme Frage untersucht werden, wozu es eigent-

lich Fenster gibt – schließlich läßt sich nicht bestreiten, daß sie auch wertvolle Bildschirmfläche wegnehmen.

Ein wichtiger Grund ist die Multitaskingfähigkeit von GEM – auch wenn in den aktuellen Versionen nur ein Hauptprogramm und sechs Accessories möglich sind. Wie – wenn nicht mit Hilfe von mehreren, überlappenden Fenstern – sollte es möglich sein, die Bildschirmausgaben mehrerer AES-Prozesse zu koordinieren? Wer also in eigenen Programmen egoistisch unter Umgehung des Fenstersystems direkt auf den Bildschirmhintergrund ausgibt, nimmt den Anwendern die Möglichkeit, die Multitaskingmöglichkeiten der AES zu nutzen.

Auch die große Vielfalt von Bildschirmen (von 320 * 200 Punkten auf dem ST bis zu Auflösungen von über 1000 * 1000 Pixeln in 16 Millionen Farben) lassen die Direktausgabe auf dem Hintergrund wie einen schlechten Scherz erscheinen.

Niemand wird sich einen Großbildschirm kaufen, um in der Textverarbeitung 160 Zeichen in eine Textzeile zu zwängen. Nur mit verschiebbaren und in der Größe verstellbaren Fenstern kann sich ein Programm sinvoll an jedes Bildschirmformat anpassen.

Und schließlich ist es für GEM-Anwendungen Standard, daß sie mehrere Dokumente gleichzeitig bearbeiten können, schon um das Ausschneiden und Einfügen von Dokumentteilen zu ermöglichen. Wie anders als durch die Nutzung der GEM-Fenster sollte man dieses Ziel komfortabel erreichen?

Konzept

Um die AES-Fensterfunktionen korrekt und effizient benutzen zu können, muß man ihre Funktionsweise genau verstehen. Daher zuerst eine Liste von Funktionen, um die sich die AES *nicht* kümmern:

- Die Verwaltung von Koordinaten: Programme, die Bildschirmausgaben in Fenstern machen wollen, müssen Koordinaten im Fenster selbsttätig in absolute Bildschirmkoordinaten umrechnen.

- Das Puffern von Bildausschnitten: die AES-Fensterfunktionen bieten *keine* Möglichkeit, diese Aufgabe den Programmen abzunehmen. Statt dessen werden Nachrichten verschickt, die über die neu zu zeichnenden Bildausschnitte Auskunft geben.

Man tut also gut daran, sich die AES-Fensterbibliothek zunächst nur als eine Sammlung von Routinen vorzustellen, die sich mit der Verwaltung rechteckiger Bildausschnitte, der Fenster, befaßt.

Rechteckliste

Fenster können sich bekanntlich gegenseitig überlappen. Die AES nehmen sich dieses Problems an, in dem sie für jedes Fenster die sichtbaren Ausschnitte in mehrere Rechtecke aufteilen. Gemeinsam bilden sie die "Rechteckliste" eines Fensters, die ein Programm mit mehreren aufeinanderfolgenden Aufrufen der Fensterbibliothek erfragen kann.

Dabei beachte man, daß Fenster auch teilweise außerhalb des Bildschirms liegen können – man muß also erst die Schnittfläche der Rechtecke mit dem sichtbaren Bildschirm bilden.

Dazu eine Beispielfunktion:

```
/* GEM-Standardstruktur für Rechtecke */

typedef struct
{
     WORD g_x;      /* X-Position */
     WORD g_y;      /* Y-Position */
     WORD g_w;      /* Breite */
     WORD g_h;      /* Höhe */
} GRECT;

/* Schnittfläche zweier Rechtecke bilden. Der Returnwert ist genau
   dann Null, wenn sich die Rechtecke nicht überschneiden. Die
   zweite GRECT-Struktur wird mit den Koordinaten der Schnittfläche
   überschrieben */

WORD rc_intersect (GRECT *p1, GRECT *p2)
{
     WORD tx, ty, tw, th;

     tw = min (p2->gx + p2->gw, p1->gx + p1->gw);
     th = min (p2->gy + p2->gh, p1->gy + p1->gh);
     tx = max (p2->gx, p1->gx);
     ty = max (p2->gy, p1->gy);
     p2->gx = tx;
     p2->gy = ty;
     p2->gw = tw - tx;
     p2->gh = th - ty;
     return ((tw > tx) && (th > ty));
}
```

Die Rechtecklisten ändern sich natürlich jedesmal dann, wenn irgendetwas an den Fenstern verändert wird (Lage oder Größe, Reihenfolge, Anzahl etc.). Fatal wäre natürlich, wenn dies *während* der Abfrage der Rechteckliste und damit verbundenen Bildschirmausgaben stattfinden würde. Die AES bieten daher ein spezielles Protokoll an: Jedes Programm, das Rechtecklisten abrufen und dabei Bildschirmausgaben machen möchte, muß dies mit der Funktion "wind_update()" ankündigen. Damit haben die AES die Möglichkeit, bei Bedarf den aufrufenden Prozeß so lange zu blockieren, bis er gefahrlos weiterarbeiten kann.

Kontrollelemente

Damit sind natürlich noch nicht alle Aufgaben der Fensterfunktionen genannt – zu einem Fenster gehören nicht nur der "Arbeitsbereich" ("work area"), in den Programme ausgeben dürfen, sondern auch die "Kontrollelemente", wie Schließbox oder Slider. Für die Verwaltung dieser Fensterelemente ist allein der Screen Manager zuständig. Das Anwendungsprogramm gibt beim Anlegen des Fensters lediglich an, welche Elemente benutzt werden sollen.

Die Interaktion mit den Kontrollelementen wird vollständig vom Screen Manager übernommen. So ist er es, der beim Verändern der Fenstergröße die neuen Umrisse des Fensters anzeigt. Hat dann am Ende tatsächlich eine Veränderung stattgefunden, wird eine entsprechende Mitteilung an den Besitzer des betreffenden Fensters geschickt. Konsequenz ist, daß das Anwendungsprogramm die endgültige Entscheidung darüber trifft, ob und wie es die Wünsche des Benutzers erfüllt.

Diese Freiheit darf man freilich nicht mißbrauchen. Wenn man beispielsweise nicht auf eine WM_CLOSE-Mitteilung zu reagieren gedenkt, sollte man gar nicht erst ein Schließ-Feld erscheinen lassen! Völlig unakzeptabel ist es, Fenster-Kontrollelemente für andere Zwecke – etwa die Umschaltung zwischen zwei verschiedenen Betriebsarten – zu mißbrauchen.

Hier eine Liste der Kontrollelemente und der Reaktionen, die normalerweise damit verbunden sein sollten:

Titelzeile	Text, der im oberen Fensterrand als Beschriftung erscheint. Sollte laut Original-Dokumentation maximal 80 Zeichen umfassen. Normalerweise werden am Anfang und am Ende je ein Leerzeichen angefügt.
Infozeile	Zusätzliche Informationszeile, die normalerweise unterhalb der Titelzeile auftaucht und ebenso bis zu 80 Zeichen Text aufnehmen kann.
Schließfeld	Kleine Box zum Schließen des Fensters (normalerweise links oben in der Ecke). Ein Mausklick hierhin führt zu einer "WM_CLOSE"-Nachricht.

Volle Größe	"Fullbox" – kleine Box zur Umschaltung zwischen maximaler und aktueller Fenstergröße (normalerweise rechts oben in der Ecke). Ein Anklicken der "Fullbox" zieht eine "WM_FULLED"-Nachricht nach sich.
Bewegungsbalken	"Move bar" – nimmt den gleichen Platz wie die Titelzeile ein und erlaubt es dem Anwender, mit Hilfe einer Umrißlinie das Fenster neu zu positionieren. Resultat ist eine "WM_MOVED"-Mitteilung.
Größenfeld	Kleine Box rechts unten im Fenster, die eine Veränderung der Fenstergröße ermöglicht. Der Anwender sieht dabei nur einen Umriß mit der neuen Fenstergröße. Nach Abschluß der Operation wird eine "WM_SIZED"-Mitteilung verschickt.
Aufwärts-Pfeil	Kleine Box mit Aufwärtspfeil am rechten Fensterrand. Nach Anklicken erhält das Programm eine "WM_ARROWED"-Mitteilung.
Abwärts-Pfeil Rechts-Pfeil Links-Pfeil	Analog zum Aufwärts-Pfeil.
Vertikaler Balken	Dieses Kontrollelement liegt zwischen Aufwärts- und Abwärtspfeil und besteht aus einem Hintergrundobjekt und dem Slider. Die Größe des Sliders zeigt dabei an, wie sich der sichtbare Anteil zur Dokumentgröße verhält. Mausklicks über und unter den Slider führen zu "WM_ARROWED"-Mitteilungen, die als Aufforderung, um einen Fensterinhalt in die betreffende Richtung zu blättern, interpretiert werden sollten. Ein Verschieben des Sliders zieht eine "WM_VSLID"-Mitteilung nach sich.
Horizontaler Balken	Analog zum vertikalen Balken.

Achtung: Man darf niemals Annahmen über Größe, Lage und Position der Fensterelemente machen – es ist durchaus möglich, den AES einen anderen Fenstermanager unterzuschieben, bei dem die Fensterkontrollen ganz anders aussehen und auch an anderer Stelle liegen. Die Funktion "wind_calc()" erlaubt es, auf portable Art und Weise aus der Gesamtfläche den Arbeitsbereich (und umgekehrt) zu berechnen.

Ab Atari-GEM 3.0 kann man mit "wind_set()" jedem Kontrollelement eigene Farben und Füllmuster zuordnen – und zwar getrennt für aktive und inaktive Fenster. Die Einstellung kann entweder für ein spezielles oder als Default für alle neu erzeugten Fenster geschehen. Diese

Aufgabe wird normalerweise vom XCONTROL-Modul "Fensterfarben" übernommen. Leider ist es nicht möglich, die aktuellen Einstellungen abzufragen.

Das Desktop-Fenster

Eine besondere Rolle nimmt Fenster Null, das Desktop- oder Hintergrund-Fenster, ein. Es nimmt die gesamte Bildschirmfläche in Anspruch, ist immer geöffnet und kann auch nicht geschlossen werden. Der Arbeitsbereich ist die Fläche unter der Menüleiste. Nur in diesem Arbeitsbereich dürfen andere Programme Bildschirmausgaben machen oder eigene Fenster öffnen.

Normalerweise erscheint der Arbeitsbereich des Desktop-Fensters als solide grüne Fläche (bei Farbbetrieb) bzw. als graues Raster (auf monochromen Bildschirmen). Der Screen Manager kümmert sich völlig selbsttätig um den Redraw. Anwendungsprogramme können mittels "wind_set()" einen beliebigen anderen Objektbaum als Hintergrund verankern. Auch dann kümmert sich der Screen Manager um das fällige Neuzeichnen von Bildausschnitten. Ein bekanntes Beispiel ist das Desktop, das die Laufwerkssymbole als eigenständige Objekte im selbstinstallierten Objektbaum verwaltet. Obwohl diese Möglichkeit natürlich sehr verlockend ist, gibt es etliche Gründe, die *gegen* eine Benutzung des Desktop-Fensters sprechen:

- Unter Atari-GEM ist es *nicht* möglich, auf saubere Art und Weise Farbe und Füllmuster des Standardhintergrunds abzufragen. Die Folge ist die sehr häßliche Änderung der Hintergrundfarbe beim Starten von Programmen, die einen eigenen Hintergrund installieren.

- Auch unter einem multitaskingfähigen GEM kann es nur *einen* Bildschirmhintergrund geben. Der sollte dem Programm vorbehalten bleiben, das daraus den meisten Nutzen ziehen kann – und das ist das Desktop bzw. ein Programm, das als Desktop-Ersatz konzipiert ist (auf dem Macintosh sieht es übrigens genauso aus: Nur der "Finder" greift auf den Bildschirmhintergrund zu).

Fazit: Der Desktophintergrund sollte nach Möglichkeit in eigenen Programmen *nicht* benutzt werden.

Implementation

Alle von den AES für Fenster benutzten Datenstrukturen sind statisch, haben also eine fixe Länge. Das liegt nicht etwa daran, daß die Programmierer von Digital Research zu dumm für eine dynamische Verwaltung waren. Der Grund ist vielmehr darin zu suchen, daß ein dyna-

misches Nach-Anfordern von Speicher beim Beenden von Programmen zu einer häßlichen Zersplitterung des freien Speichers führen würde.

Die Folge ist, daß die Fensterzahl auf das kleinstmögliche Level festgesetzt worden ist: für jeden AES-Prozeß mindestens ein Fenster.

So kommt es zur Maximalzahl von acht Fenstern in der aktuellen AES-Version. Zu beachten ist auch noch, daß jedes zusätzliche Fenster die Maximallänge einer Rechteckliste erhöht – und auch die wird in einem festen Speicherbereich abgelegt.

Auch wenn man nur selten wirklich *mehr* als die vorhandenen Fenster benötigt, wäre es schön, könnte man *vor* Start der AES auf irgendeine Weise die Maximalzahl einstellen. Vielleicht kann sich Atari ja eines Tages mal mit dieser Idee anfreunden.

Viele GEM-Applikationen, so auch das Desktop vor Atari-GEM 3.0, schränken sich freiwillig auf vier Fenster ein, um für Accessories Fenster freizuhalten. Diese Vorsicht ist allerdings aus vielerlei Gründen nicht zu empfehlen:

- Künftige AES-Versionen werden definitiv mehr Fenster gleichzeitig öffnen können.

- Jedes Programm, das Fenster benutzt, muß mit einem Fehler bei "wind_create()" rechnen. Daher ist es nicht besonders sinnvoll, dem Anwender unnötige Einschränkungen aufzuerlegen.

Prinzipielle Vorgehensweise

Wenn man Fenster in eigenen Programmen benutzen will, sollte man in etwa folgendermaßen vorgehen:

1. Die Größe der verfügbaren Bildschirmfläche feststellen. Das macht man mit "wind_get()" ("wi_gfield": WF_WORKXYWH) für Fenster Null (das Desktop-Fenster).

2. Mit "wind_calc()" berechnet man sodann, wie groß der Arbeitsbereich des Fensters bei gegebenen Randkomponenten werden kann.

3. Man vergleicht die erhaltenen Werte mit der gewünschten Größe (und ärgert sich, wenn nicht genug Platz ist).

4. Die so festgelegte Größe des Arbeitsbereiches wird zusammen mit den Randkomponenten in "wind_calc()" eingespeist, welches daraus die Außenmaße des Fensters berechnet.

5. Das Fenster wird unter Angabe dieser Außenmaße und der Fensterkomponenten angemeldet ("wind_create()"). Dabei erhält man entweder eine Fehlermeldung oder die Kennung des neuen Fensters.

6. Das Fenster wird mit "wind_open()" auf den Bildschirm gebracht. Dies löst automatisch eine Redraw-Mitteilung über die Arbeitsfläche des Fensters aus.

7. Das Fenster wird vom Programm benutzt...

8. Mit "wind_close()" wird das Fenster vom Bildschirm gelöscht, kann aber immer noch mit "wind_open()" erneut geöffnet werden.

9. "wind_delete()" entfernt das Fenster endgültig aus dem Speicher und gibt insbesondere die Fensterkennung für andere Fenster frei.

Zum Redraw eines Fensterausschnitts verfährt man wie folgt:

```
/* Ausschnitt "area" in Fenster "wh" neuzeichnen. Basiert auf
   Beispielfunktion aus "Professional GEM" (Tim Oren) */

void redraw (WORD wh, GRECT *area)
{
     GRECT   box, full;

     graf_mouse (M_OFF, NULL);      /* Maus ausschalten */
     wind_update (BEG_UPDATE);      /* Rechteckliste sperren */

     /* Arbeitsbereich von Fenster Null abfragen */
     wind_get (0, WF_WORKXYWH, &full.g_x, &full.g_y, &full.g_h,
               &full.g_w);

     /* erstes Element der Rechteckliste erfragen */
     wind_get (wh, WF_FIRSTXYWH, &box.g_x, &box.g_y, &box.g_w,
               &box.g_h);

     /* solange gültiges Rechteck (Breite und Höhe ungleich Null)*/
     while (box.g_w && box.g_h)
     {
         if (rc_intersect (&full, &box))          /* sichtbar */
         {
              if (rc_intersect (area, &box))      /* Überlappung? */
```

```
                {
                    /* Clipping auf Bereich "box" setzen und
                     Fensterinhalt neu ausgeben */
                }
            }

            /* nächstes Element der Rechteckliste erfragen */

            wind_get (wh, WF_NEXTXYWH, &box.g_x, &box.g_y, &box.g_w,
                      &box.g_h);
        }

        wind_update (END_UPDATE);
        graf_mouse (M_ON, NULL);
}
```

Tips

Und hier noch ein paar weitere Hinweise zu den Fenster-Funktionen:

- Viele Programme erlauben es, beim Verlassen Position und Größe der geöffneten Fenster zu speichern. Diese Daten sollte man immer auflösungsunabhängig (also zum Beispiel relativ zu einem "normalisierten" Koordinatensystem von 32768 * 32768 Punkten) speichern. Anderenfalls kann es zu so unschönen Effekten wie bei "Wordplus" kommen (Fenster erscheinen innerhalb der Menüleiste oder außerhalb des sichtbaren Bildschirmbereichs).

- In einigen Programmen werden Fenster an der horizontalen Position "–1" geöffnet, um den Arbeitsbereich des Fensters an Position 0 beginnen zu lassen. Dabei verläßt man sich nicht nur auf die nicht-dokumentierte Tatsache, daß die AES negative Koordinaten korrekt verarbeiten, sondern macht auch eine unsichere Annahme über die Breite der Fensterränder.

  Hinzu kommt, daß nicht bei allen Bildschirmen der Bildrand schwarz ist, so daß man nur schwer Fenster und Bildrand auseinanderhalten kann.

- Der Screen Manager verschickt in mehreren Fällen unnötige Redraw-Mitteilungen, auf deren Eintreffen man sich jedoch nicht verlassen sollte. Wenn ein Fenster zum obersten Fenster wird, erhält es eine Redraw-Mitteilung über die gesamte Arbeitsfläche – auch über die Bereiche, die bereits sichtbar waren. Gleiches gilt für das Vergrößern eines Fensters.

Die Resource-Bibliothek

Einleitung

Die Resourcen eines Programms sind alle Daten, die normalerweise in der Resource-Datei ("*.RSC") gespeichert sind. Dazu gehören Informationen über Dialoge, Menüs, Icons usw. Die Speicherung in einer separaten Datei hat den Vorteil, daß man das Erscheinungsbild (insbesondere bei Texten die Sprache) ändern kann, ohne am Programm etwas modifizieren zu müssen.

Die Resource-Datei

Am Anfang einer solchen Datei steht stets folgende Struktur:

```
typedef struct
{
     UWORD rsh_vrsn;     /* null */
     UWORD rsh_object;   /* Position des Objekt-Feldes */
     UWORD rsh_tedinfo;  /* Position der TEDINFO-Strukturen */
     UWORD rsh_iconblk;  /* Position der ICONBLK-Strukturen */
     UWORD rsh_bitblk;   /* Position der BITBLK-Strukturen */
     UWORD rsh_frstr;    /* Position der freien Strings */
     UWORD rsh_string;   /* unbenutzt */
     UWORD rsh_imdata;   /* Position der Image-Daten */
     UWORD rsh_frimg;    /* Position der freien Images */
     UWORD rsh_trindex;  /* Position der Objektbaumtabelle */
     UWORD rsh_nobs;     /* Gesamtzahl der Objekte */
     UWORD rsh_ntree;    /* Gesamtzahl der Objektbäume */
     UWORD rsh_nted;     /* Gesamtzahl der TEDINFO-Strukturen */
     UWORD rsh_nib;      /* Gesamtzahl der ICONBLK-Strukturen */
     UWORD rsh_nbb;      /* Gesamtzahl der BITBLK-Strukturen */
     UWORD rsh_nstring;  /* Gesamtzahl der Strings */
     UWORD rsh_nimages;  /* Gesamtzahl der Images */
     UWORD rsh_rssize;   /* Gesamtlänge der RSC-Datei */
} RSHDR;
```

Alle Positionsangaben sind relativ zum Dateianfang zu verstehen. Noch ein Wort zu den "freien Strings". Zu diesen gehören nicht nur die Zeichenketten, in denen sich die Daten für Alarmboxen befinden, sondern auch alle anderen Strings, die ein Programm zu seiner Arbeit

benutzt. Ein Beispiel dafür wäre der Dateiname einer einzulesenden Datei oder ein Eintrag, der mit "menu_text()" in einem Menü vorgenommen wird.

Diesem Dateikopf folgen die eigentlichen Resource-Daten. Man beachte dabei, daß eine Resource-Datei aufgrund der Verwendung von 16-Bit-Werten als Zeiger nur eine Gesamtgröße von maximal 64 KByte erreichen kann. Dateien dieses Formats werden von allen RCS-Programmen ("Resource Construction Set") abgespeichert.

Resourcen im Programm

Aus mancherlei Gründen kann es sinnvoll sein, Resource-Daten direkt in das Programm aufzunehmen. Einige Vorteile sind:

- In Accessories ist das Nachladen von RSC-Dateien problematisch (Finden der Datei, Anfordern von Speicher).
- In XCONTROL-Moduln darf "rsrc_load()" gar nicht aufgerufen werden (mehr dazu im Kapitel über "XCONTROL").
- Resource-Dateien dürfen nicht länger als 64 KByte werden.

Wenn man daher die Resource-Daten direkt einbinden möchte, muß man folgende Probleme lösen:

1. Die Resource-Strukturen vom Resource-Construction-Programm in ein Format bringen lassen, das von der benutzten Programmiersprache verstanden wird. Das ist meist nur für C-Compiler möglich.
2. In den RSC-Daten stehen anstelle "echter" Zeiger noch die Indizes der jeweiligen Feldelemente. Diese Verweise müssen vom Programm in richtige Zeiger umgerechnet werden. Das RCS "Interface" erlaubt es, schon passend initialisierte C-Datenstrukturen zu erzeugen.
3. Schließlich müssen die zeichenorientierten Koordinatenangaben in echte Pixelkoordinaten konvertiert werden. Dazu kann die Funktion "rsrc_obfix" benutzt werden.

Die Shell-Bibliothek

Die Shell-Bibliothek ist sozusagen die Schaltzentrale im Innern der AES. Sie sorgt für das Starten der einzelnen GEM-Prozesse und versorgt sie mit Informationen über ihre Arbeitsumgebung. Daneben stellt sie dem Desktop mehrere Hilfsfunktionen zur Verfügung.

Beginnen wir mit "shel_read()": Mit dieser Funktion kann ein Programm auf betriebssystemunabhängige Art und Weise Informationen über den eigenen Dateinamen und die übergebenen Parameter erfragen. Leider liefert sie nur dann korrekte Resultate, wenn das betreffende Programm tatsächlich mit Hilfe der Shell-Bibliothek gestartet worden ist.

Daher ist es meistens sinnvoller, diese Informationen aus der Basepage und dem Environment (ARGV-Verfahren) zu extrahieren.

"shel_write()" erlaubt es, einen neuen AES-Prozeß zu starten. Dies allerdings erst nach Beendigung des eigenen Prozesses und auch nur dann, wenn der Aufrufer zum Programmstart tatsächlich die AES-Shellfunktionen benutzt hat. PC-GEM kennt bei dieser Funktion spezielle Parameter zum Starten von Programmen im Textmodus (ohne VDI und AES). Künftige AES-Versionen werden vermutlich zusätzliche AES-Prozesse starten können, ohne dazu erst den aktuellen Prozeß beenden zu müssen.

Ebenso wie "shel_read()" die betriebssystemabhängige Abfrage der Kommandozeile ersparen soll, ist "shel_envrn()" zur portablen Abfrage von Environment-Variablen gedacht. Man beachte, daß "shel_envrn()" stets auf das Environment zugreift, das beim Starten der AES aktuell war. Hinzu kommen einige Probleme mit der "PATH"-Variablen (siehe unter "shel_envrn()"), so daß eine sinnvolle Nutzung dieser Funktion kaum möglich ist. Statt dessen sollte man das normale GEMDOS-Environment abfragen!

Nützlicher ist da schon "shel_find()", das zum Suchen von Dateien benutzt wird. Auch "rsrc_load()" nimmt es in Anspruch, um die angegebene RSC-Datei zu finden. Speziell seit GEM-Version 1.4 ist "shel_find()" von großer Bedeutung, da das Desktop beim Starten einer Applikation unter bestimmten Umständen nicht in das Verzeichnis der Applikation wechselt und man daher nur mit "shel_find()" Dateien, die im gleichen Verzeichnis wie das gestartete Programm liegen, finden kann.

Die AES kennen ein Programm, das eine herausgehobene Bedeutung besitzt: das Default-Desktop-Programm, das automatisch nach dem Start von GEM und nach dem Beenden einer GEM-Applikation geladen wird. Im Atari-GEM ist das Desktop fest im ROM verankert und zusammen mit den AES zu einem Programm zusammengefaßt. Unter PC-GEM kann der Name auch nachträglich noch mit "shel_rdef()" abgefragt und mit "shel_wdef()" verändert werden.

Speziell für dieses Programm gibt es den sogenannten "Shell-Puffer" – einen statischen Speicherbereich, der immer resident bleibt. Er erlaubt es dem Desktop, seine aktuellen Einstellungen vor jedem Programmstart zu sichern, ohne erst eine Datei anlegen zu müssen (man bedenke, daß 1984 Festplatten noch nicht zur Standardausstattung gehörten). Dieser Puffer kann mittels "shel_put()" beschrieben und mit "shel_read()" ausgelesen werden.

Über den *Inhalt* des Shell-Puffers entscheidet allein das Desktop-Programm. Das Atari-Desktop verwaltet den Puffer als Textdatei, die beim Systemstart aus der Datei "DESKTOP.INF" (bis einschließlich GEM 1.4) bzw. "NEWDESK.INF" ausgelesen wird. Je nach Desktop-Version wird das Ende der Daten durch ein Nullbyte oder Control-Z (ASCII 26) gekennzeichnet.

Die XGRF-Bibliothek

Diese Funktionssammlung ist eine PC-GEM-spezifische Erweiterung. Die beiden XGRF-Funktionen erlauben es, Ersatz für die im PC-GEM gestrichenen Routinen für schrumpfende, wachsende und sich bewegende Rechtecke zu schaffen.

AES-Bindings

Der AES-Parameter-Block

Ebenso wie beim VDI benutzt man auch bei den AES zur Parameterübergabe spezielle Ein- und Ausgabefelder. Im einzelnen sind das:

WORD contrl[5];

Fünf Werte, mit denen Informationen über die aufgerufene Funktion und ihre Parameter festgelegt werden. Die Belegung ist wie folgt:

contrl[0]	Nummer der AES-Funktion
contrl[1]	Anzahl der 16-Bit-Werte im int_in-Feld
contrl[2]	Anzahl der 16-Bit-Werte im int_out-Feld
contrl[3]	Anzahl der 32-Bit-Werte im addr_in-Feld
contrl[4]	Anzahl der 32-Bit-Werte im addr_out-Feld

Darüber, welche Werte vor einem AES-Aufruf gesetzt werden müssen, gibt es keine klaren Informationen. Nötig ist es auf jeden Fall für contrl[0], contrl[1] und contrl[3]. Wenig sinnvoll erscheint es für contrl[2] und contrl[4] – schließlich wissen ja die AES-Funktionen selbst, wie viele Werte sie in den Ausgabefeldern zurückliefern.

Die mit dem Original-Entwicklungspaket mitgelieferten Bindings (unten beschrieben) setzen allerdings *alle* Werte des Arrays.

WORD global[12];

Dieses Feld enthält globale Daten für die Applikation und wird teils von "appl_init()", teils von anderen AES-Funktionen benutzt. Genauere Informationen dazu im Abschnitt über die Applikations-Bibliothek.

WORD int_in[16];

In diesem Array werden alle 16-Bit großen Parameter übergeben.

WORD int_out[7];

Hier übergeben AES-Funktionen alle 16-Bit großen Rückgabewerte.

LONG addr_in[2];

Dient zur Übermittlung von Zeigerparametern (wie Zeiger auf Zeichenketten) an AES-Funktionen.

LONG addr_out[1];

Hier werden 32-Bit große Rückgabewerte zurückgeliefert. Wird zur Zeit nur für "rsrc_gaddr()" benutzt.

Die Anfangsadressen der sechs Parameterfelder werden im "AES Parameterblock" abgelegt:

```
typedef struct
{
     WORD *cb_pcontrol;   /* Zeiger auf contrl[] */
     WORD *cb_pglobal;    /* Zeiger auf global[] */
     WORD *cb_pintin;     /* Zeiger auf int_in[] */
     WORD *cb_pintout;    /* Zeiger auf int_out[] */
     LONG *cb_padrin;     /* Zeiger auf addr_in[] */
     LONG *cb_padrout;    /* Zeiger auf addr_out[] */
} AESPB;
```

Beim Aufruf von AES-Funktionen ist damit nur noch ein Zeiger auf den Parameterblock zu übergeben.

Der AES-Trap

Zum Aufruf einer AES-Funktion lädt man Datenregister D0 mit der Konstante 200 und D1 mit der Anfangsadresse des AES-Parameterblocks und macht einen "TRAP #2"-Aufruf:

```
     .text
     .globl _crystal

_crystal:
     move.l    4(sp),d1  ; Adresse AESPB (AES Parameter Block)
     move.w    #200,d0   ; Opcode für AES
     trap      #2        ; Trap für GEM
     rts
```

Die Bezeichnung "crystal" stammt aus der Ursprungszeit von GEM, als Digital Research noch nicht den endgültigen Namen gefunden hatte. Darüber, welche Register verändert werden, gibt es keine klaren Informationen. Tatsache aber ist, daß die entsprechenden Routinen im ROM *alle* Register retten. Rückgabewerte erhält man nicht in Registern, sondern im "int_out"- und "addr_out"-Feld.

Eine weitere wichtige Frage ist, inwieweit GEM fehlertolerant ist, das heißt, ob es eine Absicherung gegen Abstürze infolge fehlerhafter Aufrufe gibt: Darauf kann man nur mit einem klaren Jein antworten. Falsche AES-Funktionsnummern werden von den AES vorbildlich mit einer Alarm-Box (falsche Funktionsnummer) quittiert. Falsche Parameter hingegen werden nur in den seltensten Fällen vernünftig abgefangen und führen normalerweise zu Abstürzen oder anderem Fehlverhalten.

AES-Aufrufe aus dem Supervisor-Modus sind nur schwer durchzuführen. Dies ist auch nicht weiter schlimm, da sich Aufrufe aus Interrupts heraus sowieso verbieten – die AES sind *nicht* re-entrant. Wer es dennoch probieren will, muß folgende Punkte beachten:

- Viele AES-Funktionen kehren auch dann im User-Modus zurück, wenn man sie aus dem Supervisor-Modus heraus aufgerufen hat. Daher muß man nach jedem Aufruf mittels "Super(1L)" feststellen, ob der Modus wieder korrigiert werden muß.

- Bei AES-Aufrufen dürfen sich User-Stack und Supervisor-Stack nicht überschneiden. Genau dies ist jedoch der Fall, wenn man mittels "Super(0L)" in den Supervisor-Modus geschaltet hatte. Abhilfe: eigenen Supervisor-Stack zu Fuß anlegen.

- Hinzu kommt, daß die AES sogar über das Ende des Supervisor-Stacks hinausschreiben. Ein "sicherer" Aufruf sollte daher etwa so aussehen (Quelle: "TOS 1.4 Release Notes"):

```
char my_stack[8192];

old_ssp = Super (&my_stack[8180]);/* nicht ganz aufs Ende! */

/* Aufrufe im Supervisor-Modus... */

Super (old_ssp);
```

Ein Beispiel-Binding

Die Funktion "crys_if" ("Crystal Interface") sorgt für die Besetzung des "contrl"-Arrays und macht den eigentlichen AES-Aufruf. Dazu bedient es sich einer Tabelle, in der für jede einzelne AES-Funktion die Werte für contrl[1], contrl[2] und contrl[3] vermerkt sind:

```
        .text
        .globl _ctrl_cnts

        .data
_ctrl_cnts:
        .dc.b   0, 1, 0  * appl_init
        .dc.b   2, 1, 1  * appl_read
        .dc.b   2, 1, 1  * appl_write
        .dc.b   0, 1, 1  * appl_find
        .dc.b   2, 1, 1  * appl_tplay
        .dc.b   1, 1, 1  * appl_trecord
        .dc.b   2, 1, 0  * appl_bvset
        .dc.b   0, 1, 0  * appl_yield
        .dc.b   0, 0, 0  * AES 18
        .dc.b   0, 1, 0  * appl_exit
        .dc.b   0, 1, 0  * evnt_keybd
        .dc.b   3, 5, 0  * evnt_button
        .dc.b   5, 5, 0  * evnt_mouse
        .dc.b   0, 1, 1  * evnt_mesag
        .dc.b   2, 1, 0  * evnt_timer
        .dc.b   16,7, 1  * evnt_multi
        .dc.b   2, 1, 0  * evnt_dclicks
        .dc.b   0, 0, 0  * AES 27
        .dc.b   0, 0, 0  * AES 28
        .dc.b   0, 0, 0  * AES 29
        .dc.b   1, 1, 1  * menu_bar
```

```
.dc.b   2, 1, 1  * menu_icheck
.dc.b   2, 1, 1  * menu_ienable
.dc.b   2, 1, 1  * menu_tnormal
.dc.b   1, 1, 2  * menu_text
.dc.b   1, 1, 1  * menu_register
.dc.b   1, 1, 0  * menu_unregister
.dc.b   2, 1, 0  * menu_click
.dc.b   0, 0, 0  * AES 38
.dc.b   0, 0, 0  * AES 39
.dc.b   2, 1, 1  * objc_add
.dc.b   1, 1, 1  * objc_delete
.dc.b   6, 1, 1  * objc_draw
.dc.b   4, 1, 1  * objc_find
.dc.b   1, 3, 1  * objc_offset
.dc.b   2, 1, 1  * objc_order
.dc.b   4, 2, 1  * objc_edit
.dc.b   8, 1, 1  * objc_change
.dc.b   0, 0, 0  * AES 48
.dc.b   0, 0, 0  * AES 49
.dc.b   1, 1, 1  * form_do
.dc.b   9, 1, 0  * form_dial
.dc.b   1, 1, 1  * form_alert
.dc.b   1, 1, 0  * form_error
.dc.b   0, 5, 1  * form_center
.dc.b   3, 3, 1  * form_keybd
.dc.b   2, 2, 1  * form_button
.dc.b   0, 0, 0  * AES 57
.dc.b   0, 0, 0  * AES 58
.dc.b   0, 0, 0  * AES 59
.dc.b   0, 0, 0  * AES 60
.dc.b   0, 0, 0  * AES 61
.dc.b   0, 0, 0  * AES 62
.dc.b   0, 0, 0  * AES 63
.dc.b   0, 0, 0  * AES 64
.dc.b   0, 0, 0  * AES 65
.dc.b   0, 0, 0  * AES 66
.dc.b   0, 0, 0  * AES 67
.dc.b   0, 0, 0  * AES 68
.dc.b   0, 0, 0  * AES 69
.dc.b   4, 3, 0  * graf_rubberbox
.dc.b   8, 3, 0  * graf_dragbox
```

```
.dc.b   6, 1, 0  * graf_movebox
.dc.b   8, 1, 0  * graf_growbox
.dc.b   8, 1, 0  * graf_shrinkbox
.dc.b   4, 1, 1  * graf_watchbox
.dc.b   3, 1, 1  * graf_slidebox
.dc.b   0, 5, 0  * graf_handle
.dc.b   1, 1, 1  * graf_mouse
.dc.b   0, 5, 0  * graf_mkstate
.dc.b   0, 1, 1  * scrp_read
.dc.b   0, 1, 1  * scrp_write
.dc.b   0, 1, 0  * scrp_clear
.dc.b   0, 0, 0  * AES 83
.dc.b   0, 0, 0  * AES 84
.dc.b   0, 0, 0  * AES 85
.dc.b   0, 0, 0  * AES 86
.dc.b   0, 0, 0  * AES 87
.dc.b   0, 0, 0  * AES 88
.dc.b   0, 0, 0  * AES 89
.dc.b   0, 2, 2  * fsel_input
.dc.b   0, 2, 3  * fsel_exinput
.dc.b   0, 0, 0  * AES 92
.dc.b   0, 0, 0  * AES 93
.dc.b   0, 0, 0  * AES 94
.dc.b   0, 0, 0  * AES 95
.dc.b   0, 0, 0  * AES 96
.dc.b   0, 0, 0  * AES 97
.dc.b   0, 0, 0  * AES 98
.dc.b   0, 0, 0  * AES 99
.dc.b   5, 1, 0  * wind_create
.dc.b   5, 1, 0  * wind_open
.dc.b   1, 1, 0  * wind_close
.dc.b   1, 1, 0  * wind_delete
.dc.b   2, 5, 0  * wind_get
.dc.b   6, 1, 0  * wind_set
.dc.b   2, 1, 0  * wind_find
.dc.b   1, 1, 0  * wind_update
.dc.b   6, 5, 0  * wind_calc
.dc.b   0, 0, 0  * wind_new
.dc.b   0, 1, 1  * rsrc_load
.dc.b   0, 1, 0  * rsrc_free
.dc.b   2, 1, 0  * rsrc_gaddr
```

```
        .dc.b   2, 1, 1  * rsrc_saddr
        .dc.b   1, 1, 1  * rsrc_obfix
        .dc.b   0, 0, 0  * AES 115
        .dc.b   0, 0, 0  * AES 116
        .dc.b   0, 0, 0  * AES 117
        .dc.b   0, 0, 0  * AES 118
        .dc.b   0, 0, 0  * AES 119
        .dc.b   0, 1, 2  * shel_read
        .dc.b   3, 1, 2  * shel_write
        .dc.b   1, 1, 1  * shel_get
        .dc.b   1, 1, 1  * shel_put
        .dc.b   0, 1, 1  * shel_find
        .dc.b   0, 1, 2  * shel_envrn
        .dc.b   0, 1, 2  * shel_rdef
        .dc.b   0, 0, 2  * shel_wdef
        .dc.b   0, 0, 0  * AES 128
        .dc.b   0, 0, 0  * AES 129
        .dc.b   6, 6, 0  * xgrf_stepcalc
        .dc.b   9, 1, 0  * xgrf_2box
        .end
```

Der AES-Parameter-Block wird direkt vor dem Aufruf der Funktion "appl_init()" initialisiert (siehe dort):

```
AESPB c;

WORD crys_if (WORD opcode)
{
     WORD i;
     WORD *paesb;

     contrl[0] = opcode;
     paespb = &ctrl_cnts[(opcode-10)*3];

     for (i = 1; i < 4; i++)
          control[i] = *paespb++;

     crystal (c);
     return int_out[0];
}
```

AES-Referenz

APPL-Funktionen

APPL_INIT (AES 10)

Diese Funktion initialisiert die GEM-Arrays (namentlich das Global-Feld) und meldet das laufende Programm als GEM-Applikation an. Die DR-Dokumentation zu GEM 2.0 gibt an, daß die erste Prozeßumschaltung frühestens nach zehn AES-Aufrufen stattfindet. Innerhalb dieses Zeitraums sollte man notwendige Speicherallozierungen – sei es mittels "rsrc_load()" oder "Malloc()" – vornehmen, um eine überflüssige Speicher-Zersplitterung zu vermeiden.

Bei Accessories ist besondere Vorsicht geboten, da ab GEM 1.4 das Desktop das Laden einer Autostart-Anwendung vorsieht. Man darf also *nicht* davon ausgehen, daß der Rest des Systems während der Accessory-Initialisierung stillhält. Dadurch wird alles, was Speicheranforderungen an GEMDOS provoziert ("rsrc_load()", "v_opnvwk()" etc.), zum Problem: Es könnte bereits ein anderer GEMDOS-Prozeß am Ruder sein, bevor man seine eigene Initialisierung beendet hat. Bei Beendigung dieses Prozesses wird dann der ganze in der Zwischenzeit allozierte Speicher freigegeben!

Daher sollte die Accessory-Initialisierung

- gleich nach dem "appl_init()" beginnen (und nicht erst nach der "AC_OPEN"-Mitteilung anfangen)

- mit "wind_update()" (BEG_UPDATE/END_UPDATE) geklammert werden (damit kann man normalerweise das Starten der Autostart-Applikation bremsen).

Deklaration in C:

```
extern AESPB c;
WORD int_in[16], int_out[7], contrl[5], global[15];
LONG addr_in[2], addr_out[1];

WORD appl_init (void)
{
     c.cb_pcontrol = control;
     c.cb_pglobal = global;
     c.cb_pintin = int_in;
```

```
    c.cb_pintout = int_out;
    c.cb_padrin = addr_in;
    c.cb_padrout = addr_out;
    control[4] = 0;
    crys_if (10);
    return int_out[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	10	Opcode für APPL_INIT
contrl+2	contrl[1]	0	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	0	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_out	int_out[0]	Return-Wert	

Parameter:

appl_init(): Bei erfolgreicher Initialisierung die Identifikationsnummer für die Applikation. Im Fehlerfall erhält man −1. Dann sollte man das Programm mit einer Fehlermeldung abbrechen und insbesondere weitere AES-Aufrufe vermeiden (viele Bibliotheken liefern übrigens immer die 1!). Die AES selbst merken sich die Kennung im internen GLOBAL-Feld der Applikation.

Bemerkungen

Wenn man "appl_init()" aufruft, ohne daß die AES betriebsbereit sind (also aus dem AUTO-Ordner heraus), wird speziell die AES-Versionsnummer im GLOBAL-Feld nicht gesetzt. Wenn man dort vorher eine Null hineinschreibt, kann man also leicht feststellen, ob das Programm aus dem AUTO-Ordner heraus gestartet worden ist. Dieses Verfahren ist zwar nirgendwo offiziell dokumentiert, wird aber ebenfalls vom Atari-Mausbeschleuniger "MACCEL3" verwendet.

APPL_READ (AES 11)

Liest eine bestimmte Anzahl von Bytes aus einem Ereignispuffer (Message Pipe).

Deklaration in C:

```
WORD appl_read (WORD rwid, WORD length, void *pbuff)
{
     int_in[0] = rwid;
     int_in[1] = length;
     addr_in[0] = buff;
     return crys_if (11);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	11	Opcode für APPL_READ
contrl+2	contrl[1]	2	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	1	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	rwid	
int_in+2	int_in[1]	length	
addr_in	addr_in[0]	pbuff	
int_out	int_out[0]	Return-Wert (0: Fehler)	

Parameter:

rwid:	id der Applikation, aus deren Ereignispuffer (Message Pipe) gelesen werden soll (normalerweise der eigene!)
length:	Anzahl der zu empfangenden Bytes
pbuff:	Anfangsadresse des Puffers, in dem die Nachricht abgelegt werden soll

APPL_WRITE (AES 12)

Schreibt eine Anzahl von Bytes in einen Ereignispuffer (Message Pipe). Eine besonders nette Anwendung dieser Funktion ist es, wenn eine Applikation sich selbst eine WM_REDRAW-Mitteilung schickt. Mehrere solcher Ereignisse werden nämlich von den AES nach Möglichkeit zu einer einzigen Mitteilung zusammengefaßt.

Leider gibt es einen kleinen Haken: der AES-Dispatcher legt bei "appl_write()" einen Prozeß schlafen, wenn die Message-Queue bereits voll ist. Wenn aber alle bereits in der Message-Queue liegenden Mitteilungen an das *eigene* Programm gerichtet sind, kommt es zum Systemstillstand – nichts geht mehr!

Abhilfe: Durch einen Mechanismus im eigenen Programm dafür sorgen, daß nicht mehrere Mitteilungen auf einmal in die Message-Queue eingespeist werden.

Deklaration in C:

```
WORD appl_write (WORD rwid, WORD length, void *pbuff)
{
     int_in[0] = rwid;
     int_in[1] = length;
     addr_in[0] = pbuff;
     return crys_if (12);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	12	Opcode für APPL_WRITE
contrl+2	contrl[1]	2	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	1	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	rwid	
int_in+2	int_in[1]	length	
addr_in	addr_in[0]	pbuff	
int_out	int_out[0]	Return-Wert (0: Fehler)	

Parameter:

rwid:	id der Applikation, zu der die Mitteilung gesendet werden soll (im Normalfall ein anderes Programm, z. B. ein Accessory)
length:	Anzahl der zu sendenden Bytes (mindestens der Inhalt eines Message-Puffers, also 16 Bytes). Eigene Versuche ergaben, daß in den vorhandenen AES-Versionen maximal 128 Bytes erlaubt sind.
pbuff:	Anfangsadresse des Puffers, in dem sich die zu sendende Information befindet (sollte in den ersten 16 Bytes dem AES-Standardformat für Mitteilungen entsprechen)

APPL_FIND (AES 13)

Um Informationen mit parallel laufenden Programmen austauschen zu können, muß man zunächst deren Identifikationsnummern (id) feststellen (siehe zum Beispiel bei "appl_read()" und "appl_write()"). Mit "appl_find()" kann man die Identifikationsnummer einer Applikation, deren Dateiname bekannt ist, ermitteln – allerdings nur, wenn sie mittels "shel_write()" gestartet worden ist.

Leider arbeitet diese Funktion nicht ganz fehlerfrei. Hat man eine Applikation beendet und ist zum Desktop zurückgekehrt, behauptet "appl_find()" immer noch, die Applikation sei im Speicher! Die Ursache hierfür ist darin zu suchen, daß das Desktop auf dem Atari keine echte Applikation ist und daher die AES von seiner Existenz nichts wissen. Daher wird auch der Name der vorher aktiven Applikation nicht gelöscht.

Auf dem Atari hat das aktive Hauptprogramm immer die Identifikationsnummer 0. Nummer 1 ist der Screen Manager (SCRENMGR ist der Name der Applikation). Die folgenden Nummern werden an die Accessories vergeben. Dies kann natürlich unter einer multitaskingfähigen GEM-Version anders sein.

Deklaration in C:

```
WORD appl_find (const CHAR *pname)
{
     addr_in[0] = pname;
     return crys_if (13);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	13	Opcode für APPL_FIND
contrl+2	contrl[1]	0	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	1	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
addr_in	addr_in[0]	pname	
int_out	int_out[0]	Return-Wert (–1: Fehler)	

Parameter:

pname:	Name der zu suchenden Applikation. Dabei handelt es sich nur um den eigentlichen Dateinamen des Programms (also ohne Extension). Ist dieser kürzer, muß der String mit Leerzeichen aufgefüllt werden.
appl_find():	Identifikationsnummer des gesuchten Programms (oder –1, wenn es nicht gefunden werden konnte)

APPL_TPLAY (AES 14)

Die AES erlauben es, Benutzeraktionen (also Mausbewegungen, Tastendrücke etc.) wie ein Bandgerät zu speichern und anschließend wieder abzuspielen. Diese Wiedergabefunktion übernimmt "appl_tplay()".

Deklaration in C:

```
WORD appl_tplay (APPLRECORD *tbuffer, WORD tlength, WORD tscale)
{
     int_in[0] = tlength;
     int_in[1] = tscale;
     addr_in[0] = tbuffer;
     return crys_if (14);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	14	Opcode für APPL_TPLAY
contrl+2	contrl[1]	2	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	1	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	tlength	
int_in+2	int_in[1]	tscale	
addr_in	addr_in[0]	tbuffer	
int_out	int_out[0]	Return-Wert (immer 1)	

Parameter:

tlength: Anzahl der wiederzugebenden Ereignisse

tscale: Geschwindigkeitsfaktor beim Abspielen der Ereignisse (50: halbe Geschwindigkeit, 100: normale Geschwindigkeit, 200: doppelte Geschwindigkeit usw.)

tbuffer: Zeiger auf den Speicherbereich, in dem die Ereignisse gespeichert worden sind

APPL_TRECORD (AES 15)

Dient, analog zu "appl_tplay()", zum Speichern von Benutzerereignissen, um sie später wieder abspielen zu können. Jedes Ereignis belegt acht Bytes (bzw. 6 Bytes auf PC-GEM) im Speicher.

```
typedef struct
{
     LONG type; /* WORD auf PC-GEM!!! */
     LONG what;
} APPLRECORD;
```

Dabei gibt "type" die Art des Ereignisses an (siehe unter EVNT-Funktionen):

0: Timer-Ereignis
1: Button-Ereignis
2: Maus-Ereignis
3: Tastatur-Ereignis

"what" enthält dann jeweils folgende Informationen:

Timer-Ereignis:	Anzahl der verstrichenen Millisekunden
Button-Ereignis:	Status der Maustaste im unteren Wort (0: nicht gedrückt, 1: gedrückt); Anzahl der Tastendrücke im oberen Wort.
Maus-Ereignis:	X-Koordinate (Oberes Wort), Y-Koordinate (Unteres Wort)
Tastatur-Ereignis:	Eingegebenes Zeichen (Unteres Wort), Tastaturstatus (Oberes Wort: K_RSHIFT (0x0001), K_LSHIFT (0x0002), K_CTRL (0x0004), K_ALT (0x0008)

Deklaration in C:

```
WORD appl_trecord (APPLRECORD *tbuffer, WORD tlength)
{
     int_in[0] = tlength;
     addr_in[0] = tbuffer;
     return crys_if (15);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	15	Opcode für APPL_TRECORD
contrl+2	contrl[1]	1	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	1	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	tlength	
addr_in	addr_in[0]	tbuffer	
int_out	int_out[0]	Return-Wert	

Parameter:

lbuffer:	Speicherbereich, in dem die Benutzerereignisse gespeichert werden (sollte wegen des Speicherformats achtmal bzw. sechsmal so groß wie count sein)
tlength:	Anzahl der zu speichernden Ereignisse
appl_trecord():	Anzahl der gespeicherten Ereignisse

Bemerkung
Aufgrund eines Fehlers funktioniert diese Funktion erst ab GEM 1.2.

APPL_BVSET (AES 16) – nur in PC-GEM ab Version 2.0

PC-GEM benötigt Informationen über die angeschlossenen logischen Laufwerke (für die Dateiauswahlbox). Mit "appl_bvset()" kann man die entsprechenden Informationen setzen.

Deklaration in C:

```
WORD appl_bvset (UWORD bvdisk, UWORD bvhard)
{
     int_in[0] = bvdisk;
     int_in[1] = bvhard;
     return crys_if (16);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	16	Opcode für APPL_BVSET
contrl+2	contrl[1]	2	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	0	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	bvdisk	
int_in+2	int_in[1]	bvhard	
int_out	int_out[0]	Return-Wert	

Parameter:

bvdisk: Bitvektor mit allen vorhandenen Disklaufwerken. Bit 15 (also das höchste Bit) steht für Laufwerk "A:"!

bvhard: Bitvektor mit allen vorhandenen Harddisklaufwerken (Format wie zuvor)

APPL_YIELD (AES 17) – nur in PC-GEM ab Version 2.0

Kann benutzt werden, um andere AES-Prozesse ans Ruder zu lassen (genauer: Es wird ein AES-Prozeß-Switch erzwungen).

Auf Atari-GEM-Versionen kann man statt dessen einen entsprechend kurzen Aufruf von "evnt_timer()" machen.

Deklaration in C:

```
WORD appl_yield (void)
{
     return crys_if(17);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	17	Opcode für APPL_YIELD
contrl+2	contrl[1]	0	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	0	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_out	int_out[0]	Return-Wert	

APPL_EXIT (AES 19)

Mit "appl_exit()" meldet sich ein Programm bei den AES ab. Die id des Programms wird wieder freigegeben und steht für andere Programme zur Verfügung. Außerdem wird installierten Accessories eine AC_CLOSE-Mitteilung geschickt.

Deklaration in C:

```
WORD appl_exit (void)
{
     return crys_if (19);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	19	Opcode für APPL_EXIT
contrl+2	contrl[1]	0	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	0	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_out	int_out[0]	Return-Wert (0: Fehler)	

EVNT-Funktionen

EVNT_KEYBD (AES 20)

Wartet auf einen Tastendruck und gibt den entsprechenden Code zurück. Keyboard-Ereignisse werden übrigens immer nur der Applikation gemeldet, der das aktive Fenster gehört – also Vorsicht bei Accessories.

Deklaration in C:

```
WORD evnt_keybd (void)
{
     return crys_if (20);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	20	Opcode für EVNT_KEYBD
contrl+2	contrl[1]	0	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	0	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_out	int_out[0]	Return-Wert	

Parameter:

evnt_keybd(): Tastencode der gedrückten Taste
(Bit 0..7: ASCII-Code, Bit 8..15: Scan-Code)

EVNT_BUTTON (AES 21)

Wartet auf ein festzulegendes Maustasten-Ereignis.

Deklaration in C:

```
WORD evnt_button (WORD clicks, UWORD mask, UWORD state, WORD *pmx,
                  WORD *pmy, WORD *pmb, WORD *pks)
{
     int_in[0] = clicks;
     int_in[1] = mask;
     int_in[2] = state;
     crys_if (21);
     *pmx = int_out[1];
     *pmy = int_out[2];
     *pmb = int_out[3];
     *pks = int_out[4];
     return int_out[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	21	Opcode für EVNT_BUTTON
contrl+2	contrl[1]	3	# Einträge in int_in
contrl+4	contrl[2]	5	# Einträge in int_out
contrl+6	contrl[3]	0	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	clicks	
int_in+2	int_in[1]	mask	
int_in+4	int_in[2]	state	
int_out	int_out[0]	Return-Wert	
int_out+2	int_out[1]	pmx	
int_out+4	int_out[2]	pmy	
int_out+6	int_out[3]	pmb	
int_out+8	int_out[4]	pks	

Parameter:

clicks:	Anzahl der maximal gezählten Tastendrücke (2: Doppelklick etc.)
mask:	Maske für den gewünschten Mausknopf (Bit 0: linker Knopf, Bit 1: rechter Knopf)
state:	Erwünschter Status (Bitbelegung wie bei mask, ein Bit muß genau dann gesetzt sein, wenn der zugehörige Mausknopf gedrückt werden soll, um das Ereignis auszulösen)
pmx:	X-Koordinate des Mauszeigers bei Ereignisauslösung
pmy:	Y-Koordinate des Mauszeigers bei Ereignisauslösung
pmb:	Maustastenstatus (siehe state)
pks:	Tastaturzustand im Augenblick der Meldung des Ereignisses. Dabei gilt folgende Bitbelegung: K_RSHIFT (0x0001): rechte Shift-Taste K_LSHIFT (0x0002): linke Shift-Taste K_CTRL (0x0004): Control-Taste K_ALT (0x0008): Alternate-Taste
evnt_button():	Anzahl der aufgetretenen Maustastendrücke (ist immer kleiner oder gleich clicks)

Bemerkung

Es ist *nicht* möglich, mittels "evnt_button()" *beide* Maustasten gleichzeitig (und unabhängig voneinander) abzufragen! Ab PC-GEM/3 unterstützen die Event-Funktionen überhaupt nur noch eine (die linke) Maustaste.

EVNT_MOUSE (AES 22)

Wartet, bis der Mauszeiger einen rechteckigen Teil des Bildschirms betritt oder verläßt.

Deklaration in C:

```
WORD evnt_mouse (WORD flags, WORD x, WORD y, WORD width,
                 WORD height, WORD *pmx, WORD *pmy, WORD *pmb,
                 WORD *pks)
{
     int_in[0] = flags;
     int_in[1] = x;
     int_in[2] = y;
     int_in[3] = width;
     int_in[4] = height;
     crys_if (22);
     *pmx = int_out[1];
     *pmy = int_out[2];
     *pmb = int_out[3];
     *pks = int_out[4];
     return (int_out[0]);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	22	Opcode für EVNT_MOUSE
contrl+2	contrl[1]	5	# Einträge in int_in
contrl+4	contrl[2]	5	# Einträge in int_out
contrl+6	contrl[3]	0	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	flags	
int_in+2	int_in[1]	x	
int_in+4	int_in[2]	y	
int_in+6	int_in[3]	width	
int_in+8	int_in[4]	height	

Adresse	Feldelement	Belegung
int_out	int_out[0]	Return-Wert (reserviert, immer 1)
int_out+2	int_out[1]	pmx
int_out+4	int_out[2]	pmy
int_out+6	int_out[3]	pmb
int_out+8	int_out[4]	pks

Parameter:

flags:	Betreten (0) oder Verlassen (1) des Rechtecks melden
x, y, width, height:	Rechteck-Koordinaten
pmx, pmy:	Position des Mauszeigers
pmb:	Tastenstatus beim Eintreten des Ereignisses (Bit 0: linker Mausknopf, Bit 1: rechter Mausknopf)
pks:	Tastaturzustand im Augenblick der Meldung des Ereignisses. Dabei gilt folgende Bitbelegung:
	K_RSHIFT (0x0001): rechte Shift-Taste
	K_LSHIFT (0x0002): linke Shift-Taste
	K_CTRL (0x0004): Control-Taste
	K_ALT (0x0008): Alternate-Taste

EVNT_MESAG (AES 23)

Wartet, bis im Ereignispuffer eine Meldung (Message) vorliegt.

Deklaration in C:

```
WORD evnt_mesag (WORD *pbuff)
{
     addr_in[0] = pbuff;
     return crys_if (23);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	23	Opcode für EVNT_MESAG
contrl+2	contrl[1]	0	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	1	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_out	int_out[0]	Return-Wert (reserviert, immer 1)	
addr_in	addr_in[0]	pbuff	

Parameter:

pbuff: Adresse des Speichers für die erwartete Message (Länge: 16 Bytes)

EVNT_TIMER (AES 24)

Wartet, bis die in Millisekunden anzugebende Zeit verstrichen ist. Man sollte in Programmen niemals "irgendwelche" Warteschleifen benutzen, sondern immer nur diese (so kann nämlich in der Zwischenzeit beispielsweise einem Accessory Prozessorzeit zugewiesen werden).

Deklaration in C:

```
WORD evnt_timer (UWORD locnt, UWORD hicnt)
{
     int_in[0] = locnt;
     int_in[1] = hicnt;
     return (crys_if(24));
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	24	Opcode für EVNT_TIMER
contrl+2	contrl[1]	2	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	0	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	locnt	
int_in+2	int_in[1]	hicnt	
int_out	int_out[0]	Return-Wert (reserviert, immer 1)	

Parameter:

locnt, hicnt: Low- und High-Word der Anzahl von Millisekunden

EVNT_MULTI (AES 25)

Wartet auf eine Kombination verschiedener Ereignisse. Vorsicht – es kann mehr als ein Ereignis gleichzeitig auftreten!

Deklaration in C:

```
WORD evnt_multi (UWORD flags, UWORD bclk, UWORD bmsk, UWORD bst,
                 UWORD m1flags, UWORD m1x, UWORD m1y, UWORD m1w,
                 UWORD m1h, UWORD m2flags, UWORD m2x, UWORD m2y,
                 UWORD m2w, UWORD m2h, WORD *mepbuff, UWORD tlc,
                 UWORD thc, UWORD *pmx, UWORD *pmy, WORD *pmb,
                 WORD *pks, UWORD *pkr, WORD *pbr)
{
     int_in[0] = flags;
     int_in[1] = bclk;
     int_in[2] = bmsk;
     int_in[3] = bst;
     int_in[4] = m1flags;
     int_in[5] = m1x;
     int_in[6] = m1y;
     int_in[7] = m1w;
     int_in[8] = m1h;
     int_in[9] = m2flags;
     int_in[10] = m2x;
     int_in[11] = m2y;
     int_in[12] = m2w;
     int_in[13] = m2h;
     addr_in[0] = mepbuff;
     int_in[14] = tlc;
     int_in[15] = thc;
     crys_if (25);
     *pmx = int_out[1];
     *pmy = int_out[2];
     *pmb = int_out[3];
     *pks = int_out[4];
     *pkr = int_out[5];
     *pbr = int_out[6];
     return int_out[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung
contrl	contrl[0]	25 Opcode für EVNT_MULTI
contrl+2	contrl[1]	16 # Einträge in int_in
contrl+4	contrl[2]	7 # Einträge in int_out
contrl+6	contrl[3]	1 # Einträge in addr_in
contrl+8	contrl[4]	0 # Einträge in addr_out
int_in	int_in[0]	flags
int_in+2	int_in[1]	bclk
int_in+4	int_in[2]	bmsk
int_in+6	int_in[3]	bst
int_in+8	int_in[4]	m1flags
int_in+10	int_in[5]	m1x
int_in+12	int_in[6]	m1y
int_in+14	int_in[7]	m1w
int_in+16	int_in[8]	m1h
int_in+18	int_in[9]	m2flags
int_in+20	int_in[10]	m2x
int_in+22	int_in[11]	m2y
int_in+24	int_in[12]	m2w
int_in+26	int_in[13]	m2h
int_in+28	int_in[14]	tlc
int_in+30	int_in[15]	thc
int_out	int_out[0]	Return-Wert
int_out+2	int_out[1]	pmx
int_out+4	int_out[2]	pmy
int_out+6	int_out[3]	pmb
int_out+8	int_out[4]	pks
int_out+10	int_out[5]	pkr
int_out+12	int_out[6]	pbr
addr_in	addr_in[0]	mepbuff

Parameter:

Die meisten Parameter ergeben sich direkt aus der Definition von "evnt_mouse()", "evnt_keybd()", "evnt_button()" und "evnt_mesag()". Man beachte dabei, daß man auf zwei verschiedene Maus-Ereignisse (m1flags und m2flags etc.) prüfen kann.

Alle weiteren Parameter:

flags: Legt fest, auf was für ein Ereignis die Applikation warten soll. Die Bedeutung der einzelnen Bits:
Bit 0: MU_KEYBD (0x0001) (Tastaturereignis)
Bit 1: MU_BUTTON (0x0002) (Mausknopfereignis)
Bit 2: MU_M1 (0x0004) (erstes Mausereignis)
Bit 3: MU_M2 (0x0008) (zweites Mausereignis)
Bit 4: MU_MESAG (0x0010) (Mitteilungs-Ereignis)
Bit 5: MU_TIMER (0x0020) (Timer-Ereignis)

evnt_multi(): Die Ereignisse, die tatsächlich eingetreten sind. Bitbelegung wie bei flags.

Bemerkungen
"pmx", "pmy", "pmb" und "pks" werden *immer* – ungeachtet "flags" – zurückgeliefert. Die zurückgelieferten Werte spiegeln jeweils den Zustand bei Abfrage des Ereignisses wieder.

Durch die unglaubliche Parameterzahl verbraucht ein Aufruf von "evnt_multi()" beträchtlich Prozessorzeit. Man denke daran, daß zunächst alle Parameter auf den Stack gepackt und dann auf die Eingabearrays verteilt werden. Genau dasselbe passiert dann noch mal intern in den AES. Erschwerend kommt hinzu, daß man meistens viele Parameter gar nicht setzen will.

Das hat mittlerweile auch Digital Research erkannt: Im GEM/3-Toolkit findet man daher die folgende Alternative, bei der man nur einen Zeiger auf eine "MEVENT"-Struktur übergibt.

```
typedef struct
{
    WORD g_x, g_y, g_w, g_h;
} GRECT;

typedef struct
{
    UWORD     e_flags;
    UWORD     e_bclk;
    UWORD     e_bmsk;
    UWORD     e_bst;
    UWORD     e_m1flags;
    GRECT     e_m1;
    UWORD     e_m2flags;
    GRECT     e_m2;
    WORD      *e_mepbuf;
    ULONG     e_time;
```

```
    WORD        e_mx;
    WORD        e_my;
    UWORD       e_mb;
    UWORD       e_ks;
    UWORD       e_kr;
    UWORD       e_br;
    UWORD       e_m3flags;
    GRECT       e_m3;
    WORD        e_xtra0;
    WORD        *e_smepbuf;
    ULONG       e_xtra1;
    ULONG       e_xtra2;
} MEVENT;

WORD evnt_event (MEVENT *pmevent)
{
    memcpy (int_in, pmevent, 14 * sizeof (UWORD));
    addr_in[0] = pmevent->e_mepbuf;
    int_in[14] = (WORD) ((pmevent->e_time) 0xFFFF);
    int_in[15] = (WORD) ((pmevent->e_time) >> 16);
    crys_if (25);
    memcpy (&(pmevent->e_mx), &(int_out[1]), 6 * sizeof (UWORD));
    return int_out[0];
}
```

EVNT_DCLICK (AES 26)

Setzt die Geschwindigkeit für das "Doppelklicken" der Maus oder stellt den aktuellen Wert ein.

Deklaration in C:

```
WORD evnt_dclick (WORD rate, WORD setit)
{
     int_in[0] = rate;
     int_in[1] = setit;
     return crys_if (26);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	26	Opcode für EVNT_DCLICK
contrl+2	contrl[1]	2	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	0	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	rate	
int_in+2	int_in[1]	setit	
int_out	int_out[0]	Return-Wert	

Parameter:

rate:	Neue Geschwindigkeit für Doppelklicks (0..4)
setit:	0: aktuellen Wert feststellen und rate ignorieren
	1: neuen Wert setzen
evnt_dclick:	jetzt gültige Geschwindigkeit (unabhängig von setit)

Bemerkungen

In älteren GEM-Dokumentationen auch häufig "evnt_dclicks()" genannt.

MENU-Funktionen

MENU_BAR (AES 30)

Setzt oder desaktiviert die Menüleiste eines Menüobjektbaums. Auf gar keinen Fall vor Beendigung des Programms vergessen!

Deklaration in C:

```
WORD menu_bar (OBJECT *tree, WORD showit)
{
     addr_in[0] = tree;
     int_in[0] = showit;
     return crys_if (30);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung
contrl	contrl[0]	30 Opcode für MENU_BAR
contrl+2	contrl[1]	1 # Einträge in int_in
contrl+4	contrl[2]	1 # Einträge in int_out
contrl+6	contrl[3]	1 # Einträge in addr_in
contrl+8	contrl[4]	0 # Einträge in addr_out
int_in	int_in[0]	showit
addr_in	addr_in[0]	tree
int_out	int_out[0]	Return-Wert (0: Fehler)

Parameter:

showit: 0: Menüleiste löschen
1: Menüleiste setzen
tree: Anfangsadresse des Objektbaumes des Menüs

MENU_ICHECK (AES 31)

Löscht oder setzt ein Häkchen vor einem Menüeintrag (dazu sollte man jeden Eintrag mit zwei Leerzeichen beginnen). Ab PC-GEM 2.0 gibt es statt dessen kleine Pfeilspitzen (Dreiecke).

Deklaration in C:

```
WORD menu_icheck (OBJECT *tree, WORD itemnum, WORD checkit)
{
     addr_in[0] = tree;
     int_in[0] = itemnum;
     int_in[1] = checkit;
     return crys_if (31);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	31	Opcode für MENU_ICHECK
contrl+2	contrl[1]	2	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	1	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	itemnum	
int_in+2	int_in[1]	checkit	
addr_in	addr_in[0]	tree	
int_out	int_out[0]	Return-Wert (0: Fehler)	

Parameter:

itemnum:	Objektnummer des betreffenden Menüeintrags
checkit:	0: Häkchen löschen
	1: Häkchen setzen
tree:	Anfangsadresse des Menü-Objektbaums

MENU_IENABLE (AES 32)

Schaltet Menüeinträge ein und aus (gekennzeichnet durch graue Schattierung).

Deklaration in C:

```
WORD menu_ienable (OBJECT *tree, WORD itemnum, WORD enableit)
{
     addr_in[0] = tree;
     int_in[0] = itemnum;
     int_in[1] = enableit;
     return crys_if (32);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	32	Opcode für MENU_IENABLE
contrl+2	contrl[1]	2	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	1	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	itemnum	
int_in+2	int_in[1]	enableit	
addr_in	addr_in[0]	tree	
int_out	int_out[0]	Return-Wert (0: Fehler)	

Parameter:

itemnum:	Objektnummer des zu bearbeitenden Menüeintrags
enableit:	0: desaktivieren, 1: aktivieren
tree:	Anfangsadresse des Menü-Objektbaums

MENU_TNORMAL (AES 33)

Invertiert Menütitel.

Deklaration in C:

```
WORD menu_tnormal (OBJECT *tree, WORD titlenum, WORD normalit)
{
     addr_in[0] = tree;
     int_in[0] = titlenum;
     int_in[1] = normalit;
     return crys_if (33);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	33	Opcode für MENU_TNORMAL
contrl+2	contrl[1]	2	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	1	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	titlenum	
int_in+2	int_in[1]	normalit	
addr_in	addr_in[0]	tree	
int_out	int_out[0]	Return-Wert (0: Fehler)	

Parameter:

titlenum:	Objektnummer des Menütitels
normalit:	0: invers darstellen
	1: normal darstellen
tree:	Anfangsadresse des Menü-Objektbaums

MENU_TEXT (AES 34)

Ändert den Text in einem Menüeintrag. Damit hat man die Möglichkeit, Texte in Menüs vom momentanen Zustand eines Programms abhängig zu machen ("kontext-sensitiv").

Accessories können diese Methode natürlich *nicht* benutzen, da sie die Adresse des aktiven Menübaums nicht kennen. Statt dessen muß man also von Beginn an genug Platz für den Menüeintrag freihalten und dann den entsprechenden Text an die entsprechende Adresse kopieren.

Deklaration in C:

```
WORD menu_text (OBJECT *tree, WORD inum, CHAR *ptext)
{
     addr_in[0] = tree;
     int_in[0] = inum;
     addr_in[1] = ptext;
     return crys_if (34);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	34	Opcode für MENU_TEXT
contrl+2	contrl[1]	1	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	2	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	inum	
addr_in	addr_in[0]	tree	
addr_in+4	addr_in[1]	ptext	
int_out	int_out[0]	Return-Wert (0: Fehler)	

Parameter:

item: Objektnummer des zu ändernden Menüeintrags

text: Anfangsadresse des neuen Eintrags (GEM benutzt nur den Zeiger, legt also *keine* Kopie an!). Sollte nicht länger als der bisherige Eintrag sein.
tree: Anfangsadresse des Menü-Objektbaums

Bemerkungen
Ab PC-GEM 2.0 legen die AES eine Kopie des Menü-Objektbaums an. Daher muß man nach jeder Änderung eines Eintrags das Menü mittels "menu_bar()" neu auf den Bildschirm bringen! Für Accessories hat man sich gezwungenermaßen einen Ausweg einfallen lassen: Sie können in "tree" anstelle der (unbekannten) Baumadresse die Nummer des Accessory-Eintrags (Rückgabewert von "menu_register()") übergeben (dazu setzt man das obere Wort auf Null und übergibt im unteren Wort die Nummer des Eintrags).

MENU_REGISTER (AES 35)

Trägt Namen im ersten Drop-Down-Menü für Accessories ein. Insgesamt sechs quasi gleichzeitig laufende Accessories können gemeinsam sechs Einträge im Menü benutzen.

Deklaration in C:

```
WORD menu_register (WORD pid, const CHAR *pstr)
{
     int_in[0] = pid;
     addr_in[0] = pstr;
     return crys_if (35);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	35	Opcode für MENU_REGISTER
contrl+2	contrl[1]	1	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	1	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	pid	
addr_in	addr_in[0]	pstr	
int_out	int_out[0]	Return-Wert	

Parameter:

pid: Identifikation des Accessories (ap_id)

pstr: Anfangsadresse des Textes für den Menüeintrag (GEM benutzt nur den Zeiger, legt also *keine* Kopie an)

menu_register(): Menü-Kennung für das Accessory oder -1, falls kein Platz für weitere Einträge ist

Bemerkungen

Ein Aufruf von "menu_register()" wirkt sich erst dann aus, wenn das jeweilige Hauptprogramm einen Aufruf von "menu_bar()" macht. Daher sollte man bei Accessories gleich nach Start den Eintrag vornehmen – sonst verpaßt man nämlich den "menu_bar()"-Aufruf des Desktops.

MENU_UNREGISTER (AES 36) – nur in PC-GEM ab Version 2.0

Ab PC-GEM 2.0 können Accessories ihren Namen wieder aus dem Desk-Menü entfernen.

Deklaration in C:

```
WORD menu_unregister (WORD mid)
{
     int_in[0] = mid;
     return crys_if (36);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	36	Opcode für MENU_UNREGISTER
contrl+2	contrl[1]	1	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	0	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	mid	
int_out	int_out[0]	Return-Wert (0: Fehler)	

Parameter:

mid: Identifikation des Accessories (ap_id)

MENU_CLICK (AES 37) – nur in PC-GEM/3

Ab PC-GEM/3 kann man die Behandlung der Drop-Down-Menüs auf "Pull-Down" umstellen.

Deklaration in C:

```
WORD menu_click (WORD click, WORD setit)
{
     int_in[0] = click;
     int_in[1] = setit;
     return crys_if (37);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	37	Opcode für MENU_CLICK
contrl+2	contrl[1]	2	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	0	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	click	
int_in+2	int_in[1]	setit	
int_out	int_out[0]	Return-Wert	

Parameter:

click:	0: Dropdown,	1: Pulldown
setit:	0: abfragen,	1: setzen
menu_click():	0: Dropdown,	1: Pulldown

OBJC-Funktionen

OBJC_ADD (AES 40)

Stellt die hierarchische Verknüpfung zwischen zwei Objekten in einem Objektbaum her.

Deklaration in C:

```
WORD objc_add (OBJECT *tree, WORD parent, WORD child)
{
     addr_in[0] = tree;
     int_in[0] = parent;
     int_in[1] = child;
     return crys_if (40);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	40	Opcode für OBJC_ADD
contrl+2	contrl[1]	2	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	1	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	parent	
int_in+2	int_in[1]	child	
addr_in	addr_in[0]	tree	
int_out	int_out[0]	Return-Wert (0: Fehler)	

Parameter:

parent:	Objektnummer des Objekts, zu dem das Child hinzugefügt werden soll
child:	Objektnummer des hinzuzufügenden Objekts ("ob_head" und "ob_tail" müssen bereits passend initialisiert sein).
tree:	Adresse des Objektbaumes

OBJC_DELETE (AES 41)

Löst ein Objekt aus der Liste des Objektbaumes.

Deklaration in C:

```
WORD objc_delete (OBJECT *tree, WORD delob)
{
     addr_in[0] = tree;
     int_in[0] = delob;
     return crys_if (41);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	41	Opcode für OBJC_DELETE
contrl+2	contrl[1]	1	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	1	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	delob	
addr_in	addr_in[0]	tree	
int_out	int_out[0]	Return-Wert (0: Fehler)	

Parameter:

delob: Objektnummer des herauszulösenden Objekts
tree: Anfangsadresse des Objektbaums

OBJC_DRAW (AES 42)

Stellt ganze Objekte oder Teile von Objekten auf dem Bildschirm dar. Dabei kann zusätzlich ein Bildschirmausschnitt in Koordinaten angegeben werden, auf den die Darstellung beschränkt wird. Dies ist beispielsweise beim Wiederherstellen von Bildausschnitten, die von einem Fenster überlagert wurden, nützlich.

Deklaration in C:

```
WORD objc_draw (OBJECT *tree, WORD drawob, WORD depth, WORD xc,
                WORD yc, WORD wc, WORD hc)
{
     addr_in[0] = tree;
     int_in[0] = drawob;
     int_in[1] = depth;
     int_in[2] = xc;
     int_in[3] = yc;
     int_in[4] = wc;
     int_in[5] = hc;
     return crys_if (42);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	42	Opcode für OBJC_DRAW
contrl+2	contrl[1]	6	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	1	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	drawob	
int_in+2	int_in[1]	depth	
int_in+4	int_in[2]	xc	
int_in+6	int_in[3]	yc	
int_in+8	int_in[4]	wc	
int_in+10	int_in[5]	hc	
addr_in	addr_in[0]	tree	
int_out	int_out[0]	Return-Wert (0: Fehler)	

Parameter:

drawob:	Objektnummer des ersten zu zeichnenden Objekts
depth:	Anzahl der zu zeichnenden Objekt-Ebenen (0: nur das erste Objekt)
xc, yc, wc, hc:	Position und Maße des begrenzenden (clipping) Rechtecks
tree:	Anfangsadresse des betreffenden Objektbaums

OBJC_FIND (AES 43)

Ermittelt, über welchem Objekt sich der Mauszeiger befindet (oder allgemeiner: welches Objekt an einer gegebenen Bildschirmposition steht).

Deklaration in C:

```
WORD objc_find (OBJECT *tree, WORD startob, WORD depth, WORD mx,
                WORD my)
{
     addr_in[0] = tree;
     int_in[0] = startob;
     int_in[1] = depth;
     int_in[2] = mx;
     int_in[3] = my;
     return crys_if (43);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	43	Opcode für OBJC_FIND
contrl+2	contrl[1]	4	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	1	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	startob	
int_in+2	int_in[1]	depth	
int_in+4	int_in[2]	mx	
int_in+6	int_in[3]	my	
addr_in	addr_in[0]	tree	
int_out	int_out[0]	Return-Wert (–1: nichts gefunden)	

Parameter:

startob: Objektnummer des Objekts, bei dem die Suche beginnen soll

depth:	Anzahl der Ebenen in der Objekthierarchie, die durchsucht werden sollen (0: nur Mutterobjekt)
mx,my:	Koordinaten der betreffenden Bildschirmposition
tree:	Anfangsadresse des zu untersuchenden Objektbaums
objc_find():	Objektnummer des gefundenen Objekts

OBJC_OFFSET (AES 44)

Berechnet die Position eines Objekts in absoluten Bildschirmkoordinaten.

Deklaration in C:

```
WORD objc_offset (OBJECT *tree, WORD obj, WORD *poffx, WORD *poffy)
{
     addr_in[0] = tree;
     int_in[0] = obj;
     crys_if (44);
     *poffx = int_out[1];
     *poffy = int_out[2];
     return int_out[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	44	Opcode für OBJC_OFFSET
contrl+2	contrl[1]	1	# Einträge in int_in
contrl+4	contrl[2]	3	# Einträge in int_out
contrl+6	contrl[3]	1	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	obj	
addr_in	addr_in[0]	tree	
int_out	int_out[0]	Return-Wert (0: Fehler)	
int_out+2	int_out[1]	poffx	
int_out+4	int_out[2]	poffy	

Parameter:

obj: Objektnummer des betreffenden Objekts
tree: Anfangsadresse des Objektbaums
poffx, poffy: die berechneten Koordinaten

OBJC_ORDER (AES 45)

Die Position eines Objekts innerhalb eines Unterbaums wird geändert.

Deklaration in C:

```
WORD objc_order (OBJECT *tree, WORD mov_obj, WORD newpos)
{
     addr_in[0] = tree;
     int_in[0] = mov_obj;
     int_in[1] = newpos;
     return crys_if (45);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	45	Opcode für OBJC_ORDER
contrl+2	contrl[1]	2	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	1	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	mov_obj	
int_in+2	int_in[1]	newpos	
addr_in	addr_in[0]	tree	
int_out	int_out[0]	Return-Wert (0: Fehler)	

Parameter:

mov_obj:	Objektnummer des betreffenden Objekts	
newpos:	0:	an den Anfang
	1:	an zweite Stelle
	2:	an dritte Stelle (usw.)
	−1:	an das Ende
tree:	Anfangsadresse des Objektbaums	

OBJC_EDIT (AES 46)

Erlaubt Texteingaben in Objekte vom Typ G_FTEXT oder G_FBOXTEXT.

Deklaration in C:

```
WORD objc_edit (OBJECT *tree, WORD obj, WORD inchar, WORD *idx,
                WORD kind)
{
     addr_in[0] = tree;
     int_in[0] = obj;
     int_in[1] = inchar;
     int_in[2] = *idx;
     int_in[3] = kind;
     crys_if (46);
     *idx = int_out[1];
     return int_out[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	46	Opcode für OBJC_EDIT
contrl+2	contrl[1]	4	# Einträge in int_in
contrl+4	contrl[2]	2	# Einträge in int_out
contrl+6	contrl[3]	1	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	obj	
int_in+2	int_in[1]	inchar	
int_in+4	int_in[2]	Inhalt von idx	
int_in+6	int_in[3]	kind	
addr_in	addr_in[0]	tree	
int_out	int_out[0]	Return-Wert (0: Fehler)	
int_out+2	int_out[1]	idx	

Parameter:

obj:	Nummer des zu editierenden Objekts	
inchar:	Eingegebenes Zeichen	
idx:	Position des Zeichens im String	
kind:	Funktionsauswahl:	
	ED_START (0):	reserviert
	D_INIT (1):	Berechne aus te_ptext und te_ptmplt einen formatierten String und schalte Cursor ein
	ED_CHAR (2):	Betreffendes Zeichen verarbeiten und den String neu anzeigen
	ED_END (3):	Cursor ausschalten

Bemerkungen

Vorsicht! Bei dem hier angegebenen Binding wird davon ausgegangen, daß man für int_in[2] dieselbe Variable wie für int_out[1] verwendet – genauso sieht es auch in den Bibliotheken von DR und Megamax aus. In der ursprünglichen GEM-Dokumentation von DR hingegen waren dafür verschiedene Variablen angegeben. Der Aufruf lautete also:

```
objc_edit (tree, obj, inchar, idx, kind, &idx);
```

Möglicherweise existieren AES-Bibliotheken, bei denen "objc_edit()" noch immer auf diese Weise eingebunden ist!

OBJC_CHANGE (AES 47)

Ändert den Status (ob_state) eines Objekts und zeichnet es gegebenenfalls neu, sofern es sich innerhalb eines gegebenen Begrenzungsrechtecks (clipping rectangle) befindet.

Deklaration in C:

```
WORD objc_change (OBJECT *tree, WORD drawob, WORD resvd, WORD xc,
                  WORD yc, WORD wc, WORD hc, WORD newstate,
                  WORD redraw)
{
     addr_in[0] = tree;
     int_in[0] = drawob;
     int_in[1] = resvd;
     int_in[2] = xc;
     int_in[3] = yc;
     int_in[4] = wc;
     int_in[5] = hc;
     int_in[6] = newstate;
     int_in[7] = redraw;
     return crys_if (47);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	47	Opcode für OBJC_CHANGE
contrl+2	contrl[1]	8	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	1	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	drawob	
int_in+2	int_in[1]	resvd	
int_in+4	int_in[2]	xc	
int_in+6	int_in[3]	yc	
int_in+8	int_in[4]	wc	
int_in+10	int_in[5]	hc	
int_in+12	int_in[6]	newstate	

Adresse	Feldelement	Belegung
int_in+14	int_in[7]	redraw
addr_in	addr_in[0]	tree
int_out	int_out[0]	Return-Wert (0: Fehler)

Parameter:

drawob:	Objektnummer des betreffenden Objekts
resvd:	Reserviert; muß 0 sein
xc, yc, wc, hc:	Koordinaten des begrenzenden Rechtecks
newstate:	Neuer Status (ob_state) des Objekts
redraw:	0: Objekt nicht neu zeichnen 1: Objekt neu zeichnen
tree:	Anfangsadresse des Objektbaums

Bemerkung
In neueren Bindings wird das resvd-Feld auch häufig ausgelassen und automatisch auf Null gesetzt.

FORM-Funktionen

FORM_DO (AES 50)

Dies ist die zentrale Routine im Dialog-Manager. Sie übernimmt die komplette Verwaltung eines Formular-Objektes, bis der Benutzer ein Objekt mit EXIT- oder TOUCHEXIT-Status anklickt.

Deklaration in C:

```
WORD form_do (OBJECT *form, WORD start)
{
     addr_in[0] = form;
     int_in[0] = start;
     return crys_if (50);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	50	Opcode für FORM_DO
contrl+2	contrl[1]	1	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	1	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	start	
addr_in	addr_in[0]	form	
int_out	int_out[0]	Return-Wert	

Parameter:

start: Nummer des ersten zu editierenden Objekts im Baum (oder Null)
form: Adresse des Objektbaums
form_do(): Nummer des Objekts, das der Benutzer zur Beendigung angeklickt hat (bei Doppelklick auf TOUCHEXIT-Objekten wird Bit 15 gesetzt).

FORM_DIAL (AES 51)

"FMD_START" reserviert einen Bildschirmausschnitt. Dies hat normalerweise keine weitere Auswirkung, wird aber häufig von "Bildschirmbeschleunigern" benutzt, um den belegten Bildschirmausschnitt in einen Puffer zu kopieren. Daher auf gar keinen Fall weglassen!

"FMD_GROW" zeichnet eine Reihe von sich ausdehnenden Rechtecken. Man sollte diese Funktion nicht überstrapazieren, nur selten ist sie wirklich sinnvoll. Ihr Sinn und Zweck besteht darin, dem Benutzer *zusätzliche* Informationen darzubieten – zum Beispiel wenn ein selektiertes Objekt per Tastaturkürzel "geöffnet" wird.

Im Zweifelsfall sollte man eine Option anbieten, diese Funktion auszulassen: Geübte Benutzer werden es einem danken.

"FMD_SHRINK" ist das Gegenstück zu "FMD_GROW" (schrumpfende Rechtecke).

"FMD_FINISH" gibt den durch "FMD_START" reservierten Bildausschnitt wieder frei. In der Realität heißt das: GEM sorgt dafür, daß alle betroffenen Fensterinhalte wieder restauriert werden (das heißt allerdings auch: Das Rechteck muß vollständig innerhalb des Arbeitsbereichs von Fenster 0 liegen). Dazu verschickt es an alle betroffenen AES-Prozesse entsprechende WM_REDRAW-Mitteilungen.

Deklaration in C:

```
WORD form_dial (WORD flag, WORD littlx, WORD littly, WORD littlw,
                WORD littlh, WORD bigx, WORD bigy, WORD bigw,
                WORD bigh)
{
     int_in[0] = flag;
     int_in[1] = littlx;
     int_in[2] = littly;
     int_in[3] = littlw;
     int_in[4] = littlh;
     int_in[5] = bigx;
     int_in[6] = bigy;
     int_in[7] = bigw;
     int_in[8] = bigh;
     return crys_if (51);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	51	Opcode für FORM_DIAL
contrl+2	contrl[1]	9	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	0	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	flag	
int_in+2	int_in[1]	littlx	
int_in+4	int_in[2]	littly	
int_in+6	int_in[3]	littlw	
int_in+8	int_in[4]	littlh	
int_in+10	int_in[5]	bigx	
int_in+12	int_in[6]	bigy	
int_in+14	int_in[7]	bigw	
int_in+16	int_in[8]	bigh	
int_out	int_out[0]	Return-Wert (0: Fehler)	

Parameter:

flag: Typ der FORM_DIAL-Funktion:
- FMD_START (0): Reserviert Bildschirmbereich für Dialogbox.
- FMD_GROW (1): Zeichnet sich ausdehnendes Rechteck.
- FMD_SHRINK (2): Zeichnet schrumpfendes Rechteck.
- FMD_FINISH (3): Gibt Bildschirmbereich wieder frei.

littl...: Koordinaten für Rechteck in seiner kleinsten Größe

big...: Koordinaten für Rechteck in seiner größten Ausdehnung

Bemerkungen

Die Subfunktionen "FMD_GROW" und "FMD_SHRINK" sind übrigens in PC-GEM 2.0 dem Rechtsstreit zwischen Digitial Research und Apple zum Opfer gefallen!

"FMD_FINISH" wird gern dazu benutzt, um einen Bildschirmneuaufbau auszulösen. Aber *Vorsicht*: Die Menüleiste (die außerhalb des Arbeitsbereichs von Fenster 0 liegt) kann man so nicht restaurieren!

FORM_ALERT (AES 52)

Stellt eine Alarmbox (Alertbox) auf dem Bildschirm dar.

"form_alert()" ist kein Ersatz für eine Dialogbox und sollte ausschließlich für kurze Informationen und Fehlermeldungen eingesetzt werden. Dabei sollte man die verschiedenen Symbole folgendermaßen benutzen:

Das Ausrufezeichen ("NOTE") steht für eine kurze Meldung, die der Benutzer nur bestätigen soll. Das könnte ein Sicherheitshinweis, aber auch eine Benachrichtigung über einen abgeschlossenen Arbeitsgang sein.

Das Fragezeichen ("WAIT") dient für Sicherheitsabfragen, *bevor* ein bestimmter Vorgang (wie zum Beispiel das Formatieren einer Diskette) durchgeführt wird.

Das Stopschild ("STOP") ist angebracht, wenn ein ernsthaftes Problem aufgetreten ist, das der Benutzer *unbedingt* zur Kenntnis nehmen muß.

Deklaration in C:

```
WORD form_alert (WORD defbut, const CHAR *astring)
{
     int_in[0] = defbut;
     addr_in[0] = astring;
     return crys_if (52);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	52	Opcode für FORM_ALERT
contrl+2	contrl[1]	1	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	1	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	defbut	
addr_in	addr_in[0]	astring	
int_out	int_out[0]	Return-Wert	

Parameter:

defbut: Nummer des Default-Knopfes (0: keiner, 1: erster etc.)

astring: Adresse des Strings mit den Daten für die Alertbox.
Format: "[<Icon>][<Text>][<Buttons>]", wobei <Icon>:
NO_ICON (0): kein Icon
NOTE (1)
WAIT (2)
STOP (3)
<Text>: maximal fünf Textzeilen mit max. 30 Zeichen, untereinander durch >|< getrennt
<Buttons>: bis zu drei durch >|< getrennte Button-Namen.

form_alert(): Nummer des Knopfs, mit dem die Alertbox verlassen wurde (der am weitesten links stehende Knopf hat die Nummer 1).

FORM_ERROR (AES 53)

Zeigt eine Warnmeldung für MS-DOS-Fehler an. Dies aus Kompatibilitätsgründen zum PC-GEM. Daher muß man zunächst die GEMDOS-Fehlernummern auf die Codierung von MS-DOS umrechnen!

Deklaration in C:

```
WORD form_error (WORD errnum)
{
     int_in[0] = errnum;
     return crys_if (53);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	53	Opcode für FORM_ERROR
contrl+2	contrl[1]	1	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	0	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	errnum	
int_out	int_out[0]	Return-Wert	

Parameter:

errnum: MS-DOS-Fehlernummer. Als Formel zur Berechnung der Fehlernummer aus dem vom GEMDOS gelieferten Code hat sich "(~errno)–30" bewährt.

- 2,3,18: "Diese Anwendung kann das angesprochene Objekt nicht finden."
- 4: "Die Anwendung benötigt mehr Platz zum Öffnen einer neuen Datei. Schließen Sie eine nicht benötigte Datei."
- 5: "Objekt mit gleichem Namen bereits vorhanden bzw. hat den "Nur-Lesen"-Status."
- 8,10,11: "Der Arbeitsspeicher reicht nicht für diese Anwendung."
- 15: "Floppy mit dieser Kennung unbekannt."

form_error(): Nummer des Buttons, den der Benutzer gedrückt hat (immer 0, da alle Alert-Boxen nur einen Knopf haben)

Bemerkungen

Der Wortlaut der Fehlermeldungen in PC-GEM 2.0 weicht teilweise beträchtlich davon ab. Sinngemäß erhält man aber die gleichen Mitteilungen.

FORM_CENTER (AES 54)

Zentriert Objekt in der Bildschirmmitte und gibt die tatsächliche Position auf dem Bildschirm zurück. Dabei werden für Breite und Höhe auch die speziellen Attribute wie z. B. "Outlined" berücksichtigt (Ausnahme: "Shadowed"). Diese Werte kann man dann speziell in Verbindung mit "form_dial()" und "objc_draw()" verwenden.

In alten GEM-Versionen wird außerdem das Objekt so positioniert, daß der linke Rand auf einer Bytegrenze liegt. Wenn man also darauf erpicht ist, daß das Objekt an einer Byte-Grenze beginnt, sollte man dafür sorgen, daß die Objektbreite in Zeichen gerade ist (bei einem acht Pixel breiten Systemzeichensatz).

Deklaration in C:

```
WORD form_center (OBJECT *tree, WORD *pcx, WORD *pcy, WORD *pcw,
                  WORD *pch)
{
     addr_in[0] = tree;
     crys_if (54);
     *pcx = int_out[1];
     *pcy = int_out[2];
     *pcw = int_out[3];
     *pch = int_out[4];
     return int_out[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	54	Opcode für FORM_CENTER
contrl+2	contrl[1]	0	# Einträge in int_in
contrl+4	contrl[2]	5	# Einträge in int_out
contrl+6	contrl[3]	1	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
addr_in	addr_in[0]	tree	
int_out	int_out[0]	Return-Wert (reserviert, immer 1)	
int_out+2	int_out[1]	pcx	

Adresse	Feldelement	Belegung
int_out+4	int_out[2]	pcy
int_out+6	int_out[3]	pcw
int_out+8	int_out[4]	pch

Parameter:

tree:	Anfangsadresse des Objektbaums
pcx, pcy, pcw, pch:	zentrierte Koordinaten

FORM_KEYBD (AES 55)

Nimmt Tastatureingaben in ein Formular vor (siehe auch "objc_edit()"). Dabei wird unter Umständen das Eingabefeld geändert (Cursor-Tasten und "TAB") oder das Default-Objekt selektiert ("RETURN").

Deklaration in C:

```
WORD form_keybd (OBJECT *form, WORD obj, WORD nxt_obj,
                 WORD thechar, WORD *pnxt_obj, WORD *pchar)
{
     addr_in[0] = form;
     int_in[0] = obj;
     int_in[1] = thechar;
     int_in[2] = nxt_obj;
     crys_if (55);
     *pnxt_obj = int_out[1];
     *pchar = int_out[2];
     return int_out[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	55	Opcode für FORM_KEYBD
contrl+2	contrl[1]	3	# Einträge in int_in
contrl+4	contrl[2]	3	# Einträge in int_out
contrl+6	contrl[3]	1	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
addr_in	addr_in[0]	form	
int_in	int_in[0]	obj	
int_in+2	int_in[1]	thechar	
int_in+4	int_in[2]	nxt_obj	
int_out	int_out[0]	Return-Wert	
int_out+2	int_out[1]	pnxt_obj	
int_out+4	int_out[2]	pchar	

Parameter:

form:	Anfangsadresse des Objektbaums
obj:	Objektnummer des aktuellen EDIT-Objektes
nxt_obj:	unbenutzt – auf 0 setzen
thechar:	Eingegebenes Zeichen, das eingetragen werden soll
form_keybd():	0: Exit-Objekt gedrückt
	>0: Dialog noch nicht beendet
pnxtobj:	aktuelles EDIT-Objekt für den nächsten Aufruf (verändert sich, wenn "TAB", Cursor-Tasten oder "RETURN" übergeben wurde)
pchar:	0: Zeichen war Cursortaste, "TAB" oder "RETURN"
	>0: übergebenes Zeichen

FORM_BUTTON (AES 56)

Nimmt Mausknopfeingaben in ein Formular vor (simuliert das Anklicken eines Objekts).

Deklaration in C:

```
WORD form_button (OBJECT *form, WORD obj, WORD clks,
                  WORD *pnxt_obj)
{
     addr_in[0] = form;
     int_in[0] = obj;
     int_in[1] = clks;
     crys_if (56);
     *pnxt_obj = int_out[1];
     return int_out[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	56	Opcode für FORM_BUTTON
contrl+2	contrl[1]	2	# Einträge in int_in
contrl+4	contrl[2]	2	# Einträge in int_out
contrl+6	contrl[3]	1	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
addr_in	addr_in[0]	form	
int_in	int_in[0]	obj	
int_in+2	int_in[1]	clks	
int_out	int_out[0]	Return-Wert	
int_out+2	int_out[1]	pnxt_obj	

Parameter:

form: Anfangsadresse des Objektbaumes
obj: Objekt, das bearbeitet werden soll
clk: Anzahl der zu simulierenden Mausklicks

form_button(): 0: Es wurde zuletzt ein EXIT- oder TOUCHEXIT-Objekt angeklickt
>0: Dialog noch nicht abgeschlossen

pnxt_obj: neues aktuelles Objekt (bei Doppelklick auf ein TOUCHEXIT-Objekt wird Bit 15 gesetzt) oder
0: Das nächste Objekt hat HIDDEN- oder DISABLED-Status oder ist nicht EDITABLE.

GRAF-Funktionen

GRAF_RUBBOX (AES 70)

Stellt auf dem Bildschirm eine "Gummiband-Box" dar. Dabei steht die linke obere Ecke fest, und die rechte untere Ecke folgt den Mausbewegungen des Benutzers, solange die Maustaste festgehalten wird. Daher sollte "graf_rubbox()" auch nur bei gedrückter Maustaste aufgerufen werden.

Zurückgeliefert werden Breite und Höhe des Rechtecks in dem Moment, wenn die Maustaste losgelassen wird.

Deklaration in C:

```
WORD graf_rubbox (WORD xorigin, WORD yorigin, WORD wmin, WORD hmin,
                  WORD *pwend, WORD *phend)
{
     int_in[0] = xorigin;
     int_in[1] = yorigin;
     int_in[2] = wmin;
     int_in[3] = hmin;
     crys_if (70);
     *pwend = int_out[1];
     *phend = int_out[2];
     return int_out[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	70	Opcode für GRAF_RUBBERBOX
contrl+2	contrl[1]	4	# Einträge in int_in
contrl+4	contrl[2]	3	# Einträge in int_out
contrl+6	contrl[3]	0	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	xorigin	
int_in+2	int_in[1]	yorigin	
int_in+4	int_in[2]	wmin	

Adresse	Feldelement	Belegung
int_in+6	int_in[3]	hmin
int_out	int_out[0]	Return-Wert (0: Fehler)
int_out+2	int_out[1]	pwend
int_out+4	int_out[2]	phend

Parameter:

xorigin, yorigin:	Koordinaten der linken oberen Ecke
wmin, hmin:	Kleinste (Anfangs-)Ausdehnung des Rechtecks
pwend, phend:	Ausdehnung des Rechtecks bei Beendigung der Funktion

Bemerkungen
In älteren GEM-Dokumentationen auch häufig als "graf_rubberbox()" bezeichnet!

GRAF_DRAGBOX (AES 71)

Läßt den Benutzer den Umriß eines Rechtecks (outline) innerhalb eines anderen Rechtecks verschieben.

Deklaration in C:

```
WORD graf_dragbox (WORD w, WORD h, WORD sx, WORD sy, WORD xc,
                   WORD yc, WORD wc, WORD hc, WORD *pdx, WORD *pdy)
{
     int_in[0] = w;
     int_in[1] = h;
     int_in[2] = sx;
     int_in[3] = sy;
     int_in[4] = xc;
     int_in[5] = yc;
     int_in[6] = wc;
     int_in[7] = hc;
     crys_if (71);
     *pdx = int_out[1];
     *pdy = int_out[2];
     return int_out[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	71	Opcode für GRAF_DRAGBOX
contrl+2	contrl[1]	8	# Einträge in int_in
contrl+4	contrl[2]	3	# Einträge in int_out
contrl+6	contrl[3]	0	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	w	
int_in+2	int_in[1]	h	
int_in+4	int_in[2]	sx	
int_in+6	int_in[3]	sy	
int_in+8	int_in[4]	xc	
int_in+10	int_in[5]	yc	

Adresse	Feldelement	Belegung
int_in+12	int_in[6]	wc
int_in+14	int_in[7]	hc
int_out	int_out[0]	Return-Wert (0: Fehler)
int_out+2	int_out[1]	pdx
int_out+4	int_out[2]	pdy

Parameter:

w, h:	Maße des zu bewegenden Rechtecks
sx, sy:	Anfangskoordinaten des Rechtecks
xc, yc, wc, hc:	Maße des "äußeren" Rechtecks
pdx, pdy:	Koordinaten des verschobenen Rechtecks bei Loslassen der Maustaste

GRAF_MBOX (AES 72)

Zeichnet eine sich bewegende Box mit konstanter Größe. In älteren GEM-Dokumentationen wird diese Funktion auch häufig als "graf_movebox()" bezeichnet!

Deklaration in C:

```
WORD graf_mbox (WORD w, WORD h, WORD srcx, WORD srcy, WORD dstx,
                WORD dsty)
{
     int_in[0] = w;
     int_in[1] = h;
     int_in[2] = srcx;
     int_in[3] = srcy;
     int_in[4] = dstx;
     int_in[5] = dsty;
     return crys_if (72);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	72	Opcode für GRAF_MBOX
contrl+2	contrl[1]	6	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	0	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	w	
int_in+2	int_in[1]	h	
int_in+4	int_in[2]	srcx	
int_in+6	int_in[3]	srcy	
int_in+8	int_in[4]	dstx	
int_in+10	int_in[5]	dsty	
int_out	int_out[0]	Return-Wert (0: Fehler)	

Parameter:

w, h:	Maße des Rechtecks
stx, sty:	Anfangsposition
dstx, dsty:	Endposition

GRAF_GROWBOX (AES 73)

Zeichnet ein sich bewegendes Rechteck, das sich ausdehnt.

Deklaration in C:

```
WORD graf_growbox (WORD stx, WORD sty, WORD stw, WORD sth,
                   WORD finx, WORD finy, WORD finw, WORD finh)
{
     int_in[0] = stx;
     int_in[1] = sty;
     int_in[2] = stw;
     int_in[3] = sth;
     int_in[4] = finx;
     int_in[5] = finy;
     int_in[6] = finw;
     int_in[7] = finh;
     return crys_if (73);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	73	Opcode für GRAF_GROWBOX
contrl+2	contrl[1]	8	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	0	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	stx	
int_in+2	int_in[1]	sty	
int_in+4	int_in[2]	stw	
int_in+6	int_in[3]	sth	
int_in+8	int_in[4]	finx	
int_in+10	int_in[5]	finy	
int_in+12	int_in[6]	finw	
int_in+14	int_in[7]	finh	
int_out	int_out[0]	Return-Wert (0: Fehler)	

Parameter:

stx, sty, stw, sth:	Anfangsmaße des Rechtecks
finx, finy, finw, finh:	Endmaße des Rechtecks

Bemerkungen
Ab PC-GEM 2.0 wird der Befehl ignoriert.

GRAF_SHRINKBOX (AES 74)

Zeichnet ein sich bewegendes Rechteck, das schrumpft.

Deklaration in C:

```
WORD graf_shrinkbox (WORD finx, WORD finy, WORD finw, WORD finh,
                     WORD stx, WORD sty, WORD stw, WORD sth)
{
     int_in[0] = finx;
     int_in[1] = finy;
     int_in[2] = finw;
     int_in[3] = finh;
     int_in[4] = stx;
     int_in[5] = sty;
     int_in[6] = stw;
     int_in[7] = sth;
     return crys_if (74);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	74	Opcode für GRAF_SHRINKBOX
contrl+2	contrl[1]	8	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	0	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	finx	
int_in+2	int_in[1]	finy	
int_in+4	int_in[2]	finw	
int_in+6	int_in[3]	finh	
int_in+8	int_in[4]	stx	
int_in+10	int_in[5]	sty	
int_in+12	int_in[6]	stw	
int_in+14	int_in[7]	sth	
int_out	int_out[0]	Return-Wert (0: Fehler)	

Parameter:

finx, finy, finw, finh:	Endmaße des Rechtecks
stx, sty, stw, sth:	Anfangsmaße des Rechtecks

Bemerkungen
Wird ab PC-GEM 2.0 ignoriert.

GRAF_WATCHBOX (AES 75)

Der Objektstatus eines Objekts wird in Abhängigkeit von der Position des Mauszeigers gesetzt (je nachdem, ob er sich innerhalb oder außerhalb des Objekts befindet). Die Funktion bricht bei Loslassen des Mausknopfes ab.

Deklaration in C:

```
WORD graf_watchbox (OBJECT *tree, WORD obj, WORD instate,
                    WORD outstate)
{
     addr_in[0] = tree;
     int_in[1] = obj;
     int_in[2] = instate;
     int_in[3] = outstate;
     return crys_if (75);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	75	Opcode für GRAF_WATCHBOX
contrl+2	contrl[1]	4	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	1	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	reserviert	
int_in+2	int_in[1]	obj	
int_in+4	int_in[2]	instate	
int_in+6	int_in[3]	outstate	
addr_in	addr_in[0]	tree	
int_out	int_out[0]	Return-Wert	

Parameter:

obj: Objektnummer des betreffenden Objekts

instate:	Status des Objekts, wenn sich der Mauszeiger innerhalb der Begrenzung befindet. Bitbelegung: NORMAL (0x0000) SELECTED (0x0001) CROSSED (0x0002) CHECKED (0x0004) DISABLED (0x0008) OUTLINED (0x0010) SHADOWED (0x0020)
outstate:	Status des Objekts, wenn sich der Mauszeiger außerhalb der Begrenzung befindet
tree:	Anfangsadresse des Objektbaums
graf_watchbox():	Beim Loslassen des Knopfes war der Mauszeiger 0: außerhalb des Objekts 1: innerhalb des Objekts.

GRAF_SLIDEBOX (AES 76)

Dient zur Abfrage von "Schiebereglern". Dazu verwendet man ein Rechteck, das Child eines anderen Rechtecks in einem Objektbaum ist.

Den Umriß (outline) dieses Objekts kann man dann entweder horizontal oder vertikal innerhalb der Grenzen des übergeordneten Rechtecks bewegen.

Aufruf nur bei gedrückter Maustaste, da beim Loslassen des Knopfes abgebrochen wird.

Deklaration in C:

```
WORD graf_slidebox (OBJECT *tree, WORD parent, WORD obj,
                    WORD isvert)
{
     addr_in[0] = tree;
     int_in[0] = parent;
     int_in[1] = obj;
     int_in[2] = isvert;
     return crys_if (76);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	76	Opcode für GRAF_SLIDEBOX
contrl+2	contrl[1]	3	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	1	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	parent	
int_in+2	int_in[1]	obj	
int_in+4	int_in[2]	isvert	
addr_in	addr_in[0]	tree	
int_out	int_out[0]	Return-Wert	

Parameter:

parent:	Objektnummer des übergeordneten Rechtecks
obj:	Objektnummer des zu verschiebenden Rechtecks (der Schieber)
vh:	0: horizontal verschieben 1: vertikal verschieben
tree:	Anfangsadresse des Objektbaums
graf_slidebox():	relative Position des Schiebers innerhalb des übergeordneten Objekts (0: ganz links bzw. oben, 1000: ganz rechts bzw. unten)

GRAF_HANDLE (AES 77)

Liefert das Handle der Bildschirm-Workstation, auf die die AES ausgeben. Weiterhin erhält man die Ausmaße in Pixeln eines Zeichens im Systemzeichensatz.

Deklaration in C:

```
WORD graf_handle (WORD *pwchar, WORD *phchar, WORD *pwbox,
                  WORD *phbox)
{
     crys_if (77);
     *pwchar = int_out[1];
     *phchar = int_out[2];
     *pwbox = int_out[3];
     *phbox = int_out[4];
     return int_out[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	77	Opcode für GRAF_HANDLE
contrl+2	contrl[1]	0	# Einträge in int_in
contrl+4	contrl[2]	5	# Einträge in int_out
contrl+6	contrl[3]	0	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_out	int_out[0]	handle	
int_out+2	int_out[1]	pwchar	
int_out+4	int_out[2]	phchar	
int_out+6	int_out[3]	pwbox	
int_out+8	int_out[4]	phbox	

Parameter:

graf_handle():	Handle der von den AES geöffneten VDI-Workstation
pwchar, phchar:	Breite und Höhe (in Punkten) eines Zeichens aus dem Systemzeichensatz (für Menüs und Dialoge).
pwbox, phbox:	Maße eines Quadrates, in das ein beliebiges Zeichen aus dem Systemzeichensatz komplett hineinpassen würde. Diese Information wird auch von den AES für die Breiten der einzelnen Fenster-Elemente verwendet!

GRAF_MOUSE (AES 78)

Wählt für den Mauszeiger entweder eine von acht vordefinierten oder eine vom Benutzer definierte Form. Alles was nicht Pfeil oder "BUSYBEE" ist, sollte man nur innerhalb des Arbeitsbereichs des eigenen (obersten) Fensters verwenden!

Deklaration in C:

```
WORD graf_mouse (WORD m_number, MFORM *m_addr)
{
     int_in[0] = m_number;
     addr_in[0] = m_addr;
     return crys_if (78);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	78	Opcode für GRAF_MOUSE
contrl+2	contrl[1]	1	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	1	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	m_number	
addr_in	addr_in[0]	m_addr	
int_out	int_out[0]	Return-Wert (0: Fehler)	

Parameter:

m_number: Nummer der Mausform:

ARROW (0):	Pfeil
TEXT_CRSR (1):	Text-Cursor (vertikaler Strich): Texteingabe
HOURGLASS (2):	oder
BUSYBEE (2):	Biene (Bezeichnung stammt vom PC-GEM, bei dem statt dessen eine Sanduhr gezeigt wird): Computer ist beschäftigt (und fleißig).

POINT_HAND (3): zeigende Hand: Auswahl, Dimensionierung
FLAT_HAND (4): flache Hand mit ausgestreckten Fingern: Auswahl/Positionierung
THIN_CROSS (5): feines Fadenkreuz: Auswahl/Zeichnen
THICK_CROSS (6): breites Fadenkreuz
OUTL_CROSS (7): Umrissenes Fadenkreuz
USER_DEF (255): in faddr adressierte Mausform
M_OFF (256): Mauszeiger ausschalten (Aufrufe können verschachtelt werden!)
M_ON (257): Mauszeiger einschalten

m_addr: Zeiger auf Mausform folgender Gestalt:

```
typedef struct
{
     WORD mf_xhot;          /* X-Pos. Aktionspunkt */
     WORD mf_yhot;          /* Y-Pos. Aktionspunkt */
     WORD mf_nplanes;       /* Anzahl planes (=1) */
     WORD mf_fg;            /* Maskenfarbe (normal: 0) */
     WORD mf_bg;            /* Zeigerfarbe (normal: 1) */
     WORD mf_mask[16];      /* Maskenform */
     WORD mf_data[16];      /* Zeigerform */
} MFORM;
```

GRAF_MKSTATE (AES 79)

Liefert Mausposition sowie Status der Maustasten und der Tastatur.

Deklaration in C:

```
WORD graf_mkstate (WORD *pmx, WORD *pmy, WORD *pmstate,
                   WORD *pkstate)
{
     crys_if (79);
     *pmx = int_out[1];
     *pmy = int_out[2];
     *pmstate = int_out[3];
     *pkstate = int_out[4];
     return int_out[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	79	Opcode für GRAF_MKSTATE
contrl+2	contrl[1]	0	# Einträge in int_in
contrl+4	contrl[2]	5	# Einträge in int_out
contrl+6	contrl[3]	0	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_out	int_out[0]	Return-Wert (reserviert, immer 1)	
int_out+2	int_out[1]	pmx	
int_out+4	int_out[2]	pmy	
int_out+6	int_out[3]	pmstate	
int_out+8	int_out[4]	pkstate	

Parameter:

pmx, pmy:	aktuelle Mausposition	
pmstate:	Status der Maustasten (Bit 0: linke Taste usw.)	
pkstate:	Tastaturstatus. Bitbelegung:	
	K_RSHIFT (0x0001):	rechte Shift-Taste
	K_LSHIFT (0x0002):	linke Shift-Taste
	K_CTRL (0x0004):	Control-Taste
	K_ALT (0x0008):	Alternate-Taste

SCRP-Funktionen

SCRP_READ (AES 80)

Liest den Pfadnamen für das Directory (scrap directory), in dem Daten für die Zwischenablage (clipboard) abgespeichert werden.

Deklaration in C:

```
WORD scrp_read (CHAR *pscrap)
{
     addr_in[0] = pscrap;
     return crys_if (80);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	80	Opcode für SCRP_READ
contrl+2	contrl[1]	0	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	1	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
addr_in	addr_in[0]	pscrap	
int_out	int_out[0]	Return-Wert	

Parameter:

pscrap: Puffer für den Pfadnamen
(Vorsicht! Möglicherweise große Länge bei tief verschachtelten Ordnern!).

scrp_read(): Atari-GEM: 0: Fehler
PC-GEM 2.0: –1: Kein Pfad vorhanden
>=0: Bitvektor, der darüber Auskunft gibt, ob Dateien mit gewissen Extensions gefunden worden sind. Bitbelegung:
Bit 0: SCRAP.CSV
Bit 1: SCRAP.TXT
Bit 2: SCRAP.GEM
Bit 3: SCRAP.IMG
Bit 4: SCRAP.DCA
Bit 5: SCRAP.USR

SCRP_WRITE (AES 81)

Legt den Zugriffspfad für das aktuelle Scrap-Directory fest.

Deklaration in C:

```
WORD scrp_write (CHAR *pscrap)
{
     addr_in[0] = pscrap;
     return crys_if (81);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	81	Opcode für SCRP_WRITE
contrl+2	contrl[1]	0	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	1	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
addr_in	addr_in[0]	pscrap	
int_out	int_out[0]	Return-Wert (0: Fehler)	

Parameter:

pscrap: Name des neuen Pfades

SCRP_CLEAR (AES 82) – nur in PC-GEM ab Version 2.0

Löscht alle Dateien im aktuellen Scrap-Directory.

Deklaration in C:

```
WORD scrp_clear (void)
{
     return crys_if (82);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	82	Opcode für SCRP_CLEAR
contrl+2	contrl[1]	0	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	0	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_out	int_out[0]	Return-Wert (0: Fehler)	

FSEL-Funktionen

FSEL_INPUT (AES 90)

Bringt die handelsübliche Dateiauswahlbox (den File-Selector) auf den Bildschirm.

Alte AES-Versionen (vor GEM 1.4) können maximal 100 Dateien anzeigen und warnen gegebenenfalls mit einem "Ping", wenn Dateien deshalb nicht gezeigt werden können. Daneben gab es auch noch schwerwiegendere Fehler, namentlich bei der Behandlung von BIOS-Fehlern (zum Beispiel: Diskette nicht eingelegt).

Ab GEM 1.4 hat Atari den File-Selector überarbeitet und viele (aber leider nicht alle) Fehler beseitigt. Wichtigste Neuerungen sind die Buttons, mit denen man per Mausklick auf andere Laufwerke wechseln kann und die konfigurierbare Titelzeile (siehe "fsel_exinput()").

Auf dem PC ging die Entwicklung in eine andere Richtung. Zunächst kann man zwischen verschiedenen logischen Laufwerken umschalten, indem man nach Erreichen des Wurzelverzeichnisses nochmals die Close-Box anklickt (man erhält dann eine Liste der vorhandenen Laufwerke – siehe dazu auch "appl_bvset()"). Außerdem kann zusätzlich am Ende des Pfadnamens eine Liste von – durch Kommata getrennten – Suchmasken angegeben werden.

Deklaration in C:

```
WORD fsel_input (CHAR *pipath, CHAR *pisel, WORD *pbutton)
{
     addr_in[0] = pipath;
     addr_in[1] = pisel;
     crys_if (90);
     *pbutton = int_out[1];
     return int_out[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	90	Opcode für FSEL_INPUT
contrl+2	contrl[1]	0	# Einträge in int_in
contrl+4	contrl[2]	2	# Einträge in int_out
contrl+6	contrl[3]	2	# Einträge in addr_in

Adresse	Feldelemente	Belegung
contrl+8	contrl[4]	0 # Einträge in addr_out
addr_in	addr_in[0]	pipath
addr_in+4	addr_in[1]	pisel
int_out	int_out[0]	Return-Wert (0: Fehler)
int_out+2	int_out[1]	pbutton

Parameter:

pipath: Name des standardmäßigen *absoluten* Zugriffspfades inkl. angehängter Suchmaske. Enthält nach dem Aufruf den vom Benutzer geänderten Pfadnamen (also genau den, der auf dem Bildschirm zu sehen ist).

pisel: Vorgewählter Dateiname. Enthält nach dem Aufruf den vom Benutzer ausgewählten Dateinamen.

pbutton: 0: Es wurde "Abbruch" gedrückt.
1: Es wurde "OK" gedrückt.

fsel_input(): Sollte man auf keinen Fall ignorieren, weil beispielsweise bei Speichermangel ein Fehler gemeldet wird!

FSEL_EXINPUT (AES 91) – erst ab GEM 1.4

Entspricht "fsel_input()" mit dem Unterschied, daß zusätzlich eine Zeichenkette als Dialogbox-Titel übergeben werden kann.

Deklaration in C:

```
WORD fsel_exinput (CHAR *pipath, CHAR *pisel, WORD *pbutton,
                   CHAR *plabel)
{
     addr_in[0] = pipath;
     addr_in[1] = pisel;
     addr_in[2] = plabel;
     crys_if (91);
     *pbutton = int_out[1];
     return int_out[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	91	Opcode für FSEL_EXINPUT
contrl+2	contrl[1]	0	# Einträge in int_in
contrl+4	contrl[2]	2	# Einträge in int_out
contrl+6	contrl[3]	3	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
addr_in	addr_in[0]	pipath	
addr_in+4	addr_in[1]	pisel	
addr_in+8	addr_in[2]	plabel	
int_out	int_out[0]	Return-Wert (0: Fehler)	
int_out+2	int_out[1]	pbutton	

Parameter:

pipath: Name des standardmäßigen *absoluten* Zugriffspfades inkl. angehängter Suchmaske. Enthält nach dem Aufruf den vom Benutzer geänderten Pfadnamen (also genau den, der auf dem Bildschirm zu sehen ist).

pisel:	Vorgewählter Dateiname. Enthält nach dem Aufruf den vom Benutzer ausgewählten Dateinamen.
plabel:	Titelzeile
pbutton:	0: Es wurde "Abbruch" gedrückt. 1: Es wurde "OK" gedrückt.
fsel_exinput():	Sollte man auf keinen Fall ignorieren, weil beispielsweise bei Speichermangel ein Fehler gemeldet wird!

Bemerkungen

Und so sollte man das Vorhandensein von "fsel_exinput ()" testen (die AES-Versionen 2.x werden aus Rücksicht auf "ABC-GEM" herausgefiltert):

```
if (((global[0] >= 0x0140) && (global[0] < 0x0200)) ||
     (global[0] >= 0x0300))
{

     /* fsel_exinput verfügbar */
}

else
{
     /* statt dessen fsel_input benutzen */

}
```

WIND-Funktionen

WIND_CREATE (AES 100)

Meldet bei den AES ein neues Fenster an, für das eine entsprechende Fenster-Nummer (window handle) zurückgeliefert wird.

Gleichzeitig werden Fenster-Attribute und maximale Größe festgelegt.

Deklaration in C:

```
WORD wind_create (WORD kind, WORD wx, WORD wy, WORD ww, WORD wh)
{
     int_in[0] = kind;
     int_in[1] = wx;
     int_in[2] = wy;
     int_in[3] = ww;
     int_in[4] = wh;
     return crys_if (100);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	100	Opcode für WIND_CREATE
contrl+2	contrl[1]	5	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	0	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	kind	
int_in+2	int_in[1]	wx	
int_in+4	int_in[2]	wy	
int_in+6	int_in[3]	ww	
int_in+8	int_in[4]	wh	
int_out	int_out[0]	Return-Wert (<0: Fehler)	

Parameter:

kind:	Vorhandene Fensterkomponenten. Bitbelegung:	
	NAME (0x0001):	Titelbalken mit Name
	CLOSE (0x0002):	Schließbox
	FULL (0x0004):	Full-Box
	MOVE (0x0008):	Bewegungsbox (Schattierung im Balken)
	INFO (0x0010):	Infozeile
	SIZE (0x0020):	Größenverstellung
	UPARROW (0x0040):	Pfeil nach oben
	DNARROW (0x0080):	Pfeil nach unten
	VSLIDE (0x0100):	Vertikaler Schieber
	LFARROW (0x0200):	Pfeil nach links
	RTARROW (0x0400):	Pfeil nach rechts
	HSLIDE (0x0800):	Horizontaler Schieber
	HOTCLOSEBOX (0x1000):	Close-Box hat Auto-Repeat (erst ab PC-GEM 2.0)
wx, wy, ww, wh:	Fensterkoordinaten bei größter Ausdehnung	
wind_create():	Fenster-Kennung (window handle) des erzeugten Fensters (<0: Fehler)	

WIND_OPEN (AES 101)

Ein vorher mit "wind_create()" erzeugtes Fenster wird auf dem Bildschirm dargestellt. Titelzeile, Infozeile und Slider müssen *vorher* mit "wind_set()" gesetzt werden!

Deklaration in C:

```
WORD wind_open (WORD handle, WORD wx, WORD wy, WORD ww, WORD wh)
{
     int_in[0] = handle;
     int_in[1] = wx;
     int_in[2] = wy;
     int_in[3] = ww;
     int_in[4] = wh;
     return crys_if (101);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	101	Opcode für WIND_OPEN
contrl+2	contrl[1]	5	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	0	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	handle	
int_in+2	int_in[1]	wx	
int_in+4	int_in[2]	wy	
int_in+6	int_in[3]	ww	
int_in+8	int_in[4]	wh	
int_out	int_out[0]	Return-Wert (0: Fehler)	

Parameter:

handle:	Kennung (handle) des zu öffnenden Fensters
wx, wy, ww, wh:	Anfangsmaße des Fensters

Bemerkungen

Das Öffnen des Fensters führt automatisch dazu, daß der Screen Manager eine "Redraw"-Mitteilung über die Fläche des Arbeitsbereichs verschickt.

WIND_CLOSE (AES 102)

Ist das Gegenstück zu "wind_open()" und schließt das betreffende Fenster. Endgültig gelöscht wird es mit "wind_delete()".

Deklaration in C:

```
WORD wind_close (WORD handle)
{
     int_in[0] = handle;
     return crys_if (102);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	102	Opcode für WIND_CLOSE
contrl+2	contrl[1]	1	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	0	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	handle	
int_out	int_out[0]	Return-Wert (0: Fehler)	

Parameter:

handle: Kennung des zu schließenden Fensters

WIND_DELETE (AES 103)

Löscht ein Fenster komplett und gibt dabei den dafür intern benötigten Speicherplatz und die Kennung (handle) wieder frei.

Deklaration in C:

```
WORD wind_delete (WORD handle)
{
     int_in[0] = handle;
     return crys_if (103);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	103	Opcode für WIND_DELETE
contrl+2	contrl[1]	1	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	0	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	handle	
int_out	int_out[0]	Return-Wert (0: Fehler)	

Parameter:

handle: Kennung des zu löschenden Fensters

WIND_GET (AES 104)

Liefert je nach Funktionsnummer verschiedene Informationen über ein Fenster.

Deklaration in C:

```
WORD wind_get (WORD w_handle, WORD w_field, WORD *pw1, WORD *pw2,
               WORD *pw3, WORD *pw4)
{
     int_in[0] = w_handle;
     int_in[1] = w_field;
     crys_if (104);
     *pw1 = int_out[1];
     *pw2 = int_out[2];
     *pw3 = int_out[3];
     *pw4 = int_out[4];
     return int_out[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	104	Opcode für WIND_GET
contrl+2	contrl[1]	2	# Einträge in int_in
contrl+4	contrl[2]	5	# Einträge in int_out
contrl+6	contrl[3]	0	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	w_handle	
int_in+2	int_in[1]	w_field	
int_out	int_out[0]	Return-Wert (0: Fehler)	
int_out+2	int_out[1]	pw1	
int_out+4	int_out[2]	pw2	
int_out+6	int_out[3]	pw3	
int_out+8	int_out[4]	pw4	

Parameter:

w_handle: Kennung des betreffenden Fensters
w_field: Gewünschte Information. Die Inhalte von pw1... pw4 hängen ab von:

WF_WORKXYWH (4): Berechnet Koordinaten des Arbeitsbereichs des Fensters.
pw1: X-Position
pw2: Y-Position
pw3: Breite
pw4: Höhe

WF_CURRXYWH (5): Berechnet Koordinaten des gesamten Fensters einschließlich Ränder etc.
pw1: X-Position
pw2: Y-Position
pw3: Breite
pw4: Höhe

WF_PREVXYWH (6): Berechnet Gesamtgröße des vorherigen Fensters.
pw1: X-Position
pw2: Y-Position
pw3: Breite
pw4: Höhe

WF_FULLXYWH (7): Berechnet die Gesamtgröße des Fensters in seiner größtmöglichen Ausdehnung.
pw1: X-Position
pw2: Y-Position
pw3: Breite
pw4: Höhe

WF_HSLIDE (8): Liefert Position des horizontalen Schiebers.
pw1: (0: ganz links, 1000: ganz rechts)

WF_VSLIDE (9): Liefert Position des vertikalen Schiebers.
pw1: (0: ganz oben, 1000: ganz unten)

WF_TOP (10): Liefert Kennung des obersten Fensters.
pw1: handle

WF_FIRSTXYWH (11): Liefert Koordinaten des ersten Rechtecks in der Rechteckliste des Fensters.
pw1: X-Position
pw2: Y-Position
pw3: Breite
pw4: Höhe

WF_NEXTXYWH (12): Liefert die Koordinaten des nächsten Rechtecks in der Rechteckliste des Fensters.
pw1: X-Position
pw2: Y-Position
pw3: Breite
pw4: Höhe

WF_HSLSIZE (15): Die Größe des horizontalen Schiebers relativ zum Balken wird berechnet.
pw1: –1: Minimalgröße
1–1000: klein–gesamte Breite

WF_VSLSIZE (16): Die Größe des vertikalen Schiebers relativ zum Balken wird berechnet.
pw1: –1: Minimalgröße
1–1000: klein–gesamte Höhe

WF_SCREEN (17): Adresse und Länge des "Quarter Screen Buffers" für Drop-Down-Menüs und Alarmboxen wird berechnet. Bei TOS 1.02 wird für die Länge 0 zurückgeliefert (obwohl der Puffer 8000 Bytes faßt).
pw1: High-Word der Adresse
pw2: Low-Word der Adresse
pw3: High-Word der Länge
pw4: Low-Word der Länge

Bemerkungen
Einmal mehr hat Digital Research hier die eingeführten Bezeichnungen geändert (nunmehr "WF_WXYWH (4)", "WF_CXYWH (5)", "WF_PXYWH (6)" und "WF_FYXWH (7)").

Eine Beispielfunktion für "WF_SCREEN":

```
void getQSB (void **where, LONG *length)
{
     WORD w1, w2, w3, w4;

     /* Hinweis: bei älteren TC-Versionen ist das Binding für
        wind_get (..., WF_SCREEN, ...) nicht korrekt und liefert
        falsche Resultate! */

     wind_get (0, WF_SCREEN, &w1, &w2, &w3, &w4);
     *where = (void *)((UWORD) w1 * 65536L + (UWORD) w2);
     *length = (UWORD) w3 * 65536L + (UWORD) w4;

     /* Sonderfall: TOS 1.02 */

     if ((*length == 0L) && (global[0] == 0x0120))
          *length = 8000L;
}
```

Und der Aufruf im Programm:

```
void *where;
LONG length;
getQSB (&where, &length);
printf ("Adresse: %p, Länge: %ld Bytes\n", where, length);
```

WIND_SET (AES 105)

Je nach Funktionsnummer werden verschiedene Fenstereigenschaften verändert.

Deklaration in C:

```
WORD wind_set (WORD w_handle, WORD w_field, WORD w1, WORD w2,
               WORD w3, WORD w4)
{
     int_in[0] = w_handle;
     int_in[1] = w_field;
     int_in[2] = w1;
     int_in[3] = w2;
     int_in[4] = w3;
     int_in[5] = w4;
     return crys_if (105);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	105	Opcode für WIND_SET
contrl+2	contrl[1]	6	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	0	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	handle	
int_in+2	int_in[1]	field	
int_in+4	int_in[2]	w1	
int_in+6	int_in[3]	w2	
int_in+8	int_in[4]	w3	
int_in+10	int_in[5]	w4	
int_out	int_out[0]	Return-Wert (0: Fehler)	

Parameter:

w_handle: Kennung des Fensters
w_field: Funktionsnummer. w1...w4 werden in Abhängigkeit davon interpretiert:

WF_NAME (2): Legt neuen Fensternamen fest.
w1 und w2: Zeiger auf Zeichenkette (GEM benutzt nur den Zeiger, legt also *keine* Kopie an).

WF_INFO (3): Legt neue Info-Zeile fest.
w1 und w2: Zeiger auf Zeichenkette (GEM benutzt nur den Zeiger, legt also *keine* Kopie an).

WF_CURRXYWH (5): Setzt Fenstergröße fest.
w1: X-Position
w2: Y-Position
w3: Breite
w4: Höhe

WF_HSLIDE (8): Setzt neue Position für den horizontalen Schieber
w1: (0: ganz links, 1000: ganz rechts)

WF_VSLIDE (9): Setzt neue Position für den vertikalen Schieber
w1: (0: ganz oben, 1000: ganz unten)

WF_TOP (10): Fenster wird zum aktuellen (obersten) Fenster

WF_NEWDESK (14): Setzt neuen Default-Objektbaum; funktioniert *nur* mit Fenster 0 (Hintergrundfenster). Ein Nullzeiger erlaubt es, wieder den Standardhintergrund zu setzen (kann *nach* der Ausführung von GEM-Programmen aus eigenen Programmen heraus wichtig sein).
w1: Adresse des Objektbaums (High)
w2: Adresse des Objektbaums (Low)
w3: Objektnummer des ersten darzustellenden Objekts

WF_HSLSIZE (15): Setzt relative Größe des horizontalen Schiebers
w1: –1 (Minimalgröße)
1–1000 (klein–volle Breite)

WF_VSLSIZE (16): Setzt relative Größe des vertikalen Schiebers
w1: –1 (Minimalgröße)
1–1000 (klein–volle Höhe)

in PC-GEM 2.0 neu hinzugekommen:

WF_TATTRB (18): Setzt den "Fenster-Attributvektor" (von Digital Research nicht weiter dokumentiert).
w1: 0: Fenster ist oberstes Fenster
1: Fenster ist nicht oberstes Fenster

WF_SIZTOP (19): Macht das angegebene Fenster zum aktiven, ohne die Reihenfolge der darunterliegenden zu ändern. Gleichzeitig können neue Position und Größe angegeben werden:
w1: X-Position
w2: Y-Position

w3: Breite
w4: Höhe

hingegen in Atari-GEM ab Version 3.0:

WF_COLOR (18): Farben der verschiedenen Fensterelemente festlegen:
w1: Fensterelement:
W_BOX (0): Fensterhintergrund (G_IBOX)
W_TITLE (1): Titelbox, enthält Namen, Volle-Größe-Box und Schließbox (G_BOX)
W_CLOSER (2): Schließbox (G_BOXCHAR)
W_NAME (3): Namenszeile (G_BOXTEXT)
W_FULLER (4): Volle-Größe-Box (G_BOXCHAR)
W_INFO (5): Infozeile (G_BOXTEXT)
W_DATA (6): Objekt, das die restlichen Fensterelemente enthält (G_IBOX)
W_WORK (7): Arbeitsbereich des Fensters, freie Fläche (G_IBOX)
W_SIZER (8): Größenveränderungsbox (G_BOXCHAR)
W_VBAR (9): enthält die Elemente des vertikalen Balkens (G_BOX)
W_UPARROW (10): Auf-Pfeil (G_BOXCHAR)
W_DNARROW (11): Ab-Pfeil (G_BOXCHAR)
W_VSLIDE (12): Hintergrund für den vertikalen Slider (G_BOX)
W_VELEV (13): Vertikaler Slider (G_BOX)
W_HBAR (14): enthält die Elemente des horizontalen Balkens (G_BOX)
W_LFARROW (15): Links-Pfeil (G_BOXCHAR)
W_RTARROW (16): Rechts-Pfeil (G_BOXCHAR)
W_HSLIDE (17): Hintergrund für den horizontalen Slider (G_BOX)
W_HELEV (18): Horizontaler Slider (G_BOX)
w2: Farbe für aktive Fenster (Farbwert wie etwa das untere Wort des "ob_spec" bei einem G_BOX-Objekt) oder –1 (nicht setzen)
w3: Farbe für inaktive Fenster (Farbwert wie etwa das untere Wort des "ob_spec" bei einem G_BOX-Objekt) oder –1 (nicht setzen)

WF_DCOLOR (19): Wie "WF_COLOR", nur werden die Standardfarben für neu angelegte Fenster gesetzt (diese Aufgabe wird normalerweise von XCONTROL übernommen).

WIND_FIND (AES 106)

Ermittelt die Kennung des an einer bestimmten Koordinate befindlichen Fensters.

Deklaration in C:

```
WORD wind_find (WORD mx, WORD my)
{
     int_in[0] = mx;
     int_in[1] = my;
     return crys_if (106);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	106	Opcode für WIND_FIND
contrl+2	contrl[1]	2	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	0	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	mx	
int_in+2	int_in[1]	my	
int_out	int_out[0]	Return-Wert	

Parameter:

mx, my: Bildschirmkoordinaten
wind_find(): Kennung des gefundenen Fensters (–1: kein Fenster)

WIND_UPDATE (AES 107)

Diese Funktion dient dazu, Bildschirmzugriffe verschiedener AES-Prozesse zu synchronisieren.

BEG_UPDATE/END_UPDATE

Normalerweise ist jeder AES-Prozeß für die Inhalte der Arbeitsbereiche der von ihm geöffneten Fenster zuständig – der Rest des Bildschirms wird entweder vom Screen Manager oder anderen AES-Prozessen (zum Beispiel Accessories) verwaltet.

Die Lage ändert sich beispielsweise, wenn man eine Dialogbox zeichnen will (generell: wenn man unter Umgehung der AES-Fensterfunktionen auf den Bildschirm zugreifen will, also auch beispielsweise beim Wiederherstellen eines Bildschirmausschnitts). In einem solchen Moment dürfen sich die von den Fenstern belegten Bildschirmflächen (mithin die Rechtecklisten) nicht verändern – für wie lange, hält von der entsprechenden Operation ab.

Mit "wind_update (BEG_UPDATE)" kann also ein Programm anmelden, daß es direkt auf dem Bildschirm ausgeben möchte, ohne dabei von anderen AES-Prozessen gestört zu werden. Die AES gehen dann so vor: Falls schon ein anderer Prozeß "wind_update (BEG_UPDATE)" ausgeführt hat, wird das betreffende Programm so lange blockiert, bis der Bildschirm wieder frei ist. Anderenfalls erhält man die Kontrolle zurück und darf bis zum "wind_update (END_UPDATE)" mit dem Bildschirm alles tun, was man möchte – Hauptsache, am Ende sind die Inhalte "fremder" Fenster wieder hergestellt.

"wind_update()" dient mithin als *Semaphore*, die dafür sorgt, daß nur einer gleichzeitig auf dem Bildschirm malt. Das AES-Prozeßswitching wird dabei *nicht* unterbunden – es werden nur solche Prozesse blockiert, die selbst auch auf dem Bildschirm ausgeben wollen (und dies korrekt anmelden). Die Rechtecklisten der Fenster sind damit eingefroren (so daß man mit "wind_get()" in aller Ruhe die Liste der sichtbaren Rechtecke abrufen kann, ohne daß jemand anders dazwischenfunkt). Bei den Fensterfunktionen weiß GEM natürlich selbst Bescheid, daß sie zur Veränderung der Rechtecklisten führen können, und behandelt sie analog. Wann muß man also "wind_update (BEG_UPDATE)" benutzen? Immer dann, wenn man irgendeine Änderung des Bildschirminhalts herbeiführt, ohne dabei die AES-Fensterfunktionen zu benutzen.

BEG_MCTRL/END_MCTRL

Die Verteilung der Mausereignisse und Tastaturereignisse wird folgendermaßen vorgenommen: Mausereignisse gehen an den Prozeß, dem das betroffene Fenster gehört – vorausgesetzt,

ihm gehört auch das oberste Fenster. Ansonsten werden sie vom Screen Manager behandelt (zum Beispiel: Klick mit linker Maustaste auf nicht aktives Fenster führt zu WM_TOPPED-Mitteilung). Tastaturereignisse werden immer an den Prozeß, dem das aktive Fenster gehört, gemeldet. Dabei wird gegebenenfalls Hintergrundfenster 0 beachtet (falls kein anderes Fenster geöffnet ist).

Mit "wind_update (BEG_MCTRL)" kann man die Mauskontrolle selbst übernehmen. Diese Funktion wird immer dann benötigt, wenn Tastatur- oder Mauseingaben unabhängig von den geöffneten Fenstern bearbeitet werden sollen. Ein Beispiel dafür sind alle Funktion der GRAF-Bibliothek, insbesondere "graf_slidebox()" und auch die Dateiauswahlbox (die intern "graf_slidebox()" aufruft).

Deklaration in C:

```
WORD wind_update (WORD beg_update)
{
     int_in[0] = beg_update;
     return crys_if (107);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung
contrl	contrl[0]	107 Opcode für WIND_UPDATE
contrl+2	contrl[1]	1 # Einträge in int_in
contrl+4	contrl[2]	1 # Einträge in int_out
contrl+6	contrl[3]	0 # Einträge in addr_in
contrl+8	contrl[4]	0 # Einträge in addr_out
int_in	int_in[0]	beg_update
int_out	int_out[0]	Return-Wert (0: Fehler)

Parameter:

beg_update: Funktionsnummer:
END_UPDATE (0): Bildschirmaufbau abgeschlossen.
BEG_UPDATE (1): Bildschirmaufbau beginnt;
Rechtecklisten werden eingefroren.
END_MCTRL (2): Applikation gibt Mauskontrolle wieder ab.
BEG_MCTRL (3): Applikation übernimmt totale Kontrolle über die Maus – insbesondere die Drop-Down-Menüs sind gesperrt.

WIND_CALC (AES 108)

Berechnet entweder aus der Größe des Arbeitsbereiches eines Fensters die Gesamtmaße oder umgekehrt.

Deklaration in C:

```
WORD wind_calc (WORD wctype, WORD kind, WORD x, WORD y, WORD w,
                WORD h, WORD *px, WORD *py, WORD *pw, WORD *ph)
{
     int_in[0] = wctype;
     int_in[1] = kind;
     int_in[2] = x;
     int_in[3] = y;
     int_in[4] = w;
     int_in[5] = h;
     crys_if (108);
     *px = int_out[1];
     *py = int_out[2];
     *pw = int_out[3];
     *ph = int_out[4];
     return int_out[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	108	Opcode für WIND_CALC
contrl+2	contrl[1]	6	# Einträge in int_in
contrl+4	contrl[2]	5	# Einträge in int_out
contrl+6	contrl[3]	0	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	wctype	
int_in+2	int_in[1]	kind	
int_in+4	int_in[2]	x	
int_in+6	int_in[3]	y	
int_in+8	int_in[4]	w	
int_in+10	int_in[5]	h	

Adresse	Feldelement	Belegung
int_out	int_out[0]	Return-Wert (0: Fehler)
int_out+2	int_out[1]	px
int_out+4	int_out[2]	py
int_out+6	int_out[3]	pw
int_out+8	int_out[4]	ph

Parameter:

wctype:	WC_BORDER (0):	errechne Gesamtmaße
	WC_WORK (1):	errechne Arbeitsbereich
kind:	Komponenten des Fensters:	
	NAME (0x0001):	Titelbalken mit Name
	CLOSE (0x0002):	Schließbox
	FULL (0x0004):	Full-Box
	MOVE (0x0008):	Bewegungsbox (Schattierung im Balken)
	INFO (0x0010):	Info-Zeile
	SIZE (0x0020):	Größen-Verstellung
	UPARROW (0x0040):	Pfeil nach oben
	DNARROW (0x0080):	Pfeil nach unten
	VSLIDE (0x0100):	vertikaler Schieber
	LFARROW (0x0200):	Pfeil nach links
	RTARROW (0x0400):	Pfeil nach rechts
	HSLIDE (0x0800):	horizontaler Schieber
x, y,w, h:	Bekannte Koordinaten	
px, py, pw, ph:	Errechnete Koordinaten	

Bemerkungen

Man sollte *niemals* davon ausgehen, daß Fensterelemente eine bestimmte Größe oder Position haben, sondern statt dessen *immer* "wind_calc()" benutzen – dazu ist es schließlich da. Ein Beispiel: Es ist ohne weiteres denkbar, daß ein alternativer Bildschirmmanager alle Fensterränder breiter als "normal" darstellt.

WIND_NEW (AES 109) – erst ab GEM 1.4

Schließt und löscht alle Fenster (auch die der Accessories!) und setzt die mit "wind_update()" gesetzten Blockierungen zurück.

Deklaration in C:

```
void wind_new (void)
{
     crys_if (109);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	109	Opcode für WIND_NEW
contrl+2	contrl[1]	0	# Einträge in int_in
contrl+4	contrl[2]	0	# Einträge in int_out
contrl+6	contrl[3]	0	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out

RSRC-Funktionen

RSRC_LOAD (AES 110)

Dient zum Laden und Initialisieren einer Resource-Datei. Dabei geschieht folgendes:

1. Die Datei wird mittels "shel_find()" gesucht. Gegebenenfalls wird ein Fehler gemeldet.
2. Aus dem Dateikopf wird die Länge der Datei ermittelt (siehe Beschreibung des RSC-Formats).
3. Genausoviel Speicher wird mit "Malloc()" alloziert. Falls nicht genügend Speicher da ist, wird ein Fehler gemeldet.
4. Anfangsadresse und Länge in Bytes werden im global-Feld festgehalten (um dann für "rsrc_free()" bereitzustehen; dies ist von Atari *nicht* offiziell dokumentiert).
5. Die Datei wird geöffnet, in voller Länge an die entsprechende Adresse geladen und wieder geschlossen.
6. Nun werden noch einige Initialisierungen vorgenommen (Setzen der internen Zeiger, Anpassung der Koordinaten wie bei "rsrc_obfix()", Setzen der Tabelle der Objektbaumzeiger, Setzen von global[5,6]).

Was an dieser Stelle nicht passiert: die Umwandlung von Bitimages und Icons vom Standardformat in das geräteabhängige Format!

Deklaration in C:

```
WORD rsrc_load (const CHAR *rsname)
{
     addr_in[0] = rsname;
     return crys_if (110);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	110	Opcode für RSRC_LOAD
contrl+2	contrl[1]	0	# Einträge in int_in

Adresse	Feldelement	Belegung
contrl+4	contrl[2]	1 # Einträge in int_out
contrl+6	contrl[3]	1 # Einträge in addr_in
contrl+8	contrl[4]	0 # Einträge in addr_out
addr_in	addr_in[0]	rsname
int_out	int_out[0]	Return-Wert (0: Fehler)

Parameter:

rsname: Dateiname der Resource-Datei (Datei wird über den Standardzugriffspfad des AES gesucht)

Bemerkungen

Ein Fehler kann verschiedene Ursachen haben: Speichermangel oder Resource-Datei nicht gefunden.

RSRC_FREE (AES 111)

Gibt den durch "rsrc_load()" reservierten Speicherbereich wieder frei (sollte man am Ende eines Programmes auf gar keinen Fall vergessen).

Deklaration in C:

```
WORD rsrc_free (void)
{
     return crys_if (111);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	111	Opcode für RSRC_FREE
contrl+2	contrl[1]	0	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	0	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_out	int_out[0]	Return-Wert (0: Fehler)	

RSRC_GADDR (AES 112)

Ermittelt die Anfangsadresse einer Resource-Struktur im Speicher (nach "rsrc_load()").

Deklaration in C:

```
WORD rsrc_gaddr (WORD rstype, WORD rsid, void *paddr)
{
     int_in[0] = rstype;
     int_in[1] = rsid;
     control[4] = 1;
     crys_if (112);
     control[4] = 0;
     *paddr = addr_out[0];
     return int_out[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	112	Opcode für RSRC_GADDR
contrl+2	contrl[1]	2	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	0	# Einträge in addr_in
contrl+8	contrl[4]	1	# Einträge in addr_out
int_in	int_in[0]	rstype	
int_in+2	int_in[1]	rsid	
int_out	int_out[0]	Return-Wert (0: Fehler)	
addr_out	addr_out[0]	paddr	

Parameter:

rsid: Index (i) der gesuchten Struktur
rstype: Typ der gesuchten Struktur:
R_TREE (0): Zeiger auf Objektbaum i
R_OBJECT (1): Zeiger auf Objektstruktur i

R_TEDINFO (2):	Zeiger auf Textobjekt i
R_ICONBLK (3):	Zeiger auf Icon i
R_BITBLK (4):	Zeiger auf Bit-Block i
R_STRING (5):	Zeiger auf String i
R_IMAGEDATA (6):	Zeiger auf Bilddaten Nr. i
R_OBSPEC (7):	Zeiger auf TEDINFO-Struktur
R_TEPTEXT (8):	Zeiger auf Text
R_TEPTMPLT (9):	Zeiger auf Textmaske
R_TEPVALID (10):	Zeiger auf Textschablone
R_IBPMASK (11):	Zeiger auf Icon-Maske
R_IBPDATA (12):	Zeiger auf Icon-Daten
R_IBPTEXT (13):	Zeiger auf Icon-Text
R_BIPDATA (14):	Zeiger auf Bitmuster-Daten
R_FRSTR (15):	Adresse eines Zeigers auf freien String
R_FRIMG (16):	Adresse eines Zeigers auf freies Image

paddr: Anfangsadresse der gesuchten Struktur

Bemerkungen

Wenn der Textzeiger innerhalb einer TEDINFO-Struktur gesucht wird, muß selbstverständlich nicht die Nummer des zugehörigen Objekts, sondern die Nummer der TEDINFO-Struktur angegeben werden. Ähnliches gilt für die meisten anderen Codes.

RSRC_SADDR (AES 113)

Speichert die Anfangsadresse einer Datenstruktur im Speicher.

Deklaration in C:

```
WORD rsrc_saddr (WORD rstype, WORD rsid, void *lngval)
{
     int_in[0] = rstype;
     int_in[1] = rsid;
     addr_in[0] = lngval;
     return crys_if (113);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	113	Opcode für RSRC_SADDR
contrl+2	contrl[1]	2	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	1	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	rstype	
int_in+2	int_in[1]	rsid	
addr_in	addr_in[0]	lngval	
int_out	int_out[0]	Return-Wert (0: Fehler)	

Parameter:

rstype: Typ der Datenstruktur:
R_FRSTR (15): Adresse eines Zeigers auf freien String
R_FRIMG (16): Adresse eines Zeigers auf freies Image
rsid: Position (i) in der Datenstruktur, in die lngval geschrieben wird
lngval: die abzuspeichernde Adresse

RSRC_OBFIX (AES 114)

Wandelt in einem Objekt die Koordinatendarstellung von Zeichen- in Pixeldarstellung. Dies ist immer dann vonnöten, wenn man Objekte nicht mit "rsrc_load()" lädt, sondern direkt in das Programm eingliedert oder zur Laufzeit erzeugt.

Dabei wird jeweils das untere Byte der Koordinatenangabe mit der Größe des Systemzeichensatzes multipliziert und darauf das (vorzeichenbehaftete) obere Byte addiert. *Ausnahme*: Für eine Breite von genau 80 Zeichen wird die Breite des Bildschirms eingesetzt (was das Resource Construction Set beim Hintergrundobjekt des Menübaums nutzt).

Deklaration in C:

```
WORD rsrc_obfix (OBJECT *tree, WORD obj)
{
     addr_in[0] = tree;
     int_in[0] = obj;
     return crys_if (114);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	114	Opcode für RSRC_OBFIX
contrl+2	contrl[1]	1	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	1	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	obj	
addr_in	addr_in[0]	tree	
int_out	int_out[0]	Return-Wert (reserviert, immer 1)	

Parameter:

obj: Objektnummer des zu konvertierenden Objekts
tree: Anfangsadresse des Objektbaums

SHEL-Funktionen

SHEL_READ (AES 120)

Liefert Namen und Kommandozeile, mit der die Applikation aufgerufen wurde. Dies klappt allerdings nur, wenn der Programmstart mit "shel_write()" vorgenommen wurde. Also im Zweifelsfall besser die Werte aus der Basepage verwenden.

Deklaration in C:

```
WORD shel_read (CHAR *pcmd, CHAR *ptail)
{
     addr_in[0] = pcmd;
     addr_in[1] = ptail;
     return crys_if (120);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	120	Opcode für SHEL_READ
contrl+2	contrl[1]	0	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	2	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
addr_in	addr_in[0]	pcmd	
addr_in+4	addr_in[1]	ptail	
int_out	int_out[0]	Return-Wert (0: Fehler)	

Parameter:

pcmd: Programmname mit Pfad (wie beim Aufruf mit "Pexec()")
ptail: Kommandozeile (wie bei "Pexec()" – also nullterminiert und mit Längenangabe im ersten Byte)

Bemerkungen
Beide Puffer sollten mindestens 128 Bytes lang sein!

SHEL_WRITE (AES 121)

Informiert die AES, daß eine andere Applikation gestartet werden soll.

Im Gegensatz zum GEMDOS-Befehl "Pexec()" bleibt dabei das laufende Programm nicht resident.

Ein Beispiel für die Verwendung dieser Funktion ist das Starten von "OUTPUT.APP" durch das Präsentationsgrafikprogramm "SciGraph".

Deklaration in C:

```
WORD shel_write (WORD doex, WORD isgr, WORD isover,
                 CHAR *pcmd, CHAR *ptail)
{
     int_in[0] = doex;
     int_in[1] = isgr;
     int_in[2] = isover;
     addr_in[0] = pcmd;
     addr_in[1] = ptail;
     return crys_if (121);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	121	Opcode für SHEL_WRITE
contrl+2	contrl[1]	3	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	2	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	doex	
int_in+2	int_in[1]	isgr	
int_in+4	int_in[2]	isover	
addr_in	addr_in[0]	pcmd	
addr_in+4	addr_in[1]	ptail	
int_out	int_out[0]	Return-Wert (0: Fehler)	

Parameter:

doex: 0: keine weitere Applikation starten. Vor GEM 1.4 wurden dabei die AES komplett neu initialisiert (inkl. Laden der Accessories). Ab GEM 1.4 wird normal zum Desktop zurückgekehrt.
1: neues Programm laden

isgr: 0: keine Grafik-Applikation (Cursor wird ein-, Maus wird ausgeschaltet; Bildschirm wird gelöscht)
1: Grafik-Applikation

isover: in Atari-GEM unbenutzt (sollte auf 0 gesetzt werden). Unter PC-GEM läßt sich damit einstellen, ob das Programm als Overlay (0) oder erst nach Verlassen des aktuellen Programms (1) gestartet werden soll; Modus 2 erlaubt das Nachstarten unter Entfernung von GEM aus dem Speicher.

pcmd: Adresse eines Strings mit dem Namen des zu startenden Programms

ptail: Adresse eines Strings mit der Kommandozeile
(Vorsicht: genau wie bei "Pexec()" gibt das erste Byte die Länge der folgenden Zeichenkette an.)

Bemerkungen

Beim Start von Programmen mittels "shel_write()" wird auch – je nach Applikationstyp – der entsprechende Critical-Error-Handler (siehe "etv_critic") initialisiert. Wird eine TOS-Applikation von einem GEM-Programm mittels "Pexec()" gestartet, braucht man sich also nicht darüber zu wundern, daß zwar Alertboxen erscheinen, aber keine Maus auftaucht.

SHEL_GET (AES 122)

Dient zum Lesen von Zeichen aus dem GEM-internen Environment-Speicher. Das Desktop benutzt diesen Speicher zur Aufbewahrung der DESKTOP.INF-bzw. NEWDESK.INF-Datei (terminiert durch ASCII 26 (CTRL-Z) oder durch Null). Das Format dieser Datei ist allerdings nicht offiziell dokumentiert und kann sich ändern.

Deklaration in C:

```
WORD shel_get (CHAR *addr, WORD len)
{
     addr_in[0] = addr;
     int_in[0] = len;
     return crys_if (122);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	122	Opcode für SHEL_GET
contrl+2	contrl[1]	1	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	1	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	len	
addr_in	addr_in[0]	addr	
int_out	int_out[0]	Return-Wert (0: Fehler)	

Parameter:

len: Anzahl der zu lesenden Bytes
addr: Anfangsadresse des Zielspeichers

SHEL_PUT (AES 123)

Schreibt eine gegebene Anzahl von Zeichen in die GEM-internen Environment-Speicher. In älteren TOS-Versionen ist die Länge dieses Puffers auf 1024 Bytes beschränkt, ab GEM 1.4 sollen 4096 Bytes hineinpassen. Da das Format dieses Puffers sowieso nicht offiziell dokumentiert ist, sollte man sich nicht den Kopf darüber zerbrechen...

Deklaration in C:

```
WORD shel_put (char *addr, WORD len)
{
     addr_in[0] = addr;
     int_in[0] = len;
     return crys_if (123));
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	123	Opcode für SHEL_PUT
contrl+2	contrl[1]	1	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	1	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	len	
addr_in	addr_in[0]	addr	
int_out	int_out[0]	Return-Wert (0: Fehler)	

Parameter:

len: Anzahl der zu schreibenden Zeichen
addr: Anfangsadresse des Puffers mit den Zeichen

SHEL_FIND (AES 124)

Sucht eine Datei im aktuellen Verzeichnis und im Wurzelverzeichnis. Zusätzlich werden alle Verzeichnisse durchsucht, die in der AES-Environment-Variablen "PATH=" angegeben sind. Im Normalfall handelt es sich dabei um das Wurzelverzeichnis des Bootlaufwerks und das aktuelle Verzeichnis.

Ab GEM 1.4 wird auch der Pfad untersucht, in dem sich das gerade laufende Programm befindet (und der vor dem Programmnamen stehen sollte, den man mit "shel_read()" erfragen kann). Diese Funktion wird übrigens auch von der AES-Funktion "rsrc_load()" verwendet, um die Resource-Datei zu finden.

Deklaration in C:

```
WORD shel_find (char *ppath)
{
     addr_in[0] = ppath;
     return crys_if (124);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	124	Opcode für SHEL_FIND
contrl+2	contrl[1]	0	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	1	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
addr_in	addr_in[0]	ppath	
int_out	int_out[0]	Return-Wert (0: Fehler)	

Parameter:

ppath: Adresse eines Strings mit dem Namen der gesuchten Datei. Nach Aufruf enthält er eine Dateispezifikation, wie sie von dem aktuellen Verzeichnis aus benutzt werden kann. Diese Dateispezifikation *kann* natürlich auch einen vollständigen absoluten Pfad enthalten, daher sollte eine genügende Länge (mindestens 128 Zeichen!) gewählt werden.

SHEL_ENVRN (AES 125)

Was sind eigentlich die AES-Environment-Variablen, und wofür kann man sie einsetzen?

Die Antwort ist wie so oft ganz einfach: Es sind tatsächlich nur die Environment-Variablen des Programms, das die AES beim Booten des Rechners aufgerufen hat.

Genau wie bei den Environment-Variablen eines "normalen" GEMDOS-Programms handelt es sich um eine Liste von ASCII-Strings, in denen die verschiedenen Variablennamen und deren Werte festgehalten sind.

Wie so oft bei Environment-Strings ist die sogenannte "PATH"-Variable die wichtigste. Sie gibt einfach eine Liste von Verzeichnissen an, in denen nach bestimmten Dateien gesucht werden soll. Als Trennzeichen dienen Kommata (ab AES-Version 1.4) und Strichpunkte (schon immer). Auch leere Einträge sind erlaubt und stehen für das aktuelle Verzeichnis (";C:\" steht also für das Wurzelverzeichnis von C: *und* das aktuelle Verzeichnis).

Nun kann man natürlich außer der Pfadvariablen noch frei nach Gutdünken weitere Variablen setzen. Diese Variablen kann man dann nicht nur mit der AES-Funktion "shel_envrn()" abfragen, sondern sie werden auch vom Desktop an alle gestarteten Programme als normale GEMDOS-Environment-Variablen weitergereicht.

Naheliegend ist es beispielsweise, den Standardsuchpfad um einen Ordner zu erweitern, in dem man alle Resource-Dateien sammelt. Damit werden die Verzeichnisse auf der Festplatte deutlich übersichtlicher. Nachteil: Leider vergißt man zu oft beim Weitergeben von Programmen den Resource-File.

Jetzt ein Beispiel für das Hinzufügen neuer Environment-Variablen: Turbo-C beispielsweise sucht beim Öffnen seiner Help-Datei nach der Environment-Variablen "TC". Falls vorhanden, wird die Help-Datei im darin angegebenen Ordner gesucht. Damit ist es nicht notwendig, die Help-Datei im gleichen Ordner wie Turbo-C zu installieren.

Das BIOS installiert normalerweise als Standardpfad einfach das Wurzelverzeichnis des Bootlaufwerks. Dazu verwendet es die Systemvariable "_bootdev" und begeht dabei unglücklicherweise einen seit langem bekannten Fehler: es betrachtet die Systemvariable als Byte statt als Wort und greift damit auf die falsche Speicherzelle zu.

Resultat: Als Standardpfad wird "A:\" eingetragen – ungeachtet, von welchem Laufwerk man gebootet hat. Die meisten Harddisktreiber von Fremdherstellern umschiffen dieses Problem auf die eine oder andere Weise.

Und dann ist da noch ein anderes Problem: Leider setzt das BIOS das Environment auf eine andere Art und Weise als der ganze Rest der Computerwelt. Normalerweise handelt es sich bei den Environmentstrings um nullterminierte Zeichenketten der Form "VARIABLE=WERT". Das Ende wird demzufolge durch eine doppelte Null angegeben. Anders beim BIOS, das leider hinter der Zeichenkette "PATH=" eine Null einfügt.

Von Atari dokumentierte Abhilfe: Wenn der für "PATH=" zurückgelieferte Zeiger auf ein *Nullbyte* zeigt, sollte man ihn um eins erhöhen, um an das richtige Ergebnis zu kommen.

Deklaration in C:

```
WORD shel_envrn (char *ppath, char *psrch)
{
     addr_in[0] = ppath;
     addr_in[1] = psrch;
     return crys_if (125);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	125	Opcode für SHEL_ENVRN
contrl+2	contrl[1]	0	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	2	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
addr_in	addr_in[0]	ppath	
addr_in+4	addr_in[1]	psrch	
int_out	int_out[0]	reserviert (immer 1)	

Bemerkungen

Um das AES-Environment zu ändern, hängt man sich am besten in den Systemvektor "exec_os", über den GEM gestartet wird. In der aufgerufenen Routine liegt – wie bei einem Programm – der Basepage-Zeiger auf dem Stack. In die dort angegebene Basepage setzt man dann einfach den Zeiger auf das neue Environment ein (vor GEM 1.4 wurden freilich nur die ersten 50 Bytes übernommen!).

SHEL_RDEF (AES 126) – nur in PC-GEM ab Version 2.0

Hiermit kann man abfragen, welches Programm nach Beendigung des aktuellen gestartet wird (sollte im allgemeinen das Desktop sein).

Deklaration in C:

```
WORD shel_rdef (CHAR *lpcmd, CHAR *lpdir)
{
     addr_in[0] = lpcmd;
     addr_in[1] = lpdir;
     return crys_if (126);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	126	Opcode für SHEL_RDEF
contrl+2	contrl[1]	0	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	2	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
addr_in	addr_in[0]	lpcmd	
addr_in+4	addr_in[1]	lpdir	
int_out	int_out[0]	Return-Wert (undefiniert)	

SHEL_WDEF (AES 127) – nur in PC-GEM ab Version 2.0

Analog zu "shel_rdef()" kann man mit dieser Funktion die "Default-Applikation" festlegen.

Deklaration in C:

```
WORD shel_wdef (CHAR *lpcmd, CHAR *lpdir)
{
     addr_in[0] = lpcmd;
     addr_in[1] = lpdir;
     return crys_if (127);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	127	Opcode für SHEL_WDEF
contrl+2	contrl[1]	0	# Einträge in int_in
contrl+4	contrl[2]	0	# Einträge in int_out
contrl+6	contrl[3]	2	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
addr_in	addr_in[0]	lpcmd	
addr_in+4	addr_in[1]	lpdir	
int_out	int_out[1]	Return-Wert (undefiniert)	

XGRF-Funktionen

XGRF_STEPCALC (AES 130) – nur in PC-GEM ab Version 2.0

Mit dieser Funktion kann man sich alle Parameter für einen "xgrf_2box()"-Aufruf berechnen lassen.

Aus Anfangsgröße, Endgröße und Endposition des Rechtecks werden Endposition, Anzahl der Schritte und Größe der Schritte bestimmt.

Deklaration in C:

```
WORD xgrf_stepcalc (WORD orgw, WORD orgh, WORD xc, WORD yc, WORD w,
                    WORD h, WORD *pcx, WORD *pcy, WORD *pcnt,
                    WORD *pxstep, WORD *pystep)
{
     int_in[0] = orgw;
     int_in[1] = orgh;
     int_in[2] = xc;
     int_in[3] = yc;
     int_in[4] = w;
     int_in[5] = y;
     crys_if (130);
     *pcx = int_out[1];
     *pcy = int_out[2];
     *pcnt = int_out[3];
     *pxstep = int_out[4];
     *pystep = int_out[5];
     return int_out[0];
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	130	Opcode für XGRF_STEPCALC
contrl+2	contrl[1]	6	# Einträge in int_in
contrl+4	contrl[2]	6	# Einträge in int_out
contrl+6	contrl[3]	0	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out

Adresse	Feldelement	Belegung
int_in	int_in[0]	orgw
int_in+2	int_in[1]	orgh
int_in+4	int_in[2]	xc
int_in+6	int_in[3]	yc
int_in+8	int_in[4]	w
int_in+10	int_in[5]	h
int_out	int_out[0]	Return-Wert (0: Fehler)
int_out+2	int_out[1]	pcx
int_out+4	int_out[2]	pcy
int_out+6	int_out[3]	pcnt
int_out+8	int_out[4]	pxstep
int_out+10	int_out[5]	pystep

Parameter:

orgw:	anfängliche Breite
orgh:	anfängliche Höhe
xc, yc:	Endposition
w, h:	Endgröße
pcx, pcy:	zentrierte Position nach Ablauf des Vorgangs
pcnt:	Anzahl der Einzelschritte
pxstep, pystep:	Schrittweite in X- und Y-Richtung pro Einzelschritt

XGRF_2BOX (AES 131) – nur in PC-GEM ab Version 2.0

Zeichnet eine Reihe von Rechtecken, die sich über den Bildschirm bewegen (dies ist ein Ersatz für die in PC-GEM 2.0 gestrichenen Funktionen der FORM- und GRAF-Bibliotheken).

Deklaration in C:

```
WORD xgrf_2box (WORD xc, WORD yc, WORD w, WORD h, WORD cornes,
                WORD cnt, WORD xstep, WORD ystep, WORD doubled)
{
     int_in[0] = cnt;
     int_in[1] = xstep;
     int_in[2] = ystep;
     int_in[3] = doubled;
     int_in[4] = corners;
     int_in[5] = xc;
     int_in[6] = yc;
     int_in[7] = w;
     int_in[8] = h;
     return crys_if (131);
}
```

GEM-Arrays:

Adresse	Feldelement	Belegung	
contrl	contrl[0]	131	Opcode für XGRF_2BOX
contrl+2	contrl[1]	9	# Einträge in int_in
contrl+4	contrl[2]	1	# Einträge in int_out
contrl+6	contrl[3]	0	# Einträge in addr_in
contrl+8	contrl[4]	0	# Einträge in addr_out
int_in	int_in[0]	cnt	
int_in+2	int_in[1]	xstep	
int_in+4	int_in[2]	ystep	
int_in+6	int_in[3]	doubled	
int_in+8	int_in[4]	corners	
int_in+10	int_in[5]	xc	
int_in+12	int_in[6]	yc	
int_in+14	int_in[7]	w	

Adresse	Feldelement	Belegung
int_in+16	int_in[8]	h
int_out	int_out[0]	Return-Wert

Parameter:

cnt:	Anzahl der Einzelschritte
xstep, ystep:	Schrittweite in X- und Y-Richtung
doubled:	0: normale Schrittweite
	1: Schrittweite verdoppeln
corners:	0: ganze Rechtecke zeichnen
	1: nur die Ecken zeichnen
xc, yc:	Anfangsposition
w, h:	Anfangsgröße

Kapitel 5: XCONTROL

Einleitung

XControl ist das Kontrollfeld, das dem Atari TT und dem Mega STE seit Ende 1990 beigelegt wird. Es ist modular aufgebaut und kann erweitert werden. Die Module sind Dateien, deren Namenserweiterung "CPX" (Control Panel eXtension) heißt. XControl ist also nichts anderes als ein Steuerungsprogramm für solche Module, die im wesentlichen Konfigurationsdialoge sein sollten. Für andere Zwecke sollte man XControl *nicht* mißbrauchen.

Die Kommunikation zwischen XControl und den einzelnen Modulen erfolgt über zwei Strukturen. In der einen (XCPB) macht XControl einige Flags und eine ganze Reihe von Hilfsfunktionen zugänglich. Jede CPX wiederum liefert eine CPXINFO-Struktur mit Funktionszeigern zurück.

XControl lädt beim Start alle verfügbaren CPX-Header. Während des Bootvorgangs werden alle CPX-Dateien einmal zum Initialisieren aufgerufen, sofern im Header ein entsprechendes Flag gesetzt ist. Bei jeder Aktivierung wird die Initialisierungsroutine des CPX-Moduls aufgerufen, wobei dann ein Zeiger auf eine CPXINFO-Struktur zurückzuliefern ist.

Für jedes einzelne Modul kann spezifiziert werden, ob es resident geladen werden soll (diese Eigenschaft kann man auch mittels der mitgelieferten Konfigurations-CPX ändern). Solche Module werden gleich beim Laden vollständig geladen und bleiben dann resident – wohl ein Zugeständnis für die Benutzer, bei denen die XControl-Module auf einer Diskette oder einer langsamen Festplatte liegen. Ebenso ist es möglich, CPX-Module zu schreiben, die nur bestimmte Werte setzen ("set-only"). Derartige CPX-Module geben bei der Initialisierung einfach einen Nullzeiger zurück. Sie werden nur beim Booten bzw. beim erneuten Laden der Module durch XControl aufgerufen.

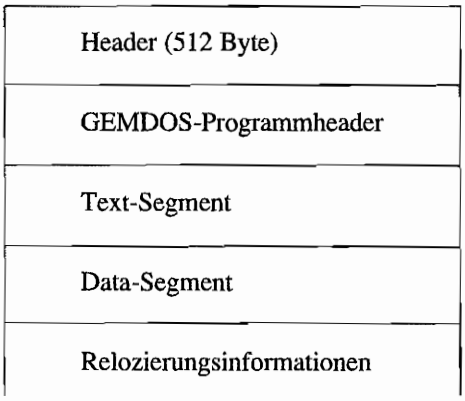
XControl verwendet "evnt_multi()" für die eigene Verwaltung und die der CPX-Module. Wird ein CPX-Modul vom Anwender ausgewählt, so lädt XControl dieses in den Speicher und ruft "cpx_init()" auf. Anschließend wird die Funktion "cpx_call()" aufgerufen, wobei im wesentlichen nun das CPX-Modul die Steuerung übernimmt. Es existieren zwei Arten von CPX-Modulen: Form-CPX und Event-CPX.

Form-CPX sind relativ einfach zu programmieren, bieten jedoch nur eine eingeschränkte Flexibilität. Event-CPX bieten mehr Flexibilität, da sie die AES-Events direkt verwerten. Alle mit XControl 1.0 ausgelieferten CPX-Module sind Form-CPX-Dateien. Dies zeigt, daß es in den meisten Fällen ausreicht, Form-CPX zu benutzen.

CPX-Format

Bevor wir zur Programmierung der CPX-Dateien übergehen, wollen wir das Dateiformat und die Terminologie näher betrachten.

Der Aufbau der CPX-Datei ist einem normalen Programm sehr ähnlich. Sie besteht aus einem 512 Byte großen Header und dem übrigen Dateiinhalt, bei dem es sich fast um eine normale GEMDOS-Programmdatei handelt.



Der Aufbau des Headers selbst:

```
typedef struct
{
     UWORD     magic;                    /* = 100 */
     struct
     {
          unsigned reserved   : 13;     /* reserviert */
          unsigned resident   : 1;      /* RAM-resident */
          unsigned bootinit   : 1;      /* Boot-Initialisierung */
          unsigned setonly    : 1;      /* Set-Only */
     } flags;
     LONG      cpx_id;              /* eindeutige CPX-ID */
     UWORD     cpx_version;         /* CPX-Versionsnummer */
     char      i_text[14];          /* Icontext */
     UWORD     sm_icon[48];         /* Bitmap (32 x 24 Pixel) */
     UWORD     i_color;             /* Iconfarbe */
```

```
    char        title_text[18];     /* Name */
    UWORD       t_color;            /* Textfarbe */
    char        buffer[64];         /* nicht flüchtiger Puffer */
    char        reserved[306];      /* reserviert */
} CPXHEAD;
```

Bemerkungen

- Die erste Funktion im Textsegment muß die Initialisierungsroutine für die CPX sein (siehe Definition von "cpx_init()").

- Zur Konstruktion solcher Header ist für eingetragene Entwickler bei Atari ein spezielles Tool erhältlich.

- Header und gelinkte Programmdatei können in den meisten UNIX-ähnlichen Shells mit dem Kommando "cat" zusammengefügt werden.

- Bei der CPX-Entwicklung ist es sehr praktisch, daß man XControl auch als Programm starten kann (einfach von "XCONTROL.ACC" in "XCONTROL.APP" umbenennen). So kommt man um ein permanentes Neu-Booten des Rechners herum.

- CPX-ID und -Versionsnummer sorgen dafür, daß jede CPX nur ein einziges Mal – und auch nur die neueste Version – erscheint.

Die offizielle Terminologie für die Dateinamen ist:

Namenserweiterung	Dateityp
*.CPX	Standard-CPX-Datei, fertig zum Gebrauch
*.CP	Standard-CPX-Datei ohne Header
*_R.CPX	residente CPX-Datei
*_S.CPX	set-only CPX-Datei
*.HDR	Header für CPX-Datei
*.CPZ	inaktive CPX-Datei (von XControl deaktiviert)

Programmierrichtlinien

In diesem Zusammenhang auch einige Grundregeln, die bei der Programmierung von CPX-Modulen beachtet werden sollen:

- Reservierter Speicher ist möglichst schnell wieder freizugeben.
- XControl-Funktionen sind immer auszunutzen, wenn dies möglich ist.
- Die Benutzerschnittstelle ist einfach und in Anlehnung an die anderen CPX-Module zu gestalten.
- Grafische Elemente sind Menükommandos vorzuziehen.
- "OK" und "Abbruch"/"Cancel" sind – sofern nötig – immer zu implementieren. Hierbei heißt es "OK" (*nicht* "Ok") und "Abbruch"/"Cancel" (*nicht* "ABBRUCH"/"CANCEL").
- Popup-Menüs sind als Text mit schattiertem Rechteck darzustellen.
- AC_CLOSE – das Verlassen des Hauptprogramms – wird als "Abbruch" gewertet.
- WM_CLOSE – das Schließen des XControl-Fensters – wird als "OK" gewertet.
- "Save"/"Sichern" ist als "OK" ohne Verlassen des Dialogs zu betrachten.
- Das Wurzelobjekt der CPX hat immer eine Größe von 256 x 176 Pixeln.
- Interrupt-Vektoren dürfen nicht verändert werden.
- "Xform_do()" darf nicht mit Funktionen für Event-CPX vermischt werden.
- Reservierter Speicher darf beim Verlassen der CPX nicht vergessen werden, da es sonst zu einer Speicherfragmentierung kommt.
- Bereits von anderen CPX verwandte IDs dürfen nicht mehr benutzt werden.
- Geöffnete Dateien müssen auf jeden Fall wieder geschlossen werden.
- Geöffnete VDI-Workstations sind auf jeden Fall wieder zu schließen (spätestens bei AC_CLOSE/WM_CLOSE!), wenn sie nicht mehr benötigt werden.

Nun zur Programmierung eines CPX-Moduls selbst. Da ein CPX-Modul – mit Ausnahme von 64 Byte – über keinen nicht-vergänglichen Speicher verfügt, ist nichts erlaubt, was Speicher in irgendeiner Form fest reserviert. Variableninhalte gehen mit dem Verlassen des CPX-Moduls in der Regel verloren! Eine Folge dieser Tatsache ist zum Beispiel, daß Resourcen statisch eingebunden werden müssen, Speicheranforderungen nur kurzzeitig erfolgen dürfen

und keine virtuellen VDI-Workstations (wg. der internen Speicheranforderung) dauerhaft angelegt werden.

Ein Beispiel-CPX

Nehmen wir als Beispiel ein ganz einfaches, aber lauffähiges CPX-Modul:

```
/* sample.c */
/* Entwickelt mit Turbo C. */
#include <aes.h>          /* AES                    */
#include <stddef.h>       /* Standard-Definitionen */
#include <cpxdata.h>      /* CPX-Datenstrukturen    */

/* Prototypen */
CPXINFO* cdecl cpx_init(XCPB *Xcpb);
WORD cdecl     cpx_call(GRECT *rect);

/* CPX-Strukturen */
XCPB    *xcpb;              /* XControl Parameter Block       */
CPXINFO cpxinfo =           /* CPX Information Structure      */
{
     cpx_call, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL
};

/* Resourcen/einfache Dialogbox */
#define BOK             2
#define NUM_OBS         3

OBJECT tree[] =
{
     {
               -1, 1, BOK,
               G_BOX, NONE, NORMAL,
               (long) 0x001101L,
               0x0000, 0x0000, 0x0020, 0x000b
     },
     {
               BOK, -1, -1,
               G_STRING, NONE, NORMAL,
                 (long) "Beispiel",
               0x000c, 0x0004, 0x0008, 0x0001
```

```
        },
        {
                0, -1, -1,
                G_BUTTON, LASTOB | EXIT | DEFAULT | SELECTABLE,
                NORMAL,
                (long) "OK",
                0x0002, 0x0009, 0x0007, 0x0001
        }
};

/* der Initialisierungsaufruf */
CPXINFO * cdecl cpx_init (XCPB *Xcpb)
{
     WORD i;

     xcpb = Xcpb;

     /* Bootvorgang? Dann Ende und anzeigen, daß die CPX
     erscheinen soll, also nicht 'set-only' ist... */
     if (xcpb->booting) return ((CPXINFO *) 1);

     /* Resourcen fixiert? */
     if (!xcpb->SkipRshFix)
          for (i = 0; i < NUM_OBS; i++)
               (*xcpb->rsh_obfix)(tree, i);
     return &cpxinfo;
}

/* der eigentliche Aufruf des CPX-Moduls */
WORD cdecl cpx_call (GRECT *rect)
{
     WORD msg[8];

     /* Dialogbox anpassen */
     tree[ROOT].ob_x = rect->g_x;
     tree[ROOT].ob_y = rect->g_y;

     /* Dialogbox ausgeben */
     objc_draw (tree, ROOT, MAX_DEPTH,
               rect->g_x, rect->g_y, rect->g_w, rect->g_h);
```

```
    /* Dialog durchführen */
     (*xcpb->Xform_do)(tree, 0, msg);

    /* fertig! */
     return 0;
}
```

Ferner wird ein kleines Assemblermodul benötigt:

```
; cpxstart.s
; ==========
; Startup-Datei für CPX-Module
;
          .globl    cpxstart
          .globl    save_vars
          .globl    cpx_init

     .text

cpxstart:
          jmp       cpx_init

     .data

; Speicherbereich für Default-Einstellungen
; (muß ggf. angepaßt werden)
save_vars:
          .dc.w   0
     .end
```

Mit der folgenden Projektdatei wird das CPX-Modul in Turbo C übersetzt:

```
SAMPLE.CP
=
CPXSTART.S
SAMPLE.C
TCSTDLIB.LIB
TCGEMLIB.LIB
```

Definitionen und Strukturen

Nun haben Sie einen Einblick in den Aufbau eines CPX-Moduls gewonnen. Es bleibt noch, die Strukturen CPXINFO und XCPB sowie alle dazugehörigen Definitionen zu erklären:

```
/* cpxdata.h */
typedef struct
{
     WORD x;
     WORD y;
     WORD buttons;
     WORD kstate;
} MRETS;

typedef struct
{
     WORD      handle;     /* aus graf_handle()-Aufruf von
                              XControl. Wichtig für v_opnvwk()! */
     WORD      booting;    /* ungl. 0: Initialisierg./Bootvorg. */
     WORD      reserved;   /* reserviert */
     WORD      SkipRshFix;/* ungleich: Resourcekoordinaten bereits
                             transformiert */
     void           *reserve1;     /* reserviert */
     void           *reserve2;     /* reserviert */
     void cdecl     (*rsh_fix)(WORD num_objs, WORD num_frstr,
                         WORD num_frimg, WORD num_tree,
                         OBJECT *rs_object, TEDINFO *rs_tedinfo,
                         char *rs_strings[], ICONBLK *rs_iconblk,
                         BITBLK *rs_bitblk, long *rs_frstr,
                         long *rs_frimg, long *rs_trindex,
                         struct foobar *rs_imdope);
     void cdecl     (*rsh_obfix)(OBJECT *tree, WORD curob);
     WORD cdecl     (*Popup)(char *items[], WORD num_items,
                    WORD default_item, WORD font_size,
                    GRECT *button, GRECT *world);
     void cdecl     (*Sl_size)(OBJECT *tree, WORD base,
                    WORD slider, WORD num_items, WORD visible,
                    WORD direction, WORD min_size);
     void cdecl     (*Sl_x)(OBJECT *tree, WORD base, WORD slider,
                    WORD value, WORD num_min, WORD num_max,
                    void (*foo)(void));
```

```
    void cdecl      (*Sl_y)(OBJECT *tree, WORD base, WORD slider,
                    WORD value, WORD num_min, WORD num_max,
                    void (*foo)(void));
    void cdecl      (*Sl_arrow)(OBJECT *tree, WORD base,
                    WORD slider, WORD obj, WORD inc, WORD min,
                    WORD max, WORD *numvar, WORD direction,
                    void (*foo)(void));
    void cdecl      (*Sl_dragx)(OBJECT *tree, WORD base,
                    WORD slider, WORD min, WORD max,
                    WORD *numvar, void (*foo)(void));
    void cdecl      (*Sl_dragy)(OBJECT *tree, WORD base,
                    WORD slider, WORD min, WORD max,
                    WORD *numvar, void (*foo)(void));
    WORD cdecl      (*Xform_do)(OBJECT *tree, WORD start_field,
                    WORD *puntmsg);
    GRECT * cdecl   (*GetFirstRect)(GRECT *prect);
    GRECT * cdecl   (*GetNextRect)(void);
    void cdecl      (*Set_Evnt_Mask)(WORD mask, MOBLK *m1,
                    MOBLK *m2, long time);
    WORD cdecl      (*XGen_Alert)(WORD id);
    WORD cdecl      (*CPX_Save)(void *ptr, long num);
    void * cdecl    (*Get_Buffer)(void);
    WORD cdecl      (*getcookie)(long cookie, long *p_value);
    WORD            Country_Code;
                        /* Länderkennung, analog zu der im
                        OSHEADER - allerdings in fester
                        Abhängigkeit von der XControl-Version */
    void cdecl      (*MFsave)(WORD saveit, MFORM *mf);
} XCPB;

typedef struct
{
    WORD cdecl      (*cpx_call)(GRECT *work);
    void cdecl      (*cpx_draw)(GRECT *clip);
    void cdecl      (*cpx_wmove)(GRECT *work);
    void cdecl      (*cpx_timer)(WORD *event);
    void cdecl      (*cpx_key)(WORD kstate, WORD key, WORD *event);
    void cdecl      (*cpx_button)(MRETS *mrets, WORD *event);
    void cdecl      (*cpx_m1)(MRETS *mrets, WORD *event);
    void cdecl      (*cpx_m2)(MRETS *mrets, WORD *event);
```

```
    WORD cdecl      (*cpx_hook)(WORD event, WORD *msg, MRETS
                    *mrets,WORD *key, WORD *nclicks);
    void cdecl      (*cpx_close)(WORD flag);
} CPXINFO;

#define VERTICAL        0
#define HORIZONTAL      1

#define SAVE_DEFAULTS   0
#define MEM_ERR         1
#define FILE_ERR        2
#define FILE_NOT_FOUND  3

#define MFSAVE          1
#define MFRESTORE       0

/* zusätzliche Definition für Xform_do() */
#define CT_KEY 53
```

Von der CPX bereitgestellte Funktionen

cpx_init (Intitialisierung)

Diese Funktion muß am Beginn des Textsegments der CPX-Datei stehen (nun ja, zumindest ein Sprungbefehl zu ihrem Anfang) und wird während der XControl-Initialisierung sowie beim Aktivieren der CPX aufgerufen.

Als Parameter erhält sie einen Zeiger auf die oben definierte XCPB-Struktur. Als Resultat liefert man einen Zeiger auf die eigene CPXINFO-Struktur oder einen Nullzeiger zurück.

Deklaration in C:

```
CPXINFO * cdecl cpx_init (XCPB *xcpb);
```

Eingabe-Parameter:

xcpb: Zeiger auf die XCPB-Struktur von XControl

Ausgabe-Parameter:

NULL: Es handelt sich um eine "set-only" CPX.

sonst: Zeiger auf die CPXINFO-Struktur der CPX (oder *während* des Bootens: irgendein Wert ungleich Null, um anzuzeigen, daß XControl die CPX in die Liste der aktivierbaren CPXe aufnehmen soll).

Funktionen aus CPXINFO

cpx_call (Aktivierungsroutine)

Quasi das Hauptprogramm des CPX-Moduls. "cpx_call()" wird nach "cpx_init()" aufgerufen, wenn der Anwender das entsprechende Modul ausgewählt hat. Als Übergabeparameter erhält man die Koordinaten der Arbeitsfläche im XControl-Fenster. Somit läßt sich zum Beispiel die Dialogbox passend plazieren.

Deklaration in C:

```
WORD cdecl (*cpx_call)(GRECT *work);
```

Eingabe-Parameter:

work: Rechteck mit den Koordinaten des XControl-Fensters

Ausgabe-Parameter:

0: Ende der Bearbeitung
sonst: CPX soll weiterbearbeitet werden

Event-Handling-Routinen aus CPXINFO (nur für Event-CPX)

cpx_draw (Redraw-Ereignis)

Diese Routine wird aufgerufen, wenn ein CPX-Modul aktiv ist und XControl ein Redraw benötigt. Die nötige Rechteckliste ist mit "GetFirstRect()" und "GetNextRect()" abzurufen.

Deklaration in C:

```
void cdecl (*cpx_draw)(GRECT *clip);
```

Eingabe-Parameter:

clip: neu zu zeichnender Bereich, der auch als Übergabeparameter für "GetFirstRect()" benötigt wird.

cpx_wmove (Fensterverschiebung)

"cpx_wmove()" wird aufgerufen, wenn der Anwender das XControl-Fenster bewegt.

Deklaration in C:

```
void cdecl (*cpx_wmove)(GRECT *work);
```

Eingabe-Parameter:

work: neue Fenster-Koordinaten

cpx_timer (Timer-Event)

Die Funktion wird aufgerufen, wenn ein Timer-Event aufgetreten ist. Timer-Events werden von Form-CPX nicht unterstützt.

Deklaration in C:

```
void cdecl (*cpx_timer)(WORD *event);
```

Ausgabe-Parameter:

event: auf 1 setzen, wenn die CPX verlassen werden soll; ansonsten ignorieren

cpx_key (Keyboard-Event)

"cpx_key()" wird aufgerufen, wenn ein Keyboard-Event aufgetreten ist.

Deklaration in C:

```
void cdecl (*cpx_key)(WORD kstate, WORD key, WORD *event);
```

Eingabe-Parameter:

kstate: Status der Umschalttasten (Alternate, Control, Shift etc.)
key: enthält im Highbyte den Scancode und im Lowbyte den ASCII-Code (sofern vorhanden) der gedrückten Taste

Ausgabe-Parameter:

event: auf 1 setzen, wenn die CPX verlassen werden soll; ansonsten ignorieren

cpx_button (Maustasten-Event)

Die Funktion wird bei einem aufgetretenen Maustasten-Event aufgerufen.

Deklaration in C:

```
void cdecl (*cpx_button)(MRETS *mrets, WORD nclicks, WORD *event);
```

Eingabe-Parameter:

mrets: Parameter der Maus, die zu diesem Event gehören
nclicks: Anzahl der Mausklicks

Ausgabe-Parameter:

event: auf 1 setzen, wenn die CPX verlassen werden soll; ansonsten ignorieren

cpx_m1, cpx_m2 (Mausrechteck-Event)

"cpx_m1()" bzw. "cpx_m2()" werden dann aufgerufen, wenn der Mauszeiger bestimmte Rechtecke betritt oder verläßt.

Deklaration in C:

```
void cdecl (*cpx_m1)(MRETS *mrets, WORD *event);
void cdecl (*cpx_m2)(MRETS *mrets, WORD *event);
```

Eingabe-Parameter:

mrets: Parameter der Maus, die zu diesem Event gehören

Ausgabe-Parameter:

event: auf 1 setzen, wenn die CPX verlassen werden soll; ansonsten ignorieren

cpx_hook (Preemption Hook)

Die Funktion wird sofort nach "evnt_multi()" aufgerufen, also noch bevor XControl das Event verarbeitet.

Deklaration in C:

```
WORD cdecl (*cpx_hook)(WORD event, WORD *msg, MRETS *mrets,
           WORD *key, WORD *nclicks);
```

Eingabe-Parameter:

event:	aufgetretene Events
msg:	Ereignispuffer
mrets:	Mausparameter
key:	Tastendruck
nclicks:	Anzahl der Mausklicks

Ausgabe-Parameter:

0:	Eventverarbeitung fortsetzen
sonst:	Eventverarbeitung abbrechen

cpx_close (Close-Event)

"cpx_close()" wird bei jeder AC_CLOSE- und jeder WM_CLOSE-Message aufgerufen. Die CPX muß dann sofort allen reservierten Speicher ("Malloc()", "v_opnvwk()", etc.) freigeben. Die Funktion muß bei jeder Event-CPX implementiert sein. AC_CLOSE ist als Klick auf "Abbruch" oder "Cancel" zu werten, WM_CLOSE als "OK" (Bemerkung: CPX-Module sollten so selten wie möglich Speicher reservieren).

Deklaration in C:

```
void cdecl (*cpx_close)(WORD flag);
```

Eingabe-Parameter:

flag:	0:	AC_CLOSE-Mitteilung
	sonst:	WM_CLOSE-Mitteilung

Von XControl bereitgestellte Funktionen

rsh_fix (Objektbaum-Umwandlung)

Wandelt einen Objektbaum auf Basis von 8x16 Pixel großen Zeichen – also pixelgenau und nicht auf Basis des aktuellen Systemzeichensatzes – um. Die CPX hat somit unter allen Auflösungen die gleiche Pixelgröße. Bei der Arbeit mit einem RCS sollte man daher ebenfalls einen Grafikmodus mit 8x16 Pixel großen Zeichen wählen.

Die Koordinatenumwandlung darf natürlich nur ein einziges Mal stattfinden – XControl stellt dazu in der XCPB-Struktur das Flag "SkipRshFix" zur Verfügung.

Deklaration in C:

```
void cdecl (*rsh_fix)(WORD num_objs, WORD num_frstr, WORDnum_frimg,
           WORD num_tree, OBJECT *rs_object, TEDINFO *rs_tedinfo,
           char *rs_string[], ICONBLK *rs_iconblk,
           BITBLK *rs_bitblk, long *rs_frstr, long *rs_frimg,
           long *rs_trindex, struct foobar *rs_imdope);
```

Eingabe-Parameter:

analog zu den C-Resourcen, die vom Atari-RCS angelegt werden

rsh_obfix (Objekt-Umwandlung)

Entspricht der Funktion "rsrc_obfix()", nur daß die CPX-spezifische Umrechnungsregel von Zeichen- in Pixelkoordinaten benutzt wird.

Deklaration in C:

```
void cdecl (*rsh_obfix)(OBJECT *tree, WORD curob);
```

Eingabe-Parameter:

tree: Zeiger Objektbaum
curob: zu konvertierendes Objekt

Popup (Popup-Menü)

Diese Funktion ermöglicht die komplette Verwaltung eines Popup-Menüs. Bei zu vielen Einträgen (ab fünf) wird automatisch gescrollt. Die Abarbeitung des Popup blockiert alle anderen Aktionen.

Deklaration in C:

```
WORD cdecl (*Popup)(char *items[], WORD num_items, WORD default_item,
            WORD font_size, GRECT *button, GRECT *world);
```

Eingabe-Parameter:

items:	Array mit Zeichenketten für die einzelnen Einträge. Jeder einzelne Eintrag muß die gleiche Länge haben sowie vorne mindestens zwei und am Ende mindestens ein Leerzeichen aufweisen.
num_items:	Anzahl der Einträge
default_item:	Default-Eintrag (Zählung beginnt bei 0) oder –1
font_size:	Zeichengröße; 8x16- oder 8x8-Font: Als Parameter sind die gleichen Werte wie in der TEDINFO-Struktur – also 3 (groß) bzw. 5 (klein) – zu verwenden. Laut Atari wird zur Zeit nur der große Zeichensatz verwendet.
button:	Rechteck des Buttons, zu dem das Popup-Menüs gehört
world:	Rechteck des Hintergrund-Objektbaums (in der Regel der Objektbaum der CPX)

Ausgabe-Parameter:

gewählter Eintrag (ab 0) oder –1

Sl_size (Slidergröße)

"Sl_size()" stellt die Größe des Sliders ein, damit die Relation der dargestellten Datenmenge zur vorhandenen gewahrt bleibt.

Deklaration in C:

```
void cdecl (*Sl_size)(OBJECT *tree, WORD base, WORD slider,
                      WORD num_items, WORD visible, WORD direction,
                      WORD min_size);
```

Eingabe-Parameter:

tree:	Zeiger auf Objektbaum
base:	Basisobjekt (Hintergrundobjekt für Slider)
slider:	Slider; Objekt, welches innerhalb des Basisobjekts bewegt wird (Child des Basisobjekts)
num_items:	Anzahl der tatsächlich vorhandenen Elemente
visible:	Anzahl der sichtbaren Elemente
direction:	Richtung (VERTICAL (0) oder HORIZONTAL (1))
min_size:	Minimalgröße des Sliders in Pixeln

Sl_x, Sl_y (Positionierung eines Sliders)

Die beiden Funktionen positionieren den Slider innerhalb eines Basisobjekts in horizontaler bzw. vertikaler Richtung.

Deklaration in C:

```
void cdecl (*Sl_x)(OBJECT *tree, WORD base, WORD slider,
                   WORD value,WORD num_min, WORD max,
                   void (*foo)(void));
void cdecl (*Sl_y)(OBJECT *tree, WORD base, WORD slider,
                   WORD value, WORD num_min, WORD max,
                   void (*foo)(void));
```

Eingabe-Parameter:

tree: Zeiger auf Objektbaum
base: Basisobjekt
slider: Slider (Child des Basisobjekts)
value: neuer Wert, den der Slider repräsentieren soll
min: Minimalwert, den value annehmen darf
max: Maximalwert, den value annehmen darf
foo: Adresse einer Funktion (oder NULL), die gleichzeitig mit der Slider-Neupositionierung aufgerufen wird; so lassen sich Sliderbewegungen ausnutzen, um auch die angezeigten Werte zu erneuern.

Sl_arrow (Sliderarrow)

Sobald einer der zu dem Slider gehörigen Pfeile angeklickt wird, ist diese Funktion aufzurufen. Sie wird auch dazu benutzt, um einen Klick auf das Basisobjekt des Sliders auszuwerten.

Deklaration in C:

```
void cdecl (*Sl_arrow)(OBJECT *tree, WORD base, WORD slider,
                       WORD obj,WORD inc, WORD min, WORD max,
                       WORD *value, WORD direction,
                       void (*foo)(void));
```

Eingabe-Parameter:

tree:	Zeiger auf Objektbaum
base:	Basisobjekt
slider:	Slider (Child des Basisobjekts)
obj:	Pfeil, der angeklickt wurde
inc:	Inkrementierung (Anzahl der Einheiten, die addiert oder subtrahiert werden sollen); sollte für angeklicktes Basisobjekt (seitenweises Blättern) und Arrowbutton (zeilenweises Blättern) unterschiedlich sein
min:	Minimalwert, der angenommen werden kann
max:	Maximalwert, der angenommen werden kann
value:	Adressen für aktuellen Wert
direction:	VERTICAL (0) oder HORIZONTAL (1)
foo:	Adresse einer Funktion analog zu "Sl_x()" und "Sl_y()"

Sl_dragx, Sl_dragy (Sliderdrag-Bewegung)

Beim Draggen des Sliders wird selbiger angeklickt und bei festgehaltener Maustaste innerhalb eines Basisobjekts bewegt. Die beiden Funktionen verwalten diese Bewegung.

Deklaration in C:

```
void cdecl (*Sl_dragx)(OBJECT *tree, WORD base, WORD slider,
                       WORD min,WORD max, WORD *value,
                       void (*foo)(void));
void cdecl (*Sl_dragy)(OBJECT *tree, WORD base, WORD slider,
                       WORD min,WORD max, WORD *value,
                       void (*foo)(void));
```

Eingabe-Parameter:

tree:	Zeiger auf Objektbaum
base:	Basisobjekt
slider:	Slider (Child des Basisobjekts)
min:	Minimalwert, der angenommen werden kann
max:	Maximalwert, der angenommen werden kann
value:	Adresse für aktuellen Wert
foo:	Adresse einer Funktion analog zu "Sl_x()" und "Sl_y()"

Xform_do (Formular-Verwaltung)

Die Formular-Verwaltung erfolgt analog zu der bekannten AES-Funktion "form_do()", ist jedoch etwas komplexer. Sie übernimmt in geringem Umfang auch das Bearbeiten von AES-Mitteilungen.

Deklaration in C:

```
WORD cdecl (*Xform_do)(OBJECT *tree, WORD startob, WORD *puntmsg);
```

Eingabe-Parameter:

tree:	Objektbaum (wie bei "form_do()")
startob:	Start-Objekt (wie bei "form_do()")
puntmsg:	Mitteilungs-Puffer (ähnlich "evnt_mesag()"/"evnt_multi()")

Ausgabe-Parameter:

−1: puntmsg enthält eine Nachricht, die auszuwerten ist:

WM_REDRAW (20): Die CPX muß solche Objekte selbst neuzeichnen, die nicht zum Objektbaum gehören. Die Rechteckliste ist über "GetFirstRect()" und "GetNextRect()" abzufragen.

AC_CLOSE (41), WM_CLOSE (22): Die CPX wurde beendet. Reservierter Speicher ist sofort freizugeben.
AC_CLOSE ist als "Abbruch" und WM_CLOSE als "OK" zu werten!

CT_KEY (53): Eine spezielle Nachricht, die die Auswertung von Tastendrücken erlaubt, sofern diese keine Auswirkungen auf Editfelder haben können – also Funktionstasten, HELP und UNDO. puntmsg[3] enthält Scancode (High-byte) und ASCII-Code (Lowbyte) der gedrückten Taste.

sonst: Nummer des angeklickten Objekts (im oberen Bit die Kennzeichnung für einen Doppelklick)

GetFirstRect, GetNextRect (Rechteckliste)

Die Rechteckliste, die zum Neuzeichnen von Fensterbereichen nach einer WM_REDRAW-Message nötig ist, wird mit den beiden Funktion "GetFirstRect()" und "GetNextRect()" abgefragt. Den Objektbaum verwaltet jedoch XControl selbst!

Deklaration in C:

```
GRECT * cdecl (*GetFirstRect)(GRECT *prect);
GRECT * cdecl (*GetNextRect)(void);
```

Eingabe-Parameter:

prect: Zu aktualisierender Bereich

Ausgabe-Parameter:

NULL: keine weiteren Ausschnitte vorhanden
sonst: wiederherzustellender Ausschnitt

Set_Evnt_Mask (Eventmaske setzen)

Bei Event-CPX bestimmt "Set_Evnt_Mask()", auf welche Events die CPX reagieren soll.

Deklaration in C:

```
void cdecl (*Set_Evnt_Mask)(WORD mask, MOBLK *m1, MOBLK *m2,
                            long time);
```

Eingabe-Parameter:

mask: erlaubte Events (analog zu "evnt_multi()")
m1, m2: jeweils Mausrechteck und -richtung
time: Zeit in Millisekunden für Timerevent

XGen_Alert (Alarmbox)

Eine einfache Form einer Alarmbox bietet "XGen_Alert()". Die Funktion bietet Alarmboxen jedoch nur für wenige Fehlermeldungen an, weitere Alarmboxen müssen selbst definiert werden. "form_alert()" bietet sich hier jedoch nicht an, da es die Alarmbox bezüglich der vollen Bildschirmfläche und nicht bezüglich des XControl-Fensters zentriert.

Deklaration in C:

```
WORD cdecl (*XGen_Alert)(WORD id);
```

Eingabe-Parameter:

id: 0: SAVE_DEFAULTS ("Voreinstellungen sichern?")
1: MEM_ERR ("Fehler bei Speicheranforderung!")
2: FILE_ERR ("Fehler beim Schreiben/Lesen von Dateien!")
3: FILE_NOT_FOUND ("Datei nicht gefunden!")

Ausgabe-Parameter:

0: "Abbruch" bzw. "Cancel" wurde angeklickt.
sonst: "OK" wurde angeklickt (wenn eine Alarmbox nur einen Knopf hat, ist es der OK-Button!).

CPX_Save (Defaults sichern)

Defaulteinstellungen können mit "CPX_Save()" gesichert werden. Dies wird beispielsweise dann benötigt, wenn bestimmte Einstellungen beim Bootvorgang voreingestellt werden sollen. Um die Einstellungen zu speichern, sucht XControl eine CPX mit passendem Namen. Wird keine Datei mit passendem Namen gefunden, dann sucht XControl nach übereinstimmender ID und Versionsnummer. Ist die CPX anhand der ID und der Versionsnummer gefunden, so wird die CPX aktiviert, ansonsten meldet XControl – wie auch im Falle einer schreibgeschützten Diskette – einen Fehler.

XControl speichert die Einstellungen im DATA-Segment der CPX. Entwickler müssen selbst für ausreichend freien Speicherplatz im DATA-Segment sorgen. Dies geschieht über das Datenfeld SAVE_VARS in CPXSTART.S.

Deklaration in C:

```
WORD cdecl (*CPX_Save)(void *ptr, long num);
```

Eingabe-Parameter:

ptr:	Adresse der zu speichernden Daten
num:	Anzahl Byte der in das DATA-Segment zu speichernden Daten

Ausgabe-Parameter:

0:	Es ist ein Fehler aufgetreten.
sonst:	Alles in Ordnung.

Get_Buffer (Zwischenspeicher ermitteln)

Einen Zeiger auf einen 64 Byte großen, residenten Speicherbereich liefert die Funktion "Get_Buffer". Hier kann eine CPX die Inhalte von Write-Only-Registern sichern, sofern TOS keine Funktion zur Abfrage bietet (Beispiel: Fensterfarben). In diesem Zusammenhang sei noch einmal darauf hingewiesen, daß jeder andere Speicher einer CPX flüchtig ist!

Deklaration in C:

```
void cdecl (*Get_Buffer)(void);
```

Ausgabe-Parameter:

Zeiger auf residenten Speicherbereich

getcookie (Cookievariablen abfragen)

"getcookie" prüft, ob ein bestimmter Cookie vorhanden ist und liefert gegebenenfalls deren Wert.

Deklaration in C:

```
WORD cdecl (*getcookie)(long cookie, long *p_value);
```

Eingabe-Parameter:

cookie: Cookievariable
p_value: Adressen einer Variablen, die den Wert beinhalten soll, oder NULL, falls der Wert uninteressant ist

Ausgabe-Parameter:

0: Cookie nicht gefunden
sonst: Cookie gefunden

MFsave (Mausform sichern oder wiederherstellen)

Die Mausform kann gesichert (und wiederhergestellt) werden, womit der Mauszeiger temporär verändert werden kann. Dies ist beispielsweise dann nötig, wenn während einer Sliderbewegung die "flache Hand" eingeschaltet wird. Es kann nicht davon ausgegangen werden, daß der Mauszeiger vorher eine bestimmte Form hatte!

Deklaration in C:

```
void cdecl (*MFsave)(WORD saveit, MFORM *mf);
```

Eingabe-Parameter:

saveit:	1: MFSAVE (Mausform sichern)
	0: MFRESTORE (Mausform wiederherstellen)
mf:	Speicherbereich zur Sicherung der Mausform

Kapitel 6: Richtlinien zur Benutzerführung und Programmierung

Grundsätzliches zu grafischen Benutzeroberflächen

Benutzeroberfläche

(user surface, user interface) Präsentation einer Software gegenüber dem Anwender durch Anzeigen von Arbeits- und Bedienungselementen auf dem Bildschirm und Unterstützung von Eingabehilfen wie Tastatur, Grafiktablett oder Maus.

Benutzerfreundlichkeit

Eigenschaft eines Computersystems und der dazugehörigen Software. Die Benutzerfreundlichkeit der Hardware (z. B. Ein-/Ausgabegeräte wie Tastatur, Bildschirm, Drucker) hängt hauptsächlich von der ergonomischen Gestaltung und einer guten Dokumentation ab. Die Benutzerfreundlichkeit der Software wird gemessen an der Art und Qualität der Bedienerführung.

Bedienerführung

Bezeichnung für die Einrichtung einer Software zur Unterstützung des Anwenders, z. B. durch Auswahlmenüs, Hilfsprogramme, Sicherheitsabfragen oder auf dem Bildschirm angezeigte Bedienungshinweise. Die Bedienerführung soll auch Fehlbedienungen und damit zusammenhängende Datenverluste verhindern.

Am Anfang der Computer-Geschichte standen kommandoorientierte Benutzeroberflächen, die mit Benutzer*freundlichkeit* eigentlich nichts zu tun hatten. Der Bedienung des Computers ging ein kleines Studium der Befehle und Kommandos voraus. Beherrschte man diese dann soweit, daß Programme gestartet werden konnten, so schloß sich das nächste Studium an. Jedes Programm hatte seine eigene – teils recht konfuse – Oberfläche, jeder Programmierer programmierte einfach drauflos. Die Bedienerführung der meisten Programme war keineswegs einheitlich und leicht erlernbar.

Der Sinn einer einheitlichen – wenn auch nicht immer optimalen – Benutzeroberfläche wurde und wird von einigen noch immer nicht recht eingesehen. Wer sich allerdings schon intensiver

mit einer einheitlichen, grafischen Benutzeroberfläche wie GEM auseinandergesetzt hat, konnte schnell feststellen, daß derartige Oberflächen ihre Berechtigung haben und die Benutzerfreundlichkeit wesentlich erhöhen. Auch das WYSIWYG-Prinzip – What You See Is What You Get – entspringt den Ideen einer grafischen Oberfläche.

Der Gestaltung und Realisierung einer durchdachten grafischen Oberfläche, die sich für alle Anwendungen einsetzen läßt, geht eine Untersuchung voraus, wie sich Menschen am Computer verhalten.

Einer dieser gewichtigen Punkte ist das Muskelgedächtnis. Wer mit einer Schreibmaschine umgehen kann, wird alsbald feststellen, daß die Fertigkeit, Texte zu tippen, immer mehr zunimmt. Die Finger "erinnern" sich an die Lage der Tasten. Je größer dieses Erinnerungsvermögen ist, um so schneller kommt der Text auf das Papier.

Das Gesetz, welches dahinter steht, ist das sogenannte Gesetz von Fitt:

$t = i \cdot \log (d / s + 0{,}5)$

mit t: Zeit, die man benötigt, um sich zu einem Objekt zu bewegen
i: Proportionalitätskonstante (100 Millisekunden pro Bit)
d: Abstand des Objekts vom gegenwärtigen Standort
s: Größe des Objekts
log: Logarithmus zur Basis 2

Die Proportionalitätskonstante resultiert aus einer Art inneren Uhr im Menschen. In bezug auf grafische Benutzeroberflächen bedeutet dies: Wege zwischen zwei Objekten möglichst kurz halten, gleichartige Objekte immer an gleichen Plätzen unterbringen, denn wenn man schon weiß, wo ein anderes Objekt zu finden ist, dann ist der Weg dorthin auch kürzer (Vermeidung von Umwegen!).

Auch die Größe der Objekte verringert die Zugriffszeit auf dasselbe Objekt. Daraus resultiert, daß z. B. Buttons in Dialogboxen recht groß ausfallen, wobei aber auf jeden Fall zu beachten ist, daß man dabei nicht das Maß der Dinge verliert. Zu groß ist nämlich nur unschön. Im Gegenteil: Ein großflächiges Rechteck wird unter Umständen gar nicht mehr als Button erkannt!

Das Muskelgedächtnis hilft, ein Objekt schnell zu finden, die Zugriffszeit kann dadurch kürzer werden. Die Zeit, die das Muskelgedächtnis selbst benötigt, läßt sich als das (Potenz-)Gesetz der Praktikabilität ausdrücken:

$t(n) = t(1) \cdot n^{-a}$

mit t(n): Zeit für den n-ten Versuch
a: approximative Konstante (ungefähr 0,4)

Sind gleichartige Objekte immer an der gleichen Stelle, so benötigt man für den ersten Zugriff die Zeit t(1). Mit jedem neuen Versuch, die gleiche Funktion auszuüben, sinkt die Zugriffszeit. Bei nicht-standardisierten Oberflächen hingegen verbleibt man ewig bei der Zeit t(1). Und das ist kein Gewinn!

Das Muskelgedächtnis hilft, die rein manuelle Zugriffszeit zu vermindern. Die Reaktionszeit bedingt jedoch das Auge! Experimente haben gezeigt, daß die normale Wahrnehmungszeit oder besser Unterscheidungszeit bei mindestens 50 bis 200 Millisekunden liegt. Bilder in zeitlichen Abständen von weniger als 50 Millisekunden werden nicht mehr bewußt, bei mehr als 200 Millisekunden jedoch garantiert als Einzelbilder wahrgenommen. Denken Sie hierbei an die niedrige Darstellungsfrequenz eines Fernsehers: Abstände zwischen zwei Bildern liegen etwa bei 40 Millisekunden, also dicht an der Grenze!

Die Wahrnehmungsfähigkeit des Auges bedingt, daß auf dem Bildschirm zwei Ereignisse, die trennbar sein müssen, erst in einem bestimmten zeitlichen Abstand erfolgen können. Man muß wahrnehmen, daß sich der Mauszeiger auf einem Objekt befindet! Diese zeitliche Verzögerung bei den Wahrnehmungen kann durch bestimmte Ereignisse verkürzt werden. Das Aufflackern eines invertierten Bildes wird schneller bemerkt. Eine Dialogbox mit hoher Farbintensität (schwarz auf weiß) wirkt für den Menschen vor dem Computer wie ein visueller Schock. Daraus folgt eine erheblich schnellere Wahrnehmung. Denken Sie an die Icons bei Alarmboxen oder die Invertierung der Menüeinträge bei Berührung mit dem Mauszeiger oder an die schwarz auf weiß darstellten Icons, wenn sie angewählt wurden!

Der Mensch verfügt über ein Kurzzeitgedächtnis, das jedoch nur eine beschränkte Informationsmenge verarbeiten kann. Die Anzahl der Informationen beschränkt sich meist auf die "magische" Zahl Sieben. Die Summe der wahrnehmbaren Informationen liegt etwa um zwei über bzw. unter diesem magischen Wert (nach Miller). Aus dieser Zahl folgt, wie ein Menü gestaltet sein muß: maximal sieben Drop-Down-Menüs (zuzüglich dem Informationsmenü mit den Accessories). In jedem Drop-Down-Menü sollten sich wenige Gruppen von jeweils maximal sieben Einträgen befinden. Zusätzlich zu dieser durch die Anzahl der auftretenden Wahlmöglichkeiten bedingten Gestaltungsform ist ein Menü, bei dem die einzelnen Untermenüs bei der Berührung der Menüzeile mit dem Mauszeiger herunterfallen (Drop-Down), einfacher zu bedienen und zu erkennen als ein Menü, bei dem erst mit der Maus auf einen Eintrag geklickt und somit das Untermenü (bei gedrückter Maustaste) heruntergezogen werden muß (Pull-Down).

Auch Farben sollten im Userinterface – wenn überhaupt – nur sehr sparsam eingesetzt werden. Einmal daher, weil die Programme sowieso ohne Verlust der Bedienbarkeit auch auf monochromen Bildschirmen funktionieren müssen. Andererseits sind Farben nur bedingt zur Übermittlung zusätzlicher Informationen geeignet, da ein bedeutender Teil der Bevölkerung unter Farbenblindheit bzw. einer Farbsehschwäche leidet und die Erkennung und Bewertung

von Farben hochgradig subjektiv ist. Farben sollten also ausschließlich dann eingesetzt werden, wenn sie zur Klarheit der Darstellung beitragen!

Der Faktor "denken" spielt somit auch eine große Rolle. Ein Mensch vor dem Computer sollte nicht dauernd nachdenken müssen! Bedienungselemente müssen intuitiv bedienbar sein! Die Überlegung, wie eine Funktion nun zu bedienen war oder wo eine bestimmte Einstellung vorgenommen werden kann, verbraucht kostbare Zeit. Daß diese Zeit nicht immer entfallen kann, dürfte klar sein. Eine Anwendung sollte aber den Menschen darin unterstützen, möglichst wenig Bedienungselemente erlernen zu müssen. Ein Ziel muß sich in maximal sieben Schritten (hier ist die magische Zahl wieder!) erreichen lassen.

Nehmen wir ein Beispiel: Sie schreiben einen Brief und entdecken einen falschen Buchstaben. Was ist zu tun? Buchstaben finden, Hand zur Maus bewegen, Mauszeiger auf Buchstaben bewegen, einmal klicken, Hand zur Tastatur bewegen, Buchstaben löschen, neuen Buchstaben eingeben. Ein Benutzer muß aber nicht nur den Bewegungsablauf kennen, der hilft, einen Buchstaben zu löschen. Vielmehr muß dabei in Erinnerung bleiben, an welcher Stelle am Dokument gerade geschrieben wurde, was gerade geschrieben werden sollte, wo der fehlerhafte Buchstabe erblickt wurde und wie der neue Buchstabe aussieht. Je größer die Anzahl der zu merkenden Dinge wird, desto eher besteht die Gefahr, durcheinanderzugeraten. Außerdem dauert ein Vorgang um so länger, je mehr Dinge in Erinnerung behalten werden müssen und je mehr Auswahlmöglichkeiten bestehen!

Denken Sie beim Entwurf einer GEM-Anwendung an die unterschiedlichsten Benutzer. Am besten ist es, während der Entwicklungsphase unterschiedlich erfahrene Tester heranzuziehen. Es hat sich schon immer als vorteilhaft erwiesen, absoluten Laien aber auch Experten ein Programm in die Hand zu geben! Ein Programm muß sich von einem Sekretär wie von einer Professorin problemlos bedienen lassen!

Die Gestaltung der Benutzeroberfläche

Einleitung

GEM stellt eine Schnittstelle zwischen einer Applikation und dem Anwender. Eine Schnittstelle hat einen einheitlichen Aufbau, damit es zwischen den beiden Partnern nicht zu Verständnisschwierigkeiten kommt. Stellen Sie sich mal vor, die RS232-Schnittstelle verhielte sich auf jedem Rechner anders! Ebenso muß diese "Software"-Schnittstelle auch auf unterschiedlicher Hardware gleich sein. Eine saubere GEM-Anwendung sieht auf DOS-Rechnern genauso aus wie auf einem Atari TT. Neben der Einhaltung "äußerer" Konventionen sind auch "innere" zu beachten. Damit ist gemeint, daß eine Anwendung nicht nur mit einem bestimmten Monitor oder einer bestimmten Speicherkonfiguration funktionstüchtig ist. Programme, die hardwarenah

geschrieben wurden, sollten zumindest darauf achten, daß sie mit anderen Konfigurationen nicht abstürzen und daß die "äußere" Schnittstelle eingehalten wird.

Menüs

Die Gestaltung des Menüs ist eine der wichtigsten Komponenten eines GEM-Programms, denn die Menüleiste sieht man zuerst. Man kann sagen, daß die Menüleiste die Visitenkarte des Programms ist. Auch wenn ein Menü (und das ganze Programm) den Konventionen entspricht, so muß das nicht heißen, daß es damit langweilig werden muß. Auch so gibt es genug Freiräume für pfiffige und durchdachte Funktionen.

Das "Muskelgedächtnis" spielt eine große Rolle bei der Gestaltung des Menüs. Wer ein GEM-Programm verlassen will, der sucht ein Dateimenü und darin den untersten Eintrag. Eine Information zum Programm vermutet man in dem Menü, in dem auch die Accessories angewählt werden.

Ein Menü besteht aus der Menüleiste mit diversen Titeleinträgen für die jeweiligen Drop-Down-Menüs ("Drop-Down" wie "herunterfallen" im Gegensatz zu "Pull-Down" wie "herunterziehen"). Jedes einzelne Drop-Down-Menü enthält in jeder Zeile einen Eintrag.

In der Menüzeile befinden sich immer die Einträge "Datei" und der Programmname – wie bei den PC-Versionen ("DESK" darf sich nur das Desktop nennen!). Der Eintrag mit dem Programmnamen (in Großbuchstaben) steht bei Atari-GEM ganz links, bei PC-GEM rechts in der Menüleiste. Man weiß also immer sofort, zu welchem Programm die Menüleiste gehört (unter einem Multitasking-GEM sehr wichtig!).

Abgesehen von diesem Titel ist das "Datei"-Menü ganz links zu finden. "Datei" und die anderen Titel sind immer mit einem Großbuchstaben am Anfang gehalten (Ausnahme: Abkürzungen). Jeder Titel besteht aus nur einem Wort und besitzt als Abgrenzung zu den anderen Titeln je ein Leerzeichen rechts und links.

Da es aber nicht nur ein Datei- und Informationsmenü gibt, haben sich weitere Standardtitel eingebürgert. Ein "Edit"- bzw. "Bearbeiten"-Menü befindet sich – sofern vorhanden – rechts vom "Datei"-Menü. Ein "Hilfe"-Menü ist ganz rechts einzuordnen und ein "Parameter"- oder "Optionen"-Menü links neben dem "Hilfe"-Menü. Somit sind schon insgesamt fünf Titel vorbelegt. Die restlichen Titel (maximal acht) können annähernd frei gesetzt werden. Auf verbreitete, gute Vorbilder sollte dabei nach Möglichkeit Rücksicht genommen werden.

Gerade bei den Menütiteln machen sich viele Unsauberkeiten breit. Manch ein Menü hat schon mehr als zehn Titel, oder die Anzahl der Leerzeichen rechts und links von einem Titel weicht

stark von den Konventionen ab. Dabei verlängern unnötig viele viele Leerzeichen die Wege, die mit dem Mauszeiger zurückgelegt werden müssen. Auch die Schreibweise der Titel ist nicht immer schön (nur Großbuchstaben oder "Desk" anstelle des Programmnamens).

Die Drop-Down-Menüs bestehen aus einzelnen Einträgen (jeweils einer pro Zeile), wobei ganz rechts ein Tastaturkürzel zu finden ist. Diesem Tastaturkürzel kann neben einem alphanumerischen Zeichen (Buchstaben, Ziffern) für eine Taste, die stellvertretend für den Eintrag steht, ein besonderes Zeichen (links von dem Buchstaben) zugeordnet werden. Dieses Zeichen ist ein Pfeil nach oben (ASCII 1) für eine der Shift-Tasten, eine Raute (ASCII 7) für die Alternate-Taste oder ein "^" (ASCII 94) für die Control-Taste.

Um die Funktion zu erreichen, die hinter dem Menüeintrag steht, ist dann die jeweilige Sondertaste in Verbindung mit der angegebenen Taste zu drücken.

Die Tastenkürzel werden mit mindestens einem Leerzeichen vom restlichen Eintrag abgesetzt. Der eigentliche Eintrag beginnt immer zwei Zeichen vom linken und eines vom rechten Rand entfernt, um Platz genug für ein Häkchen zu haben. Dabei ist es *nicht* relevant, ob dort überhaupt ein Häkchen plaziert werden soll.

Das Häkchen steht für eine aktivierte Funktion. Solche Menüeinträge mit "Umschaltcharakter" sollten sparsam eingesetzt werden. Bei vielen Einstellungen ist es ratsam, diese in einer Dialogbox zusammenzufassen.

Drei Punkte rechts vom Text eines Eintrags sind immer dann hinzuzufügen, wenn nach einem Klick auf den Eintrag oder nach dem Drücken der zugehörigen Tastenkombination ein Dialog folgt. Einträge, die nicht selektiert werden dürfen, werden vom Programm automatisch auf hell – DISABLED – gesetzt. Dies geschieht auch während des Programmablaufs.

Es hat ja auch wenig Sinn, "sichern" anklicken zu dürfen, wenn nichts zu sichern ist. Sobald eine Wahl des Eintrags wieder sinnvoll ist, wird der Status DISABLED zurückgenommen.

Eine letzte Form der Einträge sind Separatoren. Ein Separator ist eine durchgehende Linie (eine Reihe einfacher Bindestriche), die vom linken bis zum rechten Rand geht. Hier werden ausnahmsweise keine Leerzeichen am linken und rechten Rand freigelassen. Separatoren dienen zur Unterteilung von Einträgen in einem Drop-Down-Menü. Sie fassen mehrere Einträge optisch zu einer Funktionsgruppe zusammen.

Jedes Drop-Down-Menü hat seine typischen Einträge. So hat das Menü mit dem Programmnamen den Info-Eintrag.

Daneben findet man in diesem Menü die Einträge für die verschiedenen Accessories.

Das "Datei"-Menü beinhaltet als untersten Eintrag ("Ende") immer die Möglichkeit, das Programm zu verlassen. Eine Sicherheitsabfrage vor dem Verlassen eines Programms ist nur dann sinnvoll, wenn irgendwelche Änderungen an einer Datei noch nicht gesichert wurden!

Die anderen Einträge befassen sich tatsächlich mit Dateien und lauten von oben nach unten "neu anlegen", "öffnen", "schließen", "sichern", "sichern unter", "abbrechen" und "an Ausgabe". Hierbei bedeutet "neu anlegen", daß ein Arbeitsbereich für eine neue Arbeit unter dem Namen 'NAMENLOS' angelegt wird. Beim Abspeichern ist dann ein Name zu vergeben, da eine Datei ohne Namen logischerweise auf dem Massenspeicher nicht existieren kann. "Sichern" oder auch "abbrechen" sind nach Anwahl dieses Eintrags sinnigerweise gesperrt.

Bei "Öffnen" wird eine bereits bestehende Datei über eine Dateiauswahlbox ausgesucht und geladen, bei "schließen" wird die aktuell bearbeitete Datei geschlossen, ohne gesichert zu werden. Jedoch sollte unbedingt eine Sicherheitsabfrage erfolgen, um Fehlbedienungen auszuschließen.

Das Abspeichern einer Datei wird über "sichern" oder "sichern unter" ermöglicht, wobei bei der zweiten Variante über eine Dateiauswahlbox ein neuer Dateiname gewählt werden kann. "Sichern" kann nur dann gewählt werden, wenn das oberste Fenster ein nicht-namenloses Dokument enthält. Ist eine vorher namenlose Arbeit über "sichern unter" abgespeichert worden, dann ist sie fortan nicht mehr ohne Namen, und die Einträge "sichern" und "abbrechen" können freigegeben werden.

Die Option "sichern unter" läßt sich auch hervorragend dazu einsetzen, einen selektierten Block abzuspeichern. Ist ein Block gekennzeichnet, so fragt die Anwendung nach, ob der gesamte Text oder nur der Block gesichert werden soll.

"Letzte Fassung" setzt alle seit dem Öffnen der Datei gemachten Änderungen wieder zurück, gegebenfalls durch einfaches Neuladen der Datei. Klickt man auf "an Ausgabe", so startet das GEM-Ausgabeprogramm, um beispielsweise den Ausdruck einer Meta-Datei zu bewerkstelligen. Allerdings kann auch die Ausgabe direkt erfolgen.

Einträge wie "ausschneiden", "kopieren", "einfügen" oder "Undo" befinden sich im Bearbeiten- bzw. Edit-Menü.

Unter "Undo" versteht man eine Funktion, die die letzte Aktion wieder rückgängig macht. "ausschneiden" und "kopieren" (engl. "cut" und "copy") bewirken, daß ein gekennzeichneter Block in einen internen Speicher (oder ins Clipboard; über Parameter einstellbar) kopiert wird. "ausschneiden" beinhaltet zusätzlich das Löschen des Blocks. "einfügen" bringt den gespeicherten Block wieder zurück in den Arbeitsbereich. Die drei Grundfunktionen ermöglichen das Löschen eines Blocks (in den Puffer kopieren) und auch das Verschieben (ausschneiden

und an anderer Stelle wieder einfügen), womit zahlreiche Funktionen überflüssig werden können. Es bleibt den Programmierern überlassen, weitere sinnvolle Funktionen zu ergänzen.

Das "Parameter"- oder "Optionen"-Menü enthält Einträge, die zu weiteren Funktionen der Applikation führen. Die Funktionen erlauben es jedem Anwender, die Applikation den eigenen Bedürfnissen anzupassen und diese Einstellungen in einer Datei (mit der Namenserweiterung "INF") zu sichern.

Eine Hilfe zu einer Applikation kann einerseits über die Taste "Help" und andererseits über das "Hilfe"-Menü abgerufen werden. Hier werden zumeist weitere Dialoge mit diversen Hilfstexten angezeigt. Die kleinen Alertboxen sollten nicht benutzt werden, da sie zu wenig Informationsmöglichkeiten bieten. Auch Einträge wie "^ = Control" dürfen nicht aufgeführt werden, da sie einerseits einen Eintrag unnötig belegen und andererseits irreführend sind: obwohl selektierbar, würde mit ihnen keine Funktion ausgelöst werden.

Nun zu den Proportionen des Menüs. Wie schon eingangs erwähnt, sollten maximal acht Drop-Down-Menüs vorhanden sein. Die Titel überschreiten in der Regel eine Breite von etwa 12 bis 16 Buchstaben nicht. In jedem Drop-Down-Menü finden sich höchstens 10 bis 12 Einträge inklusive der Separatoren. Ein Eintrag ist (mit den zwei Leerzeichen zu Beginn und dem Tastenkürzel und einem Leerzeichen am Ende) maximal 20 bis 25 Zeichen breit.

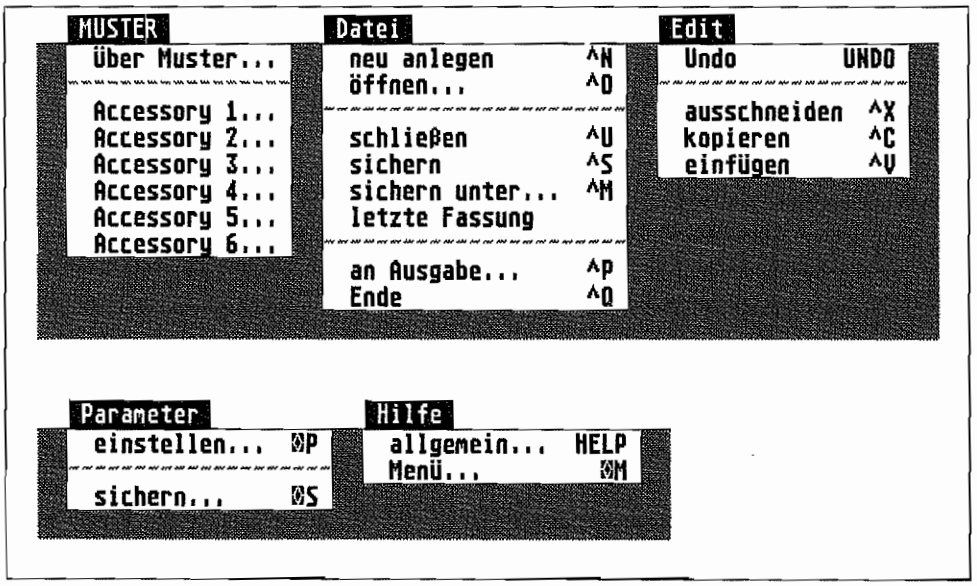


Abb. 6.1: Vorschläge für vorbildliche Menüs

Dialoge

Bei der Gestaltung der Dialogboxen ist auf innere und äußere Konsistenz zu achten. Man orientiert sich an vorbildlichen Dialogboxen anderer Programme (äußere Konsistenz) und achtet darauf, daß innerhalb einer Anwendung die Dialogboxen eine einheitliche Gestaltung haben (innere Konsistenz). Zur inneren Konsistenz gehört zudem noch, daß jede Aktion vorhersehbar ist und nicht etwas völlig Unerwartetes passiert.

Die Lage der Exit-Buttons ist durch die äußere Konsistenz bereits nahezu vorgegeben, da die Dateiauswahlbox die Exit-Buttons rechts unten und die Alertboxen die Buttons rechts (PC-GEM) oder unten (Atari-GEM) haben. Eine Plazierung am oberen oder linken Rand sollte also vermieden werden. Nach Möglichkeit sollten die Buttons in einer Reihe neben- oder übereinander angeordnet werden.

Die Größen der Buttons folgen der einfachen Regel "nicht zu groß und nicht zu klein": in der Höhe die eineinhalbfache Zeichenhöhe und die Breite des breitesten Buttontexts in der Dialogbox nicht wesentlich überschreiten. Der breiteste Button ist so breit wie der Text, der in dem Button steht, zuzüglich einem Leerzeichen am rechten und linken Rand, also maximal Textbreite plus zwei (Ausnahme: ein "pathologischer" Fall wie "OK").

Als Standard-Exit-Buttons sind nahezu immer die Buttons "OK" und "Abbruch" zu finden, wobei "OK" eine Einstellung oder die Kenntnisnahme von einer Meldung bestätigt und "Abbruch" die in diesem Dialogschritt gemachten Änderungen ignoriert. Statt des "OK"-Buttons findet man auch vielfach Buttons, die mit einer bestimmten Aktion wie zum Beispiel "kopieren" gekennzeichnet sind. Damit die innere Konsistenz gewahrt bleibt, werden "OK" und "Abbruch" immer an den gleichen Stellen plaziert. Weitere Standard-Buttons existieren nicht. Aber auch unabhängig von der Lage diverser Objekte innerhalb einer Dialogbox existieren Regeln für die Anwendung eben dieser Objekte, damit die Bedienung einheitlich ist und sich jeder Anwender sofort zurechtfindet.

GEM erlaubt es, Objekttypen wie Boxen (Rechtecke), Texte (veränderbar und nicht veränderbar), Bilder (Images und Icons) sowie Mischformen daraus (Texte in Rechtecken: Boxtext, Button, Boxchar, ...) zu verwenden. Sollte dies noch immer nicht reichen, so wird die Typenvielfalt mit benutzerdefinierten Objekten ergänzt. Alle diese Typen können mit diversen Eigenschaften wie "wählbar" oder "Ausgang" und Status wie "abgehakt" oder "gesperrt" versehen werden.

Ausgang-Buttons (Flag EXIT) befinden sich immer dort, wo auch "OK" und "Abbruch" zu finden sind. Diese Buttons sind mit dem Flag EXIT zu versehen. Bei anderen sollte das EXIT-Flag nur dann gesetzt werden, wenn damit der Dialog nicht beendet wird. Das Flag DEFAULT (alternativ mit der Taste "Return" zu bedienen) darf nur dann gesetzt werden, wenn damit

keine irreversible Funktion – wie das Formatieren einer Diskette – aktiviert wird. Bei nicht umkehrbaren Aktionen sollten generell (über eine Option abschaltbare) Sicherheitsabfragen erscheinen.

Gesperrt (Status DISABLED) sind alle Objekte, die bei dem betreffenden Dialogschritt nicht gewählt werden dürfen. Versteckt (Flag HIDETREE) werden die Objekte, die zu einem gewissen Zeitpunkt nicht nötig sind und nur störend wirken würden. Beispielsweise ist ein Hinweis auf eine Abbruchmöglichkeit eines Kopiervorgangs dann nicht nötig, wenn gar kein Kopiervorgang im Gange ist. Der Status SELECTED signalisiert, daß eine bestimmte Wahl getroffen wurde oder eine Aktion aktiv ist. Daher darf dieser Status nicht an anderer Stelle (beispielsweise für Dialogtitel) verwendet werden.

Texte dienen auf der einen Seite der Information, und auf der anderen Seite ermöglichen sie eine Auswahl – z. B. bei den Dateinamen in der Dateiauswahlbox. Stehen diese Informationstexte als Beschreibung in einem Dialog wie "Größe:", "Datum:", dann stehen die Doppelpunkte untereinander, und die Texte sind bezüglich dieser durch die Doppelpunkte vorgegebenen Linie rechtsbündig ausgerichtet. Bei linksbündigen Texten kann es zu zu großen Abständen kommen, wodurch der Zusammenhang verlorengeht.

Bilder – Sie kennen den Spruch "ein Bild sagt mehr als 1000 Worte"? – werden als Hinweiserklärung oder Abkürzung benutzt, müssen aber verständlich bleiben. Ihr Einsatz kann bei ungeschickter Wahl der Symbole leicht zu Verwirrung führen – wer verbindet schon einen Zauberhut mit einer Löschfunktion?

Ein besonderes Kapitel sind Objekte mit den Flags SELECTABLE und RBUTTON (Radioknopf). Diese Flags lassen sich prinzipiell für jedes Objekt benutzen. Schwierig wird es, wenn zu unterscheiden ist, ob ein Objekt nun wählbar ist oder nicht. Dies muß aus der Gestaltung der Dialogbox klar hervorgehen!

Einfache, längere Texte oder Bilder werden nicht als Radioknöpfe installiert. Diese Möglichkeit bleibt einfachen Rechtecken, Zeichen in Rechtecken oder kurzen Texten wie "ja" oder "nein" vorbehalten. Ein Rechteck ist ein gutes Signal dafür, daß ein wählbares Feld oder ein Radioknopf vorliegt, wobei hier bei der Gestaltung der Dialogbox darauf geachtet werden muß, daß derartige Objekte nicht mit Exit-Buttons verwechselt werden können. Der wohl beste Weg für Radioknöpfe ist, zusammengehörige in einem großen Rechteck unter- oder nebeneinander zusammenzufassen oder spezielle Felder (wie bei Formularen) vor Texten dafür vorzusehen. Einfach wählbare Objekte lassen sich prima darstellen, wenn vor einem Text ein kleines Rechteck zu sehen ist, welches "angekreuzt" werden kann. Allerdings ist dies nur über selbst definierte Objekte möglich.

So vielfältig die Status sein mögen, sie sind sparsam – weniger ist mehr! – zu verwenden. Auch von der Verwendung verschiedener Füllmuster sollte man Abstand nehmen, denn

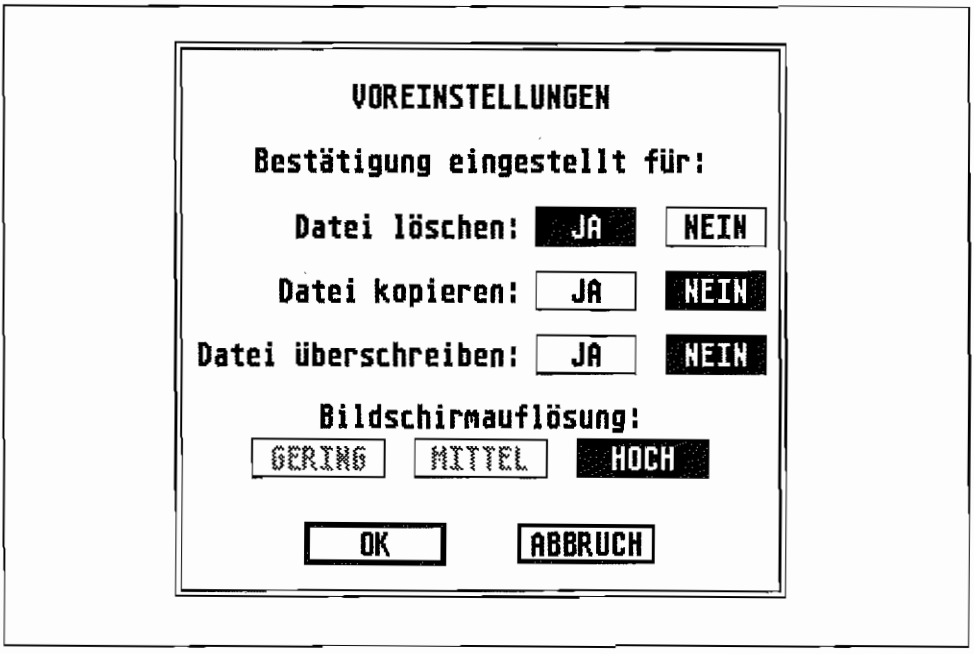


Abb. 6.2: Eine übersichtliche Dialogbox

gerade die kontrastreichen Muster lenken Anwender nur von den eigentlichen Funktionen und Einstellungen ab. Neben der Möglichkeit, Objekte in einfacher Form zu gruppieren, gibt es eine weitere, um komplexe Objekte aufzubauen.

Als Beispiel seien hier Listbox (wie in der GEM-Dateiauswahlbox) und Popup-Menüs (wie in XControl) genannt.

Letztlich die Farben: Sie lockern die Dialogbox auf, lenken jedoch stark ab, wenn eine Dialogbox zu bunt gestaltet ist. Für Texte wird in der Regel schwarz gewählt, Warnungen ließen sich rot darstellen. Insgesamt sollten Farben sehr sparsam verwandt werden!

Werkzeugleiste

Reichen Menüs und Dialoge für die Zugriffsmöglichkeiten auf die Funktionen eines Programms nicht aus, so ergänzen zusätzliche Fenstermenüs oder Popup-Menüs die Anwendung. Die zusätzlichen Kontrollelemente finden in der Regel im rechten Randbereich eines Fensters ihren Platz.

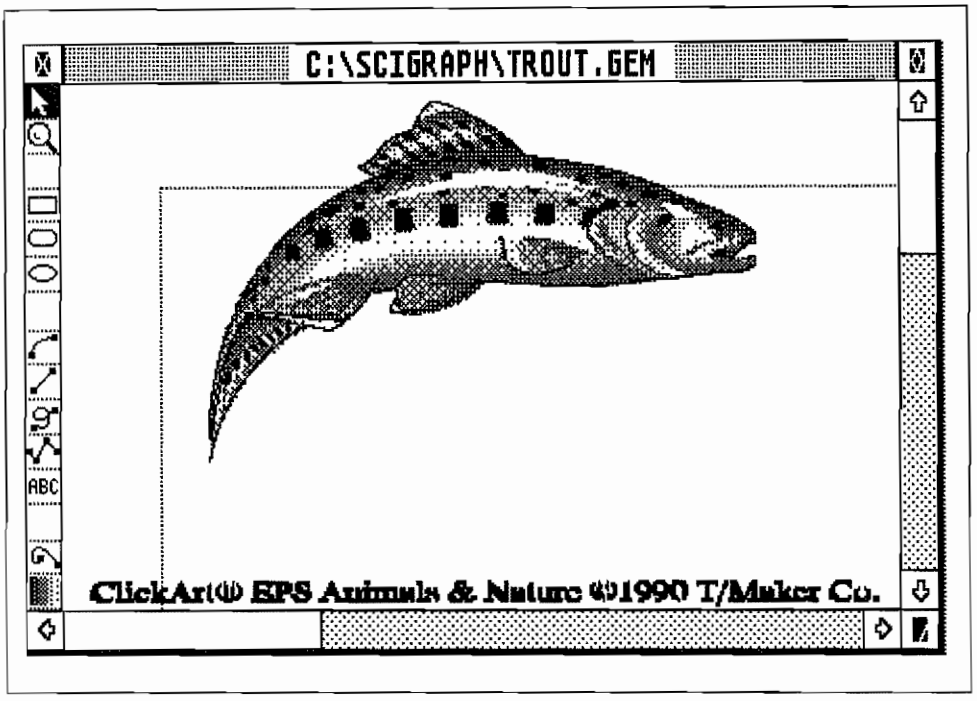


Abb. 6.3: Ein Fenster mit Werkzeugleiste

Es hat sich eingebürgert, diese Menüs mit Bildsymbolen zu gestalten. Je nach Anwendung verbergen sich hinter den Icons weitere Popup-Menüs ("RCS") oder "Schalter" zwischen verschiedenen Betriebsmodi ("SciGraph").

Maus und Mauszeiger

Sie werden sich sicherlich fragen, weshalb man sich mit so unscheinbaren GEM-Elementen wie dem Mauszeiger befassen kann. Aber auch der Mauszeiger hat je nach Form feste Bedeutungen.

Ein Einfachklick dient der Auswahl bestimmter Elemente (Radioknöpfe, Icons) oder dem Zeigen auf diese, ein Doppelklick leitet eine spezielle Funktion ein (Öffnen eines Fensters nach einem Doppelklick auf ein Icon). Nach einem Einfachklick kann in der Regel eine weitere Aktion ausgelöst werden, wie zum Beispiel das Ändern der Farbe eines Grafikobjekts. Prinzi-

piell ist neben dem Einfach- und Doppelklick auch ein Drei- oder Vierfachklick möglich. Da die Ausführung eines Drei- oder Vierfachklicks den meisten Anwendern erhebliche Schwierigkeiten bereiten dürfte, ist diese Form des Mausklicks wenn überhaupt nur optional zu verwenden.

Aus der Einfachklickmöglichkeit ergibt sich ein weiteres Bedienungselement. Läßt man nämlich die Maustaste nicht sofort los, sondern hält sie gedrückt, so können ausgewählte Objekte verschoben oder eine Gruppe von Objekten angewählt werden.

Diese Möglichkeit wird beispielsweise vom Desktop ausgenutzt, um mehrere Dateien selektieren zu können oder Laufwerkssymbole zu verschieben. Aber auch Grafikprogramme nutzen den Klick mit Festhalten, beispielsweise, um Linien zu ziehen.

In Verbindung mit den Tasten "Alternate", "Control" oder "Shift" ergeben sich weitere Möglichkeiten der Mausbenutzung, wobei diese Bedienungselemente nicht a priori klar sind. Üblich ist, daß in Verbindung mit einer Shift-Taste weitere Objekte (auch unter Verwendung der Rubberbox) ausgewählt werden können ("Erweitern der Auswahl"). Die Form bzw. Darstellungsart des Mauszeigers ist je nach Funktion, die gerade ausgeübt werden soll, unterschiedlich:

Darstellungsart	**Funktion**
Pfeil:	allgemeine Bedienung (Regelfall)
Balken:	Texteingabe
Biene bzw. Sanduhr:	der Rechner ist beschäftigt, z. B. mit Speichern einer Datei
Zeigefinger:	Auswahl oder Dimensionierung
flache Hand:	Verschieben von Objekten, Positionierung
Fadenkreuz, dünn:	Zeichnen oder Auswahl
Fadenkreuz, dick:	keine feste Bedeutung
Fadenkreuz, Umriß:	keine feste Bedeutung
unsichtbar:	bei Zeichenoperationen, wenn der Mauszeiger stört, oder bei Rasterkopie
sichtbar:	eigentlich immer, wenn er nicht gerade zum Zeichnen abgeschaltet ist

Unabhängig von der Möglichkeit, den Mauszeiger an- und abschalten zu können, sollte der Mauszeiger immer sichtbar sein.

Auch bei langwierigen Operationen wie dem Kopieren von Disketten bleibt der Mauszeiger – als Biene bzw. Sanduhr – sichtbar!

Tastaturbelegung

Die Finger "gewöhnen" sich nur langsam an die Lage der einzelnen Tasten auf der Tastatur – wer häufig zwischen Rechnern mit deutscher und amerikanischer Tastaturbelegung wechseln muß, kennt das Problem. Gleiches gilt natürlich für Befehlstasten-Kombinationen, mit denen Standardfunktionen aufgerufen werden. Daher ist eine Normierung zumindest der wichtigsten Shortcuts unbedingt nötig. Die aufgelisteten Tastenkombinationen sollen verdeutlichen, was sich bisher eingebürgert hat und somit als Quasistandard anzusehen ist.

Genormt laut Beschluß der Entwicklerkonferenz vom August 1989:

Tastenbelegung	Funktion
Ctrl-C	kopieren ("copy")
Ctrl-F	suchen ("find")
Ctrl-O	öffnen ("open")
Ctrl-Q	Programm beenden ("quit")
Ctrl-V	einfügen ("paste")
Ctrl-X	ausschneiden ("cut")
Shift-"Pfeil oben"	eine Seite zurückblättern
Shift-"Pfeil unten"	eine Seite vorwärtsblättern
Shift-"Pfeil links"	Einfügemarke zum Zeilenbeginn
Shift-"Pfeil rechts"	Einfügemarke zum Zeilenende
Ctrl-"Pfeil links"	um ein Wort zurück
Ctrl-"Pfeil rechts"	um ein Wort vorwärts
Home	Dokumentanfang
Shift-Home (Clr)	Dokumentende

Weitere übliche Tastenkombinationen:

Tastenbelegung	Funktion
Ctrl-A	alles auswählen
Ctrl-G	nächste Fundstelle
Ctrl-M	Sichern unter...
Ctrl-N	Neues Dokument
Ctrl-P	Drucken
Ctrl-S	Sichern
Ctrl-R	Ersetzen
Ctrl-U	Oberstes Fenster schließen

Tastenbelegung	Funktion
Ctrl-W	zum nächsten Fenster blättern
Ctrl-Y	aktuelle Zeile ausschneiden
Ctrl-Z	Shell starten

Wer schon einmal an einer englischen Tastatur gesessen hat, der kennt die Probleme, die sich aus der Vertauschung der Tasten Y und Z ergeben. Aber nicht nur bei den üblichen Buchstaben-Tasten spielt die Gewohnheit eine große Rolle: Auch bei Tastenkombinationen für Befehle spielt die Gewöhnung eine sehr große Rolle. Um Problemen mit unterschiedlichen Scancodes aus dem Wege zu gehen, empfiehlt es sich, nach Möglichkeit immer den ASCII-Code des Zeichens zu benutzen. Für Tastenkombinationen mit "Alternate" kann man die XBIOS-Funktion "Keytbl()" zu Rate ziehen. Eine kleine Routine nach einer Vorlage von Ken Badertscher zur unabhängigen Auswertung der Tastatur:

```
/* Bits zur Kennzeichnung der Shift-Status */
#define KsCAPS          0x10
#define KsALT           0x08
#define KsCONTROL       0x04
#define KsSHIFT         0x03
#define KsLSHIFT        0x02
#define KsRSHIFT        0x01

/*
 * Masken für die Status-Bits, die im oberen Byte des von MapKey
 * zurückgegebenen Words plaziert sind
 */
#define KbSCAN          0x8000
#define KbNUM           0x4000
#define KbALT           0x0800
#define KbCONTROL       0x0400
#define KbSHIFT         0x0300
#define KbLSHIFT        0x0200
#define KbRSHIFT        0x0100
/* Scan-Codes für die Tasten, deren ASCII-Code 0 ist */

/*
 * ISO Taste (erscheint, wenn ein Nicht-US-Keyboard mit US TOS
 * benutzt wird)
 */
#define KbISO           0x37
```

```
/* Funktionstasten */
#define KbF1            0x3b
#define KbF2            0x3c
#define KbF3            0x3d
#define KbF4            0x3e
#define KbF5            0x3f
#define KbF6            0x40
#define KbF7            0x41
#define KbF8            0x42
#define KbF9            0x43
#define KbF10           0x44

/* Shift-Funktionstasten */
#define KbF11           0x54
#define KbF12           0x55
#define KbF13           0x56
#define KbF14           0x57
#define KbF15           0x58
#define KbF16           0x59
#define KbF17           0x5a
#define KbF18           0x5b
#define KbF19           0x5c
#define KbF20           0x5d

/* Cursor-Bereich */
#define KbUNDO          0x61
#define KbHELP          0x62
#define KbINSERT        0x52
#define KbHOME          0x47
#define KbUP            0x48
#define KbDOWN          0x50
#define KbLEFT          0x4b
#define KbRIGHT         0x4d
/* Alternate-numerische Taste */
#define KbAlt1          0x78
#define KbAlt2          0x79
#define KbAlt3          0x7a
#define KbAlt4          0x7b
#define KbAlt5          0x7c
#define KbAlt6          0x7d
#define KbAlt7          0x7e
```

```
#define KbAlt8          0x7f
#define KbAlt9          0x80
#define KbAlt0          0x81

typedef struct
{
     char *unshift;
     char *shift;
     char *caps;
} KEYTABLE;

KEYTABLE *kt;

/* Ändert eine VDI-Taste (Rückgabewert von evnt_keybd())
   zu einem Word-großen, codierten Zeichen:

   Highbyte Lowbyte
   -------- --------   ACLR sind Shift-Status-Bits
   SxxxACLR CHARCODE = ASCII (S = 0)- oder Scan (S = 1)-Code

    Mit dieser Funktion wird das diffizile Problem der internatio-
    nalen Tastaturen und der Umwandlung der länderspezifischen VDI-
    Tasten in unabhängige ASCII- oder Scan-Codes gelöst. */

WORD MapKey (WORD key)
{
     WORD keystate, scancode, ret;

     /* Ermittlung der Tastaturtabellen, falls noch nicht
        geschehen */
     if (!kt)
          kt = (KEYTABLE *)Keytbl ((VOID*) -1L, (VOID*) -1L,
               (VOID*) -1L);
     /* Feststellung des Scan-Codes und der Shift-Status
        (ohne die Shift-Status von evnt_multi oder graf_mkstate) */
     scancode = (key >> 8) & 0xFF;
     keystate = (WORD) Kbshift (-1);

     /* Prüfung des ASCII-Codes mit der geeigneten Tabelle */
     /* Anpassung der Alternate-numerischen Tasten */
```

```
    if ((keystate & KsALT) && (scancode >= 0x78)
         && (scancode <= 0x83))
         scancode -= 0x76;
    if (keystate & KsCAPS)
    {
         ret = kt->caps[scancode];
    }
    else
    {
         if (keystate & KsSHIFT)  /* Shift-Funktionstasten haben
                                     korrespondierende Scan-Codes */
              ret = kt->shift[((scancode >= KbF11) &&
                  (scancode <= KbF20))? scancode - 0x19:scancode];
         else
              ret = kt->unshift[scancode];
    }


    /* Rückgabe der Scancodes, zu denen es keinen ASCII-Code gibt,
       und Kennzeichnung der numerischen Tastatur */
     if (!ret)
         ret = scancode | KbSCAN;
    else if ((scancode == 0x4a) || (scancode == 0x4e) ||
         ((scancode >= 0x63) && (scancode <= 0x72)))
         ret |= KbNUM;

    return (ret | (keystate << 8));
}
```

Selektion

Wie man Objekte auswählt, macht das Desktop bereits vor: einzelne Objekte durch einfaches Anklicken, mehrere Objekte durch Shift-Klick oder die Rubberbox. Eine spezielle Methode zum Auswählen ist das "Echtzeit"-Selektieren.

Viele Programme, wie "Wordplus", "Edison" oder "Turbo C" verfolgen dieses Verfahren. Kernpunkt ist, daß der aktuell ausgewählte Block bereits während der Mausbewegung invertiert wird (um so besser, wenn beim Erreichen des Fensterrandes auch automatisch gescrollt wird). Weitere Varianten erlauben das wortweise Markieren (Markiervorgang mit einem Doppelklick beginnen) und vieles mehr.

Bei Programmen, die diese im "Inside Macintosh" beschriebene Methode konsequent nutzen, gibt es immer nur einen Block oder eine Einfügemarke – nie aber beides gleichzeitig. Das ist kein Fehler, sondern ist im Konzept dieses Mechanismus begründet: Alles dreht sich um die aktuelle "Selektion" (damit ist der gerade markierte Block gemeint). Der Cursor (besser: die Einfügemarke) ist nur ein Sonderfall eines Blocks (nämlich einer mit der Länge Null).

Auch beim Einfügen gibt es keine Unterscheidung zwischen Blöcken und einzelnen Zeichen (ein Zeichen ist eben ein Block der Länge Eins). Der einzufügende Block wird immer anstelle des gerade selektierten Blocks eingesetzt. Anschließend erscheint die Einfügemarke (also der "leere" Block) rechts des eingefügten Blocks. Das gesamte Macintosh-Blockkonzept beruht somit auf dem Einfügen und Entfernen von Blöcken: Gute Ideen erkennt man daran, daß sie einfach sind!

Einige "Blöcke", die man einfügt, haben natürlich besondere Funktionen. "Backspace" und "Delete" löschen den selektierten Block. Falls nichts selektiert ist, wird vorher entweder das Zeichen rechts oder links der Einfügemarke selektiert. Ähnlich intuitiv sind auch andere Funktionen wie das Finden von Textstellen (der gefundene Text wird zum aktuell selektierten Block und kann damit direkt überschrieben werden) oder das Löschen von Zeilen (aktuelle Zeile als Block markieren und dann ausschneiden). Und in diesem Zusammenhang werden auch die einzelnen Menüpunkte des "Edit"-Menüs leicht verständlich. "Ausschneiden" entfernt den aktuellen Block und kopiert ihn in die "Zwischenablage", einen internen Puffer. "Kopieren" kopiert den selektierten Block in die "Zwischenablage", ohne ihn zu löschen. "Einfügen" fügt den momentanen Inhalt der Zwischenablage in das Dokument ein. "Löschen" löscht den aktuellen Block, "Alles auswählen" selektiert alle Objekte im Dokument. "Undo" schließlich macht die letzte Blockoperation rückgängig.

Die Zwischenablage enthält immer nur den zuletzt hineinkopierten Block. Viele Programme erlauben es, beim Ausschneiden und Kopieren durch Festhalten der Shift-Taste den Block an den Inhalt der Zwischenablage anzuhängen, statt ihn zu ersetzen. Ob die Zwischenablage nun als programminterner Puffer oder direkt über das GEM-Klemmbrett (siehe Beschreibung der Scrap-Funktionen) verwirklicht wird, ist zunächst egal. Falls nicht sowieso das GEM-Klemmbrett (also der Weg über Dateien im Scrap-Verzeichnis) benutzt wird – die heutigen schnellen Festplatten machen das ja ohne weiteres möglich – sollte man diesen Weg zumindest als Option anbieten. "SciGraph" beispielsweise bietet für diese Umschaltung den Menüpunkt "Klemmbrett benutzen" an.

Feedback

Feedback bedeutet: dem Anwender Informationen über die ablaufenden Arbeitsschritte geben. Ein derartiges Feedback läßt sich auf vielerlei Art und Weise erreichen. Dazu gehört,

daß Menütitel während der gesamten Ausführung einer Funktion, die einem Eintrag in diesem Menü zuzuordnen ist, invers dargestellt bleiben. Auch die wachsenden und schrumpfenden Rechtecke vor bzw. nach einem Dialog stellen eine Methode des Feedback dar und sind besonders dann nützlich, wenn per Tastaturshortcut auf ein selektiertes Objekt zugegriffen wird.

Die Form des Mauszeigers signalisiert, was gerade zu erledigen ist. Dauert die Ausführung einer Funktion länger an, so hat der Mauszeiger die Form einer Biene. Befindet sich der Mauszeiger über einem Texteingabefeld, so wird er als gesplitteter Balken dargestellt.

Bei längeren Vorgängen wie dem Kopieren von Dateien oder dem Formatieren von Disketten signalisiert eine Dialogbox, welche Datei gerade kopiert wird oder wie weit der Formatiervorgang fortgeschritten ist. Diese Anzeige ist nach Möglichkeit grafisch zu gestalten.

Allgemeine Regeln

Die Bedeutung und Anwendung wichtiger GEM-Komponenten wurde bereits angesprochen. Nicht nur der korrekte Einsatz der Komponenten, sondern auch das Zusammenspiel ist eine Herausforderung an den Programmierer. GEM wurde unter anderem deshalb geschaffen, um den Menschen, der an der Maschine sitzt, nicht von der eigentlichen Aufgabe abzulenken. Wer einen Brief schreibt, will nicht erst zahlreiche Kommandos erlernen.

Auch die Kommandoebene, die ein Erlernen der Kommandos erfordert, soll einleuchtend gestaltet sein. Es ist eben einfacher, ein Dateisymbol auf ein Laufwerk zu ziehen, um eine Datei zu kopieren, als "`COPY A:\DATEI.XYZ B:\ORDNER\`" einzugeben.

GEM bietet die Möglichkeit, mit Bildern in Form von Icons die Arbeit zu erleichtern. Zudem ermöglichen die vielseitigen Ausgabefunktionen, das WYSIWYG-Prinzip (What You See Is What You Get) einzuhalten. Bei einer Textverarbeitung lassen sich z. B. Buchstaben, die auf dem Papier fett dargestellt werden, auch schon auf dem Bildschirm in fetter Schrift darstellen. GEM-Programme müssen immer so geartet sein, daß einem Anwender sofort klar ist, was etwa nach einem Klick auf einen Menüpunkt geschieht.

Auch irreversible Eingriffe dürfen erst gar nicht entstehen oder benötigen zumindest eine Sicherheitsabfrage.

Jede Komponente des GEM (Menüs, Dialoge, Fenster) hat seine feste Bedeutung. Programme lassen sich leichter bedienen, wenn es bei der Gestaltung der Oberfläche nicht zu Widersprüchen kommt. Die korrekte Ausnutzung dieser Komponenten gehört zu jedem GEM-Programm. Selbstgestrickte Menüs, Dialoge und Fenster, die auf den ersten Blick schön oder "modern" erscheinen mögen, erschweren den Menschen vor der Maschine Computer nur die Arbeit und schrecken auf Dauer ab.

Programmtechnisches

Einleitung

Um es vorwegzunehmen: In diesem Kapitel soll kein vollständiges GEM-Programm mit Menü-, Dialog- und Fensterverwaltung entwickelt werden. Ein derartiges Beispielprogramm würde zu lang und könnte nicht allgemein genug gehalten werden. Das Hauptaugenmerk liegt vielmehr auf einigen Anregungen und Tips, die bei der Programmierung unter GEM Beachtung finden sollten.

Ein Rahmenprogramm

Programmierung unter GEM setzt die Kenntnis der Funktionen und Strukturen voraus, Feinheiten wie die Parameter der einzelnen Funktionen sind jedoch nicht unbedingt nötig. Nur wer die Funktionen kennt und weiß, wie und wann sie eingesetzt werden, kann sichere GEM-Programme schreiben. Bevor Sie also zur Programmierung schreiten, lesen Sie die Kapitel mit den AES-Funktionen einmal – zumindest diagonal – und informieren Sie sich, was das VDI alles zu bieten hat. Schenken Sie beim VDI den Auskunftsfunktionen besonderes Augenmerk!

Die Basis für alle GEM-Programme ist die An- und Abmeldung beim AES. Hierzu finden die Funktionen "appl_init()" und "appl_exit()" Verwendung. Die folgenden Programme in C und Modula-2 zeigen, wie der Minimalrahmen für ein Programm auszusehen hat:

1. C-Programm:

```
#include <aes.h>
#include <portab.h>

WORD apl_id;

void main (void)
{
     if ((apl_id = appl_init()) >= 0)
     {
          /* hier kommt das eigentliche Programm... */

          appl_exit();
     }
}
```

2. Modula-2-Programm:

```
MODULE Programm;

FROM AES IMPORT ApplInit, ApplExit;

VAR AplId : INTEGER;

BEGIN
  AplId := ApplInit();
  IF AplId >= 0
  THEN

    (* hier kommt das eigentliche Programm... *)

    ApplExit()
  END
END Programm.
```

Diesen Minimalrahmen ergänzt bei einem Accessory ein Aufruf der Funktion "menu_register()" (für den Eintrag im ersten Drop-Down-Menü). In diesem Zusammenhang sei darauf verwiesen, daß man mit keiner GEM-Funktion abfragen kann, ob eine ausführbare Datei als Accessory oder Programm gestartet wurde. Funktionen wie "Accessory()" bei Megamax Modula-2 oder Variablen wie "_app" bei Turbo C gehören nicht zum GEM, aber es ist von Atari dokumentiert, wie man solche Informationen ermittelt (siehe AES-Einleitung).

Da Accessories GEM-typisch ereignisorientiert sind, erzwingt dies eine Verwendung von "evnt_mesag()" oder "evnt_multi()". Der Aufruf der Funktion "appl_exit()" entfällt, da Accessories unter keinen Umständen verlassen werden dürfen. Daher endet der Programmablauf im Fehlerfall in einer Endlosschleife, in der "evnt_mesag()" aufgerufen wird.

Der Rahmen nimmt somit folgende Gestalt an:

1. C-Programm:

```
#include <aes.h>
#include <portab.h>

char acc_name[] = "  Accessory...";  /* Immer zwei Leerzeichen am
                                        Anfang! */
WORD apl_id, menu_id;
```

```
void main (void)
{
     WORD msg_buffer[8];

     if ((apl_id = appl_init()) >= 0)
     {
          if ((menu_id = menu_register(apl_id, acc_name) >= 0)
          {
               for (;;) /* Endlosschleife */
               {
                    evnt_mesag (msg_buffer);
                    switch (msg_buffer[0])
                    {
                         case AC_OPEN:
                              if (msg_buffer[4] == menu_id)
                              {
                                  /* Accessory wurde geöffnet... */
                               }
                              break;
                         case AC_CLOSE:
                              if (msg_buffer[3] == menu_id)
                              {
                                /* Accessory wurde geschlossen...*/
                               }
                              break;
                    }
               }
          }
     }
     for (;;) evnt_mesag (msg_buffer);
}
```

2. Modula-2-Programm:

```
MODULE Accessory;
FROM AES     IMPORT ApplInit, MenuRegister, EvntTimer, EvntMesag,
                    KAcOpen, KAcClose;
FROM Strings IMPORT Assign, String;

CONST AccName = "  Accessory...";
VAR   AplId,
```

```
      MenuID        : INTEGER;
      AccNameGlobal : String;
      MsgBuffer     : ARRAY [0..7] OF INTEGER;

BEGIN
  AplId := ApplInit();
  IF AplId >= 0
  THEN
    Assign(AccName, AccNameGlobal);
    MenuId := MenuRegister(AplId, AccNameGlobal);
    IF MenuId >= 0
    THEN
      WHILE TRUE
      DO
        EvntMesag(MsgBuffer);
        CASE MsgBuffer[0] OF
          KAcOpen :
            IF MsgBuffer[4] = MenuId
            THEN
              (* das Accessory wurde angewählt... *)
            END|
          KAcClose :
            IF MsgBuffer[3] = MenuId
            THEN
              (* das Accessory wurde geschlossen... *)
            END
        END
      END
    END
  END;
  WHILE TRUE
  DO
    EvntMesag(MsgBuffer)
  END
END Accessory.
```

Fortan werden in erster Linie nur noch C-Beispiele benutzt, da die Ableitung von Modula-2- oder Pascal-Programmen klar sein dürfte.

Die Namensbildung wird an anderer Stelle angesprochen, eine Kurzbeschreibung der C-Syntax befindet sich im Anhang.

Im Hinblick auf die Benutzung der VDI-Funktionen – sei es für Grafikausgabe in Fenstern, sei es für Objekte vom Typ G_USERDEF – ist das Öffnen einer virtuellen VDI-Workstation wichtig. Die Anmeldung erfolgt, indem das Handle der physikalischen Workstation via "graf_handle()" abgefragt, ein Array mit passenden Werten gefüllt und die Funktion "v_opnvwk()" aufgerufen wird. Eine Fehlerabfrage geschieht durch Überprüfung des neuen Handles, das im Fehlerfall den Wert Null hat.

Hier das Öffnen einer virtuellen Workstation auf einem Bildschirm, mit allen Grafiktypen und Rasterkoordinaten:

```
v_handle = graf_handle (&gr_hwchar, &gr_hhchar, &gr_hwbox,
                        &gr_hhbox);
for (i = 0; i < 10; work_in[i++] = 1);
work_in[10] = 2;
v_opnvwk (work_in, &v_handle, work_out);
if (v_handle != 0)
{
  ...
}
```

Geschlossen wird diese virtuelle Workstation mit:

```
v_clsvwk (v_handle);
```

Nach dem Anmelden beim AES und dem Öffnen der virtuellen Workstation für die Grafikausgaben (falls nötig), steht nun der Programmierung nichts mehr im Wege. Als nächstes soll die Frage nach dem eigentlichen Programmablauf gestellt werden.

Programmsteuerung über Ereignisse

Im Gegensatz zu der mehr traditionellen Programmierung mit Warte- und Abfrageschleife, Ganzseitenmenüs und vollständiger Kontrolle des Bildschirms und der anderen Ausgabegeräte durch das Programm wendet man sich bei der Programmierung unter GEM einer Orientierung auf Ereignisse zu. Diese Art der Programmierung findet sich auch bei den anderen, modernen Systemen wie etwa dem Presentation Manager unter OS/2 oder dem X-Window-System unter Unix.

Ein ereignisorientiertes Programm schlummert so lange, bis es angestoßen wird. Ein solcher Anstoß – ein Ereignis – erfolgt je nach den Wünschen der Programmierer. GEM-Programme können auf Ereignisse unterschiedlichster Art reagieren:

- Tastatur-Ereignisse
- Mausknopf-Ereignisse
- Maus-Ereignisse
- Mitteilungs-Ereignisse
- Zeit-Ereignisse

Die einzelnen Ereignisse werden je nach ihrer Verwendung getrennt oder gesammelt abgefragt. Ein Zeit-Ereignis etwa bei längeren Operationen wird mit "evnt_timer()" regelmäßig abgerufen, um anderen Applikationen etwas Zeit zu geben. Ein Accessory kann sich auf das Warten auf eine Nachricht beschränken, statt mit einem "evnt_multi()" auch noch (unnötige) Zeit-Ereignisse zu empfangen.

Sobald die Programme komplexer gestaltet sind, kommt nur noch der Aufruf von "evnt_multi()" in Frage, denn hier muß ein Programm auf Ereignisse unterschiedlichster Art reagieren. Da wird mit der Maus auf ein Objekt geklickt (Mausknopf-Ereignis), ein Menüeintrag gewählt (Mitteilungs-Ereignis), ein Zeichen über Tastatur eingegeben (Tastatur-Ereignis) oder auch ein Fenster verschoben (Mitteilungs-Ereignis). Die möglichen Ereignisse sind in der AES-Einleitung dokumentiert.

Um den prinzipiellen Aufbau einer Ereignisabfrage zu verstehen, ist es wichtig, den Unterschied zu der traditionellen Programmierung zu verstehen. Beachten Sie dabei aber unbedingt, daß ein Programm erst auf ein Ereignis reagiert und nicht etwa in einer Schleife fortlaufend die Tastatur abfragt, ob eine Taste gedrückt wurde!

vorher:

```
LOOP
  BusyRead(c);
  CASE c OF
    Space : EXIT|
    ...
  END
END
```

nachher:

```
LOOP
  EvntKeybd(c);
  CASE c OF
    Space : EXIT|
    ...
  END
END
```

Der wesentliche Unterschied der kurzen Modula-2-Routinen zwischen "vorher" und "nachher" besteht darin, daß die Schleife bei "vorher" ununterbrochen durchlaufen wird, bei "nachher" jedoch pro gedrückte Taste nur einmal! Auch ist nur bei dem Aufruf von

"evnt_keybd()" ein Weiterarbeiten anderer, parallel laufender GEM-Applikationen möglich. Der Unterschied der beiden Tastaturabfragen mag auf den ersten Blick nicht offensichtlich sein, doch stellen Sie sich einmal vor, es ist auch noch in regelmäßigen Zeitabständen eine bestimmte Funktion aufzurufen. Im ersten Fall müßte dann neben einer eventuell gedrückten Taste auch noch die Uhrzeit abgefragt werden, im zweiten ist es aber nur erforderlich, "evnt_multi()" anstelle von "evnt_keybd()" zu verwenden.

Hier ein kleines Beispiel zur Ereignisabfrage:

```
for (;;) /* Endlosschleife */
{
     evnt = evnt_multi (MU_KEYB|MU_BUTTON|MU_M1|MU_M2|MU_MESAG|
                        MU_TIMER,
                  ev_mbclicks, ev_mbmask, ev_mbstate,
                  ev_mm1flags, ev_mm1x, ev_mm1y, ev_mm1width,
                  ev_mm1height,
                  ev_mm2flags, ev_mm2x, ev_mm2y, ev_mm2width,
                  ev_mm2height,
                  ev_mmgpbuff,
                  ev_mtlocount, ev_mthicount,
                  ev_mmox, ev_mmoy, ev_mmobutton, ev_mmokstate,
                  ev_mkreturn,
                  ev_mbreturn);

     if (evnt & MU_KEYBD)
          hdle_keybd (ev_mkreturn);      /* gedrückte Taste */

     if (evnt & MU_BUTTON)
          hdle_button (ev_mmox, ev_mmoy, /* Position des
                                            Mauszeigers */
               ev_mmobutton,     /* Status der Maustasten */
               ev_mmokstate,     /* Status der Umschalttasten */
               ev_mbreturn);     /* Anzahl der Mausklicks */

     if (evnt & MU_M1)
          hdle_m1 (ev_mmox, ev_mmoy,  /* Position des Mauszeigers*/
                ev_mmobutton,         /* Status der Maustasten */
                ev_mmokstate);        /* Status der Sondertasten */

     if (evnt & MU_M2)
          hdle_m2 (ev_mmox, ev_mmoy,  /* Position des Mauszeigers*/
```

```
                ev_mmobutton,          /* Status der Maustasten */
                ev_mmokstate);         /* Status der Sondertasten */

    if (evnt & MU_MESAG)
        hdle_mesag (ev_mmgpbuf); /* Adresse des Messagepuffers */

    if (evnt & MU_TIMER)
        hdle_timer ();
}
```

Da ein "evnt_multi()"-Aufruf nicht notwendigerweise nur genau ein Ereignis registriert, ist eine Benutzung einer Exklusiv-Auswahl wie mit

```
switch (evnt)
{
   case .. :
     break;
   case .. :
     break;
   ...
}
```

nicht möglich! Bei einer Abfrage, welches Mitteilungs-Ereignis (MU_MESAG) aufgetreten ist, kann jedoch eine derartige Abfrage wieder erfolgen, da "evnt_multi()" nur maximal eine Mitteilung abgibt. Auf die einzelnen Ereignisse wird sinnvollerweise in gesonderten Funktionen reagiert, da so auch eine logische Aufteilung der Ereignisse im Programmlisting erreicht werden kann. Die Position des Mauszeigers und den Status der Sonder- und Maustasten wird von "evnt_multi()" übrigens *immer* zurückgegeben. Daher können diese Werte auch an alle Funktionen weitergereicht werden.

Fensterverwaltung

Im Rahmen der Fensterverwaltung wollen wir an die oben besprochene Steuerung des Programms über Ereignisse anknüpfen.

Für die Fensterverwaltung ist die in dem Beispiel erwähnte Routine "hdle_mesag()" von Interesse. Ihr wird die Adresse des Messagepuffers übergeben, der – im Falle eines Mitteilungs-Ereignisses – die relevante Information enthält.

Die Funktion "hdlw_mesag()" könnte folgenden Aufbau haben:

```
void hdle_mesag (WORD *ev_mmgpbuf)
{
     switch (ev_mmgpbuf[0])
     {
          case AC_OPEN:
            /* Accessory wurde angewählt... */
            /* Dies könnte das Öffnen eines Fensters bedingen. */
            break;

          case AC_CLOSE:
            /* Accessory wurde geschlossen... */
            /* Dies könnte das Schließen eines Fensters bedingen. */
            break;

          case MN_SELECTED:
            /* Menüverwaltung... */
            /* Dies könnte das Öffnen eines Fensters bedingen. */
            break;

          /* ...und hier die Fensterverwaltung! */

          /* falls Redraw-Meldung... */

           case WM_REDRAW: /* ID und Koordinaten übergeben */
                wind_redraw (ev_mmgpbuf[3], ev_mmgpbuf[4],
                             ev_mmgpbuf[5], ev_mmgpbuf[6],
                             ev_mmgpbuf[7]);
                break;

          /* falls Fenster aktiviert wurde... */

           case WM_TOPPED:
                wind_topped (ev_mmgpbuf[3]); /* ID des Fensters */
                break;

          /* falls Fenster geschlossen werden soll... */

           case WM_CLOSED:
                wind_closed (ev_mmgpbuf[3]); /* ID des Fensters */
                break;
```

```
/* falls Fenster auf volle Größe gebracht werden soll */

 case WM_FULLED:
      wind_fulled (ev_mmgpbuf[3]); /* ID des Fensters */
      break;

/* falls einer der Pfeile angeklickt wurde... */

 case WM_ARROWED:
      wind_arrowed (ev_mmgpbuf[3],  /* ID des Fensters */
           ev_mmgpbuf[4]); /* Art der Veränderung */
      break;

/* falls der horizontale Slider bewegt wurde... */

 case WM_HSLID:
     wind_hslid (ev_mmgpbuf[3],  /* ID des Fensters */
           ev_mmgpbuf[4]); /* Position in Promille */
      break;

/* falls der vertikale Slider bewegt wurde... */

 case WM_VSLID:
     wind_vslid (ev_mmgpbuf[3],  /* ID des Fensters */
           ev_mmgpbuf[4]); /* Position in Promille */
      break;

/* falls die Größe des Fensters verändert wurde... */

 case WM_SIZED:
     wind_sized (ev_mmgpbuf[3],  /* ID des Fensters */
          ev_mmgpbuf[4],  /* x-Koordinate des Fensters */
          ev_mmgpbuf[5],  /* y-Koordinate des Fensters */
          ev_mmgpbuf[6],  /* neue Breite */
          ev_mmgpbuf[7]); /* neue Höhe */
      break;

/* falls das Fenster bewegt wurde... */

case WM_MOVED:
     wind_moved (ev_mmgpbuf[3],  /* ID des Fensters */
```

```
                    ev_mmgpbuf[4],  /* neue x-Koordinate */
                    ev_mmgpbuf[5],  /* neue y-Koordinate */
                    ev_mmgpbuf[6],  /* Breite */
                    ev_mmgpbuf[7]); /* Höhe */
                break;
    }
}
```

Die in diesem Beispiel aufgeführten wind-Funktionen sind wiederum durch die Programmierer zu deklarieren und beinhalten Aufrufe der AES-Fensterfunktionen. Den jeweiligen Fensteraktionen können beispielsweise folgende AES-Funktionen zugeordnet werden:

- beim Anlegen eines Fensters
  (legt die Fensterstruktur an, dies beinhaltet kein Öffnen des Fensters!)

wind_create ():	legt maximale Größe fest und liefert ID
wind_calc ():	Größe berechnen

- beim Öffnen eines Fensters

wind_open ():	öffnet das Fenster, setzt die Anfangsmaße
wind_set (WF_NAME, ...):	setzt Fensternamen
wind_set (WF_INFO, ...):	setzt Infozeile
wind_set (WF_CURRXYWH, ...):	setzt Fenstergröße
wind_set (WF_HSLIDE, ...):	setzt Position des horizontalen Sliders
wind_set (WF_VSLIDE, ...):	setzt Position des vertikalen Sliders
wind_set (WF_HSLSIZE, ...):	setzt Größe des horizontalen Sliders
wind_set (WF_VSLSIZE, ...):	setzt Größe des vertikalen Sliders

- bei Ausgaben in ein Fenster

wind_get (WF_WORKXYWH, ...):	Größe des Arbeitsbereichs
wind_get (WF_HSLIDE, ...):	Position des horizontalen Sliders
wind_get (WF_VSLIDE, ...):	Position des vertikalen Sliders
wind_get (WF_FIRSTXYWH, ...):	erstes Rechteck der Rechteckliste
wind_get (WF_NEXTXYWH, ...):	nächstes Rechteck der Rechteckliste
wind_get (WF_HSLSIZE, ...):	Größe des horizontalen Sliders
wind_get (WF_VSLSIZE, ...):	Größe des vertikalen Sliders
wind_set (WF_TOP, ...):	Fenster wird zum aktuellen
wind_update (BEG_UPDATE):	Beginn einer Ausgabe/Erneuerung
wind_update (END_UPDATE):	Ende einer Ausgabe/Erneuerung

– bei einer Größenveränderung

wind_get (WF_CURRXYWH, ...):	aktuelle Größe
wind_get (WF_PREVXYWH, ...):	bisherige Größe
wind_get (WF_FULLXYWH, ...):	maximale Größe
wind_set (WF_CURRXYWH, ...):	setzt Fenstergröße
wind_set (WF_HSLIDE, ...):	setzt Position des horizontalen Sliders
wind_set (WF_VSLIDE, ...):	setzt Position des vertikalen Sliders
wind_set (WF_HSLSIZE, ...):	setzt Größe des horizontalen Sliders
wind_set (WF_VSLSIZE, ...):	setzt Größe des vertikalen Sliders

– Verschieben eines Fensters

wind_set (WF_CURRXYWH, ...):	setzt Fensterkoordinaten

– Schließen eines Fensters

wind_close ():	schließt das Fenster

– Löschen eines Fensters (entfernt die Fensterstruktur aus dem Speicher!)

wind_delete ():	gibt die ID wieder frei

Dialogverwaltung

Zum Aufruf eines Dialogs sind verschiedene Schritte nötig. Zum einen gehören dazu Elemente des Feedbacks – die größer bzw. kleiner werdenden Rechtecke – und zum anderen die programmtechnisch erforderlichen Elemente.

Die Verwaltung eines Dialogs teilt sich grundsätzlich in drei Teile auf:

1. Vorbereitung
2. Durchführung
3. Nachbereitung

Zur Vorbereitung gehört, daß kein anderes Programm weiter Ausgaben tätigt, die Mauskontrolle bei der Applikation liegt, die zu der Dialogbox gehört, und der Hintergrund reserviert wird. Die Durchführung beinhaltet den Aufruf der eigentlichen Verwaltungsroutine. Zuletzt die

Nachbereitung, die den Hintergrund wieder freigibt, die Kontrolle der Maus abgibt und anderen Applikationen die Ausgabe wieder erlaubt. Somit entsteht folgende Minimalfunktion:

```
WORD do_dialog (OBJECT *dial)
{
     GRECT size;
     WORD  rc;

     /* Vorbereitung */

     wind_update (BEG_UPDATE);
     wind_update (BEG_MCTRL);
     form_center (dial, &size.g_x, &size.g_y, &size.g_w,
                  &size.g_h);
     form_dial (FMD_START, size.g_x, size.g_y, size.g_w, size.g_h,
                size.g_x, size.g_y, size.g_w, size.g_h);
     objc_draw (dial, ROOT, MAX_DEPTH, size.g_x, size.g_y,
                size.g_w,size.g_h);

     /* Durchführung */

     rc = form_do (dial, ROOT) & 0x7FFF;

     /* Nachbereitung */
      form_dial (FMD_FINISH, size.g_x, size.g_y, size.g_w, size.g_h,
                 size.g_x, size.g_y, size.g_w, size.g_h);
     wind_update (END_MCTRL);
     wind_update (END_UPDATE);
     dial[rc].ob_state &= (~SELECTED); /* Knopf deselektieren */

     return rc;
}
```

Die Funktion "do_dialog()" liefert die Nummer des Buttons zurück, mit dem der Dialog beendet wurde. Zu den Vorbereitungen gehört meistens das Setzen von Radioknöpfen, die Initialisierung der Editfelder und vieles andere. Der Nachbereitung ist somit die Auswertung veränderter Objekte und Editfelder zuzuordnen.

Eine Abwandlung des Durchführungsteils erlaubt etwas komplexere Verarbeitungen. Hier läßt sich zum einen der Doppelklick auswerten (das oberste Bit bei dem von "form_do()" zurückgegebenen Wert) und zum anderen ein "Spezial"-Objekt installieren. "form_do()"

kann so lange in einer Schleife wiederholt werden, bis ein bestimmter Button angeklickt worden ist. Bei Auswahl eines anderen Elements lassen sich Slider und sonstige interessante Dinge installieren. Der Durchführungsteil hätte dann die Form:

```
for (;;)
{
     rc = form_do (dial, ROOT);
     doppel_klick = rc & 0x8000;
     rc &= 0x7FFF;
     if ((rc == OK) || (rc == ABBRUCH))
          break;
     /* weitere, spezielle Auswertungen... */
}
```

Abfragen z. B. auf den SELECTED-Status oder auf den Inhalt eines Editfeldes erfolgen direkt über die Objektstruktur.

Frei definierte Objekttypen

Ein komplettes Listing! Das kleine Programm demonstriert die Verwendung des Objekttyps G_USERDEF. Statt des "normalen" Select-Status wird ein Objekt mit einem Kreuzchen versehen. Ursprünglich handelte es sich bei dem betreffenden Objekt um eines des Typs

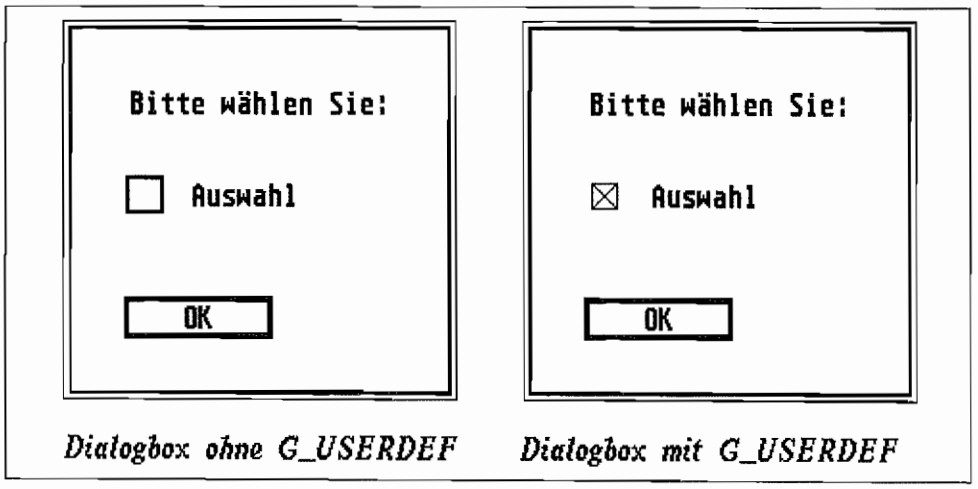


Abb. 6.4: Dialogbox mit USERDEF-Objekt

G_BOX, das jedoch lediglich dazu benutzt wurde, die Größe des selbst definierten Objekts festzulegen. Das Programm erkennt, daß ein Typ G_USERDEF zu setzen ist, an dem vom GEM ignorierten oberen Byte des Typs (der in unserem Falle auf 0x01 gesetzt wurde).

```
/* C-Programm, das ein Objekt vom Typ G_USERDEF benutzt. */

/* Includes für GEM */
#include <aes.h>
#include <vdi.h>
#include <portab.h>

/* Resourcedatei-Indizes */
#include "demo.h"

WORD apl_id,          /* Application-ID */
     v_handle;        /* VDI-Handle */
OBJECT  *dial;        /* Zeiger auf Objektbaum */
USERBLK user;         /* USERBLK für G_USERDEF-Objekt */

/* Zeichenketten */
char  no_resource[]="[3][|Resourcedatei|fehlt leider!][ OK ]",
      rsrc_name[]  ="DEMO.RSC";

/* Prototypen */
WORD cdecl selbutton (PARMBLK *parmblock);
void instal (OBJECT *dial, WORD obj);
void tree_walk (OBJECT *dial, WORD start, void (*callrout)
     (OBJECT *dial, WORD obj));
void do_dialog (OBJECT *dial);
void main (void);

/* Routine, die neuen Selectbutton ausgibt */
WORD cdecl selbutton (PARMBLK *parmblock)
{
     WORD xy[8], xy_clip[4];
     /* Clipping festlegen */
     xy_clip[0] = parmblock->pb_xc;
     xy_clip[1] = parmblock->pb_yc;
     xy_clip[2] = parmblock->pb_xc + parmblock->pb_wc-1;
     xy_clip[3] = parmblock->pb_yc + parmblock->pb_hc-1;
     vs_clip (v_handle, 1, xy_clip);
```

```
    /* Rechteck mit Rahmen ausgeben */
     xy[0] = parmblock->pb_x+1;
     xy[1] = parmblock->pb_y+1;
     xy[2] = parmblock->pb_x+parmblock->pb_w-2;
     xy[3] = parmblock->pb_y+parmblock->pb_h-2;
     vsf_interior (v_handle, FIS_HOLLOW);
     v_bar (v_handle, xy);

     /* Kreuz als Kennzeichen für SELECTED ausgeben */
     if (parmblock->pb_currstate & SELECTED)
     {
          xy[6] = xy[0];
          xy[5] = xy[1];
          xy[4] = xy[2];
          xy[7] = xy[3];
          v_pline (v_handle, 2, xy);
          v_pline (v_handle, 2, &xy[4]);
     }

     /* Clipping wieder aus */
     vs_clip(v_handle, 0, xy_clip);

     return (parmblock->pb_currstate & ~SELECTED);
}


/* installiert G_USERDEF-Objekte */
void instal (OBJECT *dial, WORD obj)
{
     switch ((dial[obj].ob_type & 0xFF00) >> 8)
     {
          case 0x01 :
               dial[obj].ob_type = G_USERDEF;
               dial[obj].ob_spec.userblk = &user;
               user.ub_code = selbutton;
     }
}

/* wandert durch den ganzen Objektbaum ab (exklusiv) start */
void tree_walk (OBJECT *dial, WORD start,
     void (*callrout)(OBJECT *dial, WORD obj))
```

```
{
     WORD i;

     for (i = dial[start].ob_head;  /* Kopf der Objektfolge */
         (i != start) && (i != -1); /* letztes zeigt auf Parent */
         i = dial[i].ob_next)       /* nächstes Objekt */
     {
          (*callrout)(dial, i);
          tree_walk (dial, i, callrout);
     }
}

/* do_dialog: siehe oben */

/* Hauptprogramm */
void main (void)
{
     WORD i;
     WORD work_in[12], work_out[57], void_word;

     /* Application anmelden */
     apl_id = appl_init();

     if (apl_id >= 0)
     {
          graf_mouse (HOURGLASS, 0L);

          /* Resourcedatei laden */
          if (rsrc_load(rsrc_name))
          {
               /* Adresse ermitteln */
               rsrc_gaddr (R_TREE, DEMODIAL, &dial);

               /* VDI-Workstation anmelden */
               v_handle = graf_handle (&void_word, &void_word,
                    &void_word, &void_word);
               for (i = 0; i < 10; work_in[i++] = 1);
               work_in[10] = 2;
               v_opnvwk (work_in, &v_handle, work_out);
```

```
                /* G_USERDEF setzen */
                tree_walk (dial, ROOT, instal);

                /* Mauszeiger als Pfeil */
                graf_mouse (ARROW, 0L);

                /* Dialog durchführen */
                do_dialog (dial);

                /* VDI-Workstation schliepen */
                v_clsvwk (v_handle);
           }
           else /* Fehlermeldung */
                form_alert (1, no_resource);

           appl_exit ();
      }
}
```

Hardwareunabhängigkeit

GEM-Programme unabhängig von der Hardware zu schreiben ist nicht schwer. Die Grundregeln lassen sich schnell auf einen einfachen Nenner bringen:

Aufrufe tieferer Betriebssystemschichten unterlassen und keine unsicheren Annahmen über irgendwelche Gegebenheiten machen.

Befehle und Bibliotheken der diversen Programmiersprachen (wie C) bieten einen großen Befehlsreichtum, womit Betriebssystemaufrufe in aller Regel unterbleiben können.

Wird jedoch einmal eine spezielle Funktion benötigt, so deklariert man diese in einem eigenen Modul. Bei einer Portierung des Programms auf ein anderes System ist dann nur dieser eine Modul zu erneuern, die restlichen Moduln bleiben unangetastet.

Um die Portabilität der verschiedenen C-Compiler untereinander zu erhöhen (manch ein Compiler hat "int" als 16-Bit-Wert deklariert, ein anderer als 32-Bit-Wert), findet eine Datei "portab.h" Verwendung, wie sie hier in aller Kürze aufgeführt ist.

(Die Datei "portab.h" von Digital Research bzw. Atari, wie sie z. B. bei Laser C mitgeliefert wird, enthält noch weitere Definitionen.)

```
/* PORTAB.H
   Include-Datei mit portablen Typendefinitionen für C. */

#ifndef __PORTAB
#define __PORTAB

#define BYTE    signed char     /* Byte (8 bits) mit Vorzeichen */
#define UBYTE   unsigned char   /* Byte ohne Vorzeichen */

#define         BOOLEAN int     /* 2 Werte: true/false */

#define WORD    signed int      /* Word (16 bits) mit Vorzeichen */
#define UWORD   unsigned int    /* Word ohne Vorzeichen */

#define LONG    signed long     /* Long (32 bits) mit Vorzeichen */
#define ULONG   unsigned long   /* Long ohne Vorzeichen */

#endif __PORTAB
```

Namensgebung von Betriebssystemfunktionen

Wer mit mehr als einem Entwicklungssystem arbeitet und hier und da mal GEM-Programme von einem System auf das andere portieren muß, kennt das Problem, daß die Namen der GEM-Routinen nicht einheitlich sind.

Bei C-Entwicklungssystemen ist es dabei noch recht einfach, denn die C-Namen der Funktionen und Konstanten sowie deren Aufbau sind von Digital Research vorgegeben.

Bei Pascal-Entwicklungssystemen läßt sich eine Namensübertragung sehr leicht realisieren, denn die C-Namen können übertragen werden. Schwieriger erscheint jedoch die Namensgebung auf Modula-2-Systemen, denn zum einen ist die Schreibweise der Bezeichner anders (Groß- und Kleinbuchstaben, in der Regel keine Unterstriche, längere Namen), und zum anderen existiert kein bindender Standard für GEM/Modula-2-Namen. Als einfachster Weg der Namensbildung auf Grundlage der C-Nomenklatur bietet sich folgender von R. Schneider (Digital Research) vorgeschlagene an:

- alle Funktionsnamen (und Silben) beginnen mit großen Buchstaben und werden pro Silbe mit kleinen Buchstaben fortgesetzt

- alle Unterstriche fallen weg.

Silben beziehen sich hierbei nicht auf sprachliche Silben, sondern auf die durch Unterstriche getrennten Bestandteile des C-Funktionsnamens. Somit ergeben sich beispielsweise die Namen

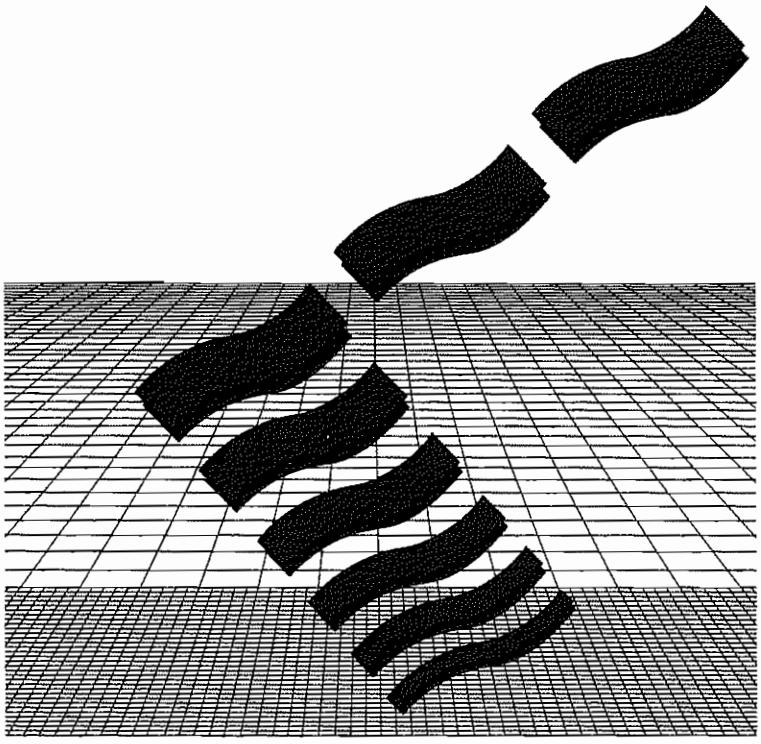
- VRbox aus v_rbox
- VFormAdv aus v_form_adv
- ShelRdef aus shel_rdef

Konstanten entstehen nach dem gleichen Prinzip, jedoch kann den Namen ein Buchstabe – beispielsweise K – vorangestellt werden. Beispiel:

- KGBoxtext aus G_BOXTEXT
- KShadowed aus SHADOWED

Der Sinn dieser Namenswahl ist, daß auch ein Modula-2-Programmierer sofort auch die C-Literatur und dieses Buch verwenden kann, ohne jedesmal nachdenken zu müssen, wie der Modula-2-Bezeichner denn nun heißen könnte. An die etwas kryptische Schreibweise gewöhnt man sich recht schnell.

Teil II



Die Hardware des ST

Einführung

Überblick über das System

Bei den Atari-Computern der ST-Serie handelt es sich, wie schon bei den 8-Bit-Modellen der XL- und XE-Serie, um sogenannte Tastaturcomputer.

Was andeutet, daß sich fast alle zum Betrieb benötigten Hardwarekomponenten zusammen mit der Tastatur in einem Gehäuse befinden. Ein solcher Tastaturcomputer ist bereits nach Anschluß der Stromversorgung und eines Sichtgerätes (Monitor) betriebsfähig. Sinnvolles Arbeiten ist jedoch erst mit Anschluß eines Massenspeichers (Floppy-Disk/Harddisk) möglich, weshalb bei den Modellen 520ST(FM)/1040ST(FM) bereits ein Floppy-Disk-Laufwerk mit einer Speicherkapazität von 720 KByte ins Gehäuse eingebaut wird.

Die Geräte der MEGA ST-Serie fallen schon nicht mehr in diese Kategorie, da sie eine abgesetzte Tastatur besitzen. Die Intelligenz befindet sich einschließlich Stromversorgung in einem separaten Gehäuse, welches auch eine 720 KByte-Floppy beinhaltet und als Monitoruntersatz benutzt werden kann.

Im Prinzip stellen sich jedoch die Computer der ST-Serie als ein System mit den Grundkomponenten Zentraleinheit, Grafiksystem, Musik- und Tonsystem und verschiedenen peripheren Untersystemen dar. Die Zentraleinheit besteht aus:

- einem mit 8 MHz getakteten MC68000 Mikroprozessor
- 192 KByte Festwertspeicher (ROM), durch einsteckbares Cartridge auf 320 KByte erweiterbar
- 512 KByte Arbeitsspeicher (RAM) bei 260ST/520ST(FM) bzw. 1 MByte bei 520ST+ und 1040 ST(FM) (1, 2 oder 4 MByte bei der MEGA ST-Serie).

Das Grafiksystem in Stichworten:

- 32 KByte Bildschirmspeicher (beliebig im Arbeitsspeicher gelegen)
- folgende Anzeigemodi sind möglich:

  320 x 200 Bildpunkte in 16 aus 512 möglichen Farben

640 x 200 Bildpunkte in vier aus 512 möglichen Farben
640 x 400 Bildpunkte in S/W mit Monochrom-Monitor

- Anschluß eines hochauflösenden Monochrom- oder eines Farbmonitors mit RGB-Eingang (beim 520ST(F)M/1040ST(F)M ist der Anschluß eines Fernsehers durch eingebauten HF-Modulator möglich).

Das Musik- und Tonsystem enthält einen Soundchip mit drei programmierbaren Tonkanälen und einem Rauschgenerator. Die Hüllkurve ist programmierbar.

Die peripheren Untersysteme im ST bestehen aus:

- Tastatur-, Maus- und Joystick-Interface mit eigenem Mikroprozessor und über eine serielle Schnittstelle mit dem Hauptsystem verbunden
- Parallel-Schnittstelle (in der Regel als Druckerinterface nach Centronics-Standard genutzt)
- serielle Schnittstelle (RS232) für Modem oder seriellen Drucker
- Floppy-Disk-Interface für zwei Laufwerke bis je 720 KByte (formatierter) Speicherkapazität
- ATARI Computer Sytem Interface (ACSI) mit DMA-Unterstützung zum Anschluß von schnellen Peripheriegeräten wie Harddisk, CD-ROM, Laserdrucker usw.
- MIDI-Interface für Anschluß an digital steuerbare Musikinstrumente.

Immer mehr Grips in immer weniger Chips

Im ST befinden sich neben Standardbausteinen wie der 68000er-CPU, WD1772-Floppy-Disk-Controller, AY-3-8910- oder YM2149-Soundchip, 68901-Multifunktionsbaustein, 6850-ACIAs (Bausteine für asynchrone, serielle Datenübertragung), RAMs und ROMs noch vier sogenannte Custom-Chips (beim STE wurde die Anzahl auf drei Custom-Chips reduziert, da GLUE und MMU zur GSTMCU zusammengefaßt wurden). Diese speziell von Atari für die ST-Computer entwickelten Chips enthalten eine Vielzahl von Funktionsgruppen, die für das einwandfreie Funktionieren des Systems erforderlich sind. Ihnen ist es zu einem großen Teil auch zu verdanken, daß ein Computer dieser Leistungfähigkeit zu einem so niedrigen Preis zu haben ist.

Der *SHIFTER* sorgt dafür, daß Sie ein klares Bild von Ihrem ST erhalten. Er setzt nämlich die Information des Hauptspeicherbereichs, der für den Bildschirmspeicher reserviert ist, in ein Videosignal für den Monitor um.

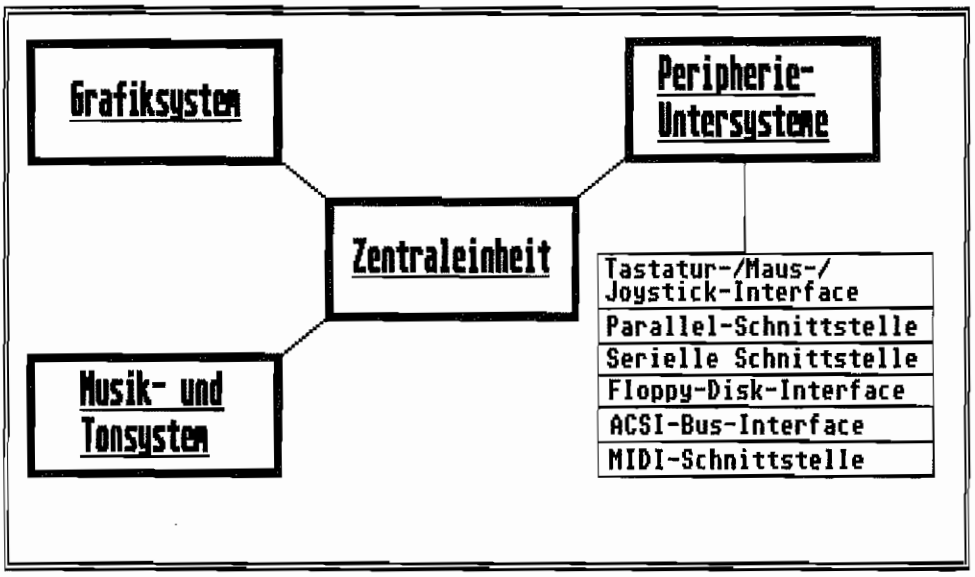


Abb. 0.1: Die einzelnen Komponenten eines ST

Dazu verfügt er zwar über einen 16 Bit-Datenbusanschluß, der Adreßzähler für die entsprechende darzustellende Bildschirmspeicher Adresse befindet sich jedoch in der MMU und wird dort verwaltet. Über entsprechende Registerauswahl- und Steuerleitungen lädt die MMU unter Mithilfe des GLUE (Steuersignalerzeugung) die SHIFTER-Register mit den Werten aus dem Bildschirmspeicher und für die Farbpalette. Mit dem 32 MHz-Mastertakt wird die Bildinformation dann als Color- oder Monochromsignal ausgegeben.

Der *BLITTER-Chip* ist in den MEGA STs/STEs eingebaut und realisiert die Line-A-Funktion "Bit-Block-Transfer" mit Hardware-Unterstützung. Mit seiner Hilfe lassen sich ohne Umweg über die CPU blitzschnell große Datenbereiche im Bitraster manipulieren.

Der BLITTER darf dazu ebenfalls wie die DMA-Einheit auf den Hauptspeicher direkt zugreifen. Zugriffe müssen jedoch mit der CPU und der DMA-Einheit abgestimmt werden, wobei der DMA-Betrieb des BLITTERs niedrigste Priorität besitzt. Im Gegensatz zum SHIFTER ist der BLITTER ein eigener Co-Prozessor-Chip, der über einen vollwertigen Adreß- und Datenbus verfügt. Außerdem besitzt er alle nötigen Steueranschlüsse für die Steuerung des Daten- und Adreßbusses als eigenständige Einheit in einem 68000er-Prozessorsystem.

Die *DMA-Einheit* hilft den Peripheriegeräten wie Floppy-Controller und den am ACSI-Bus angeschlossenen schnellen Peripheriegeräten wie Harddisk, CD-ROM und evtl. Laserdrucker

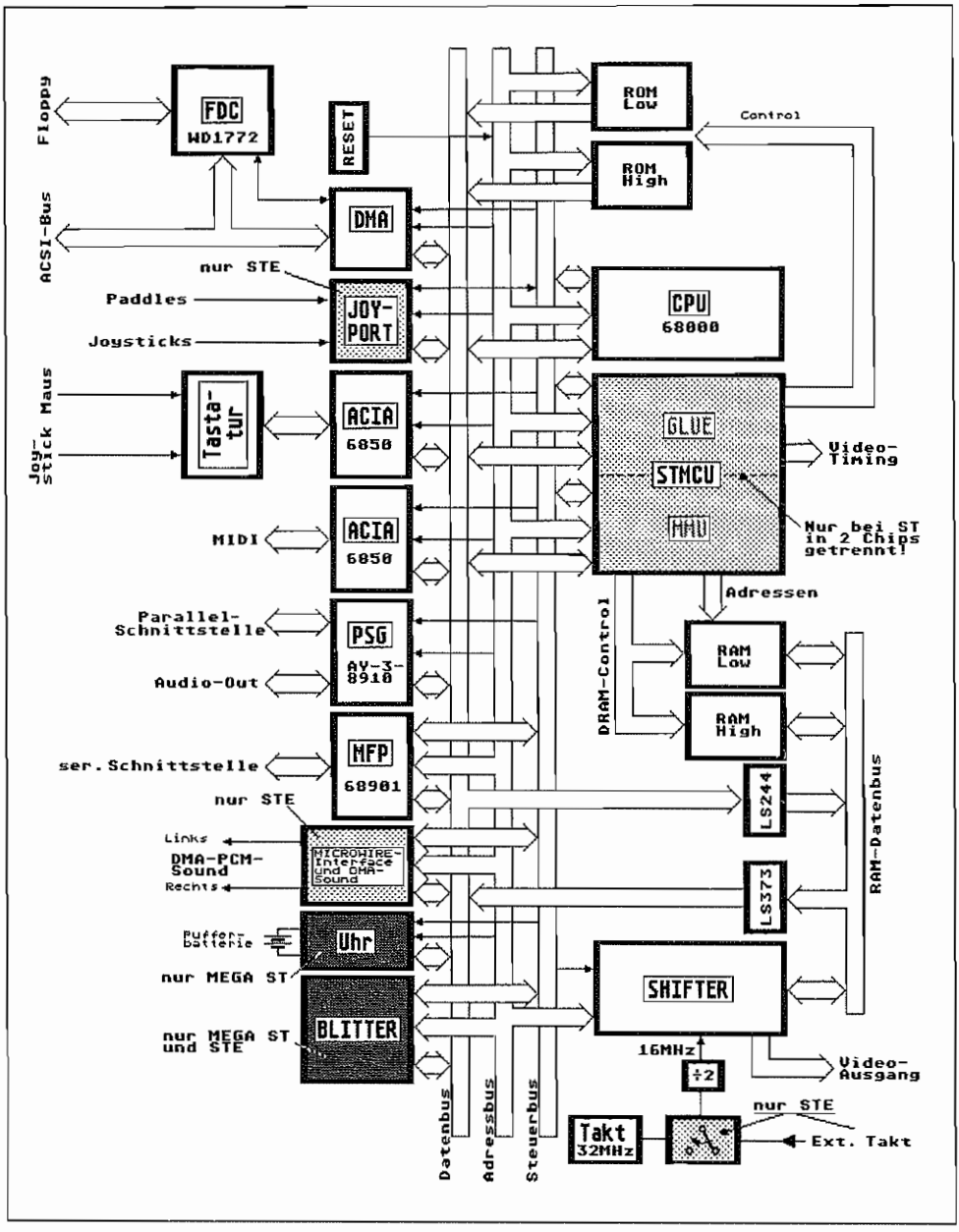


Abb. 0.2: Das Blockschaltbild des ST

beim Datenaustausch. Sie kann dabei, genau wie die CPU direkt auf den Hauptspeicher zugreifen (deshalb DMA = Direct Memory Access) und die erforderlichen Daten für die Peripheriegeräte dort abholen oder deponieren.

Die CPU hat mit dem Datentransfer also nicht mehr viel zu tun und könnte sich anderen Aufgaben widmen (tut sie aber nicht!). Dem DMA-*Chip* ist es ähnlich gegangen wie dem SHIFTER. Die entsprechenden Adreßzähler für die Abwicklung von DMA-Operationen befinden sich in der MMU. Lediglich ein 16 Bit-Anschluß zum Datenbus ist vorhanden, über den der DMA-Chip den DMA-Transfer abwickelt. Die Steuerung des Adreßbusses übernimmt der GLUE in Zusammenarbeit mit der MMU!

Die Memory Management Unit (*MMU*) besorgt die Umsetzung der von der CPU ausgehenden Adreßinformation in einen gemultiplexten Adreßbus für die dynamischen RAMs. Außerdem werden von ihr die Refresh-Signale für die RAMs zur Verfügung gestellt, die dafür sorgen, daß die Speicherzellen in den RAM-Chips nicht plötzlich an "Gedächtnisschwund" leiden und ihren Informationsinhalt verlieren. Die MMU ist dazu vollständig an den Adreßbus angebunden mit den zugehörigen Handshake-Leitungen für den Memory-Zugriff.

Auf den Datenbus greift die MMU direkt nur zur Hälfte (D0..D7) zu. Diese Hälfte wird für das Laden der in diesem Chip integrierten Register wie für Memory-Configuration, die Video-Base- und Video-Counter-Register sowie die DMA-Base-Register verwendet.

Der *GLUE* übernimmt die Decodierung der Adreßbereiche für RAM, Betriebssystem- und Cartridge-ROM und die peripheren Chips wie Soundchip, ACIAs und 68901. Weiterhin werden die für die CPU erforderlichen Steuer- und Quittungssignale für Interrupts, Zugriffsberechtigung auf geschützte Speicherbereiche usw. vom GLUE erzeugt.

Der GLUE-Chip hat dazu einen vollwertigen Adreßbus-Anschluß und Zugriff auf alle CPU-Steuersignale. Dafür ist jedoch sein Datenbus-Anschluß entsprechend mager ausgefallen (gerade mal D0 und D1 sind angebunden – also genausoviel wie der GLUE braucht, um das interne Sync-Mode-Register zu bedienen!).

Wenn also bei den weiteren Erläuterungen von *dem* Video-Controller oder *der* DMA-Einheit die Rede ist, sind damit alle Register und Logikkomponenten gemeint, die für die angesprochene Funktionseinheit erforderlich sind, egal in welchem Custom-Chip sie zu finden sind.

Kapitel 1: Die Zentraleinheit

Die Zentraleinheit der Atari ST Computer besteht aus dem Mikroprozessor, dem Hauptspeicher und der DMA-Einheit.

Der Mikroprozessor

Das Herz des ATARI ST ist eine MC68000-CPU, die mit einer Frequenz von 8 MHz getaktet wird. Hierbei handelt es sich um eine 32-Bit-CPU, die intern eine Verarbeitung von Daten mit 32 Bit Breite vornimmt. Der Datentransfer von und zur CPU erfolgt jedoch über einen 16 Bit breiten Datenbus (Daher auch die Bezeichnung "ST" = Sixteen Thirtytwo = 16 Bit ext. Datenbus/32 Bit int. Datenbus).

Es können also gleichzeitig zwei Bytes (ein Datenwort) übertragen werden, wobei aber auch Einzelbyte-Zugriffe möglich sind. Ein Befehl, um z. B. ein Datenwort aus einer Speicherstelle in eines der CPU-Register zu laden, benötigt im ST vier Taktzyklen. Bei einer Taktfrequenz von 8 MHz dauert das Ganze also nur 0,5 Mikrosekunden!

Die CPU besitzt acht Daten- und neun Adreßregister mit je 32 Bit Breite, wobei zwei der neun Adreßregister Stapelzeiger (Stackpointer) darstellen. Zwei Stapelzeiger deshalb, weil die CPU zwei Betriebsarten kennt, den Supervisor-Modus und den User-Modus. Je nach Betriebsart "sieht" das laufende Programm den Supervisor- oder den User-Stapelzeiger.

Im Supervisor- oder System-Modus ist vom Programm her der Zugriff auf alle Speicherbereiche und das gesamte Statusregister der CPU möglich.

Im User- oder Benutzer-Modus kann nicht auf Daten des Supervisorspeicherbereichs zugegriffen werden, so daß sich eine erhöhte Sicherheit gegen versehentliches Manipulieren von für das Betriebssystem "lebenswichtigen" Daten (Systemvariablen), die im Supervisorspeicherbereich untergebracht sind, durch ein User-Programm ergibt.

Der Programmzähler der CPU ist zwar 32 Bit breit, herausgeführt auf den Adreßbus sind jedoch nur die Adreßleitungen A1 .. A23. Die Adreßleitung A0 existiert beim MC68000 nicht, da ja mit einem 16-Bit-Datenbus gearbeitet wird. Um aber auch auf ungerade Adressen (einzelne Bytes) zugreifen zu können, liefert die CPU die low-aktiven Steuersignale UDS und LDS (Upper-Data-Strobe und Lower-Data-Strobe).

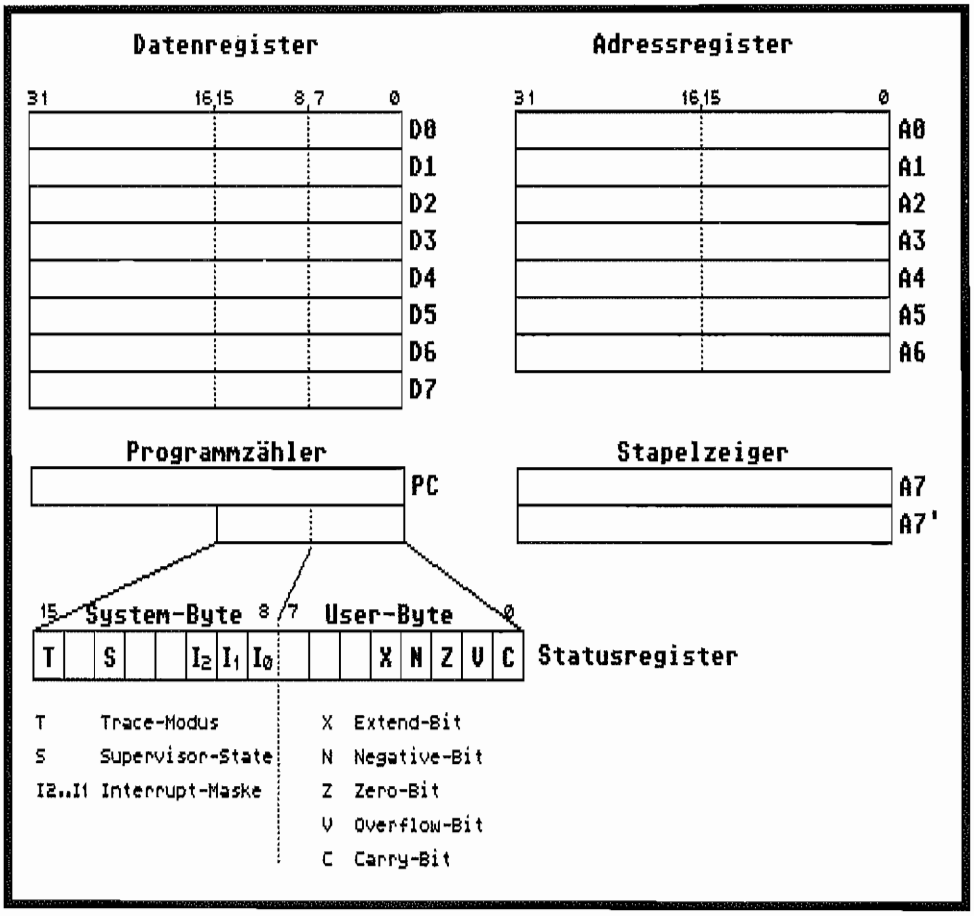


Abb. 1.1: Die Register des MC68000

UDS ist aktiv (Low), wenn auf das höherwertige Byte eines Wortes zugegriffen wird (Bits 8..15 des Datenbusses, gerade Speicheradresse), und LDS ist entsprechend aktiv (Low) bei einem Zugriff auf das niederwertige Byte (Bits 0..7 des Datenbusses, ungerade Speicheradresse). Logischerweise sind also dann bei einem Wortzugriff beide Leitungen aktiv!

Mit den Adreßleitungen A1..A23 lassen sich also "nur" max. 16 MBytes an Speicherraum direkt ansprechen, diese 16 MByte jedoch durchgehend linear ohne Segmentierung wie z. B. bei den 8086/8088-CPUs. Dieser Adreßraum von 16 MByte wird im ATARI ST, wie in Abbildung 1.2 gezeigt, eingeteilt.

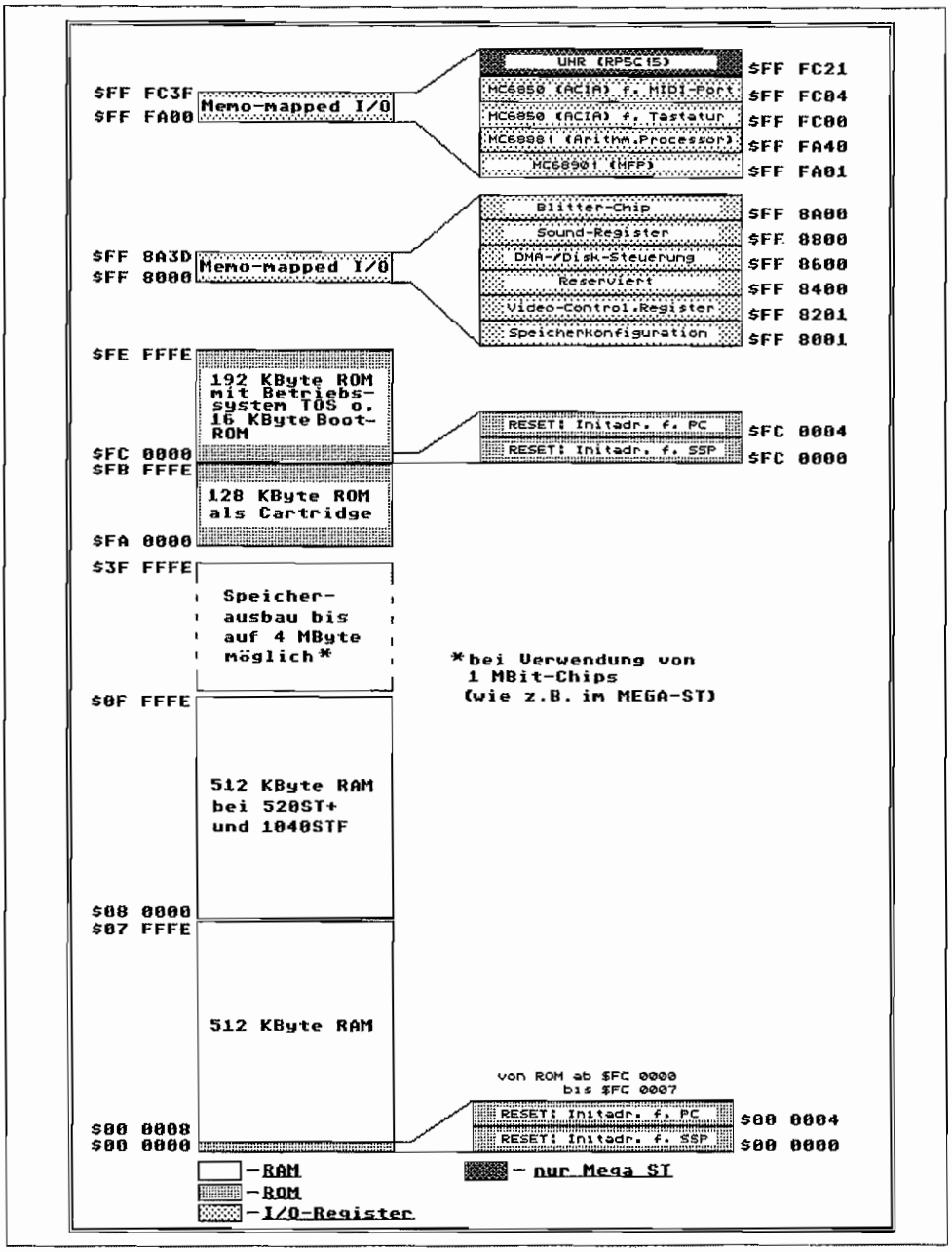


Abb. 1.2: Die Speicheraufteilung des ST im Überblick

RAM und ROM

Die ersten zwei KByte des Adreßraums des ST sind für die Ausnahmevektoren und den Supervisor-Stack reserviert und liegen (fast ausschließlich) im RAM. Dabei belegen die im ST benutzten Ausnahmevektoren die ersten zwei Pages des Speicherraums (von $8 bis $13C). Ab Adresse $380 schließt sich daran der Bereich mit den verschiedenen Systemvariablen an (siehe Teil 1, Kapitel 1, "BIOS und XBIOS").

Bei den ersten acht Bytes ab Adresse $0 handelt es sich nicht um RAM, sondern um ROM! Es sind dies die ersten acht Bytes des ST-ROMs ab Adresse $FC 0000, die durch entsprechende Adreßdecodierung an diese Stelle des Adreßraums gedoppelt werden. Der Inhalt dieser beiden Langwörter sorgt dafür, daß nach dem Einschalten des ST oder nach einem RESET das gesamte Computersystem von einem definierten Ausgangspunkt hochgefahren werden kann.

Die 68000er-CPU erwartet nach einem RESET nämlich in Adresse $0 einen Anfangswert für den Supervisor-Stackpointer und in Adresse $4 einen Pointer auf die Adresse des ersten auszuführenden Befehls!

Wer jetzt mal schnell bei Adresse $0 nachgeschaut hat, wundert sich bestimmt über diesen seltsamen Wert, der dort als Startwert für den Supervisor-Stackpointer aufbewahrt wird. Keine Bange, der Wert stimmt schon, nur enthält das Langwort (eigentlich ja bei $FC 0000 im ROM gelegen) einen Sprungbefehl auf die Systemstart-Routinen im ROM. Das ist aber nicht weiter tragisch, da der ST nach einem RESET zunächst sowieso nicht weiß, welche Speicherbestückung für das RAM vorliegt. Zuerst muß nämlich mit Hilfe der MMU und den eingebauten RAM-Chips ein durchgehender Speicherbereich "zusammengebaut" werden. Dazu später noch ein wenig mehr.

Die Geräte 260ST und 520ST(FM) besitzen in ihrer Grundausstattung eine Speicherkapazität von 512 KByte RAM. Somit endet für diese Geräte der RAM-Bereich bei $07 FFFF. Diese 512 KByte werden mittels 16 Dynamischer RAM-Chips des Typs 41256 realisiert, welche pro Chip eine Speicherkapazität von 256K x 1 Bit aufweisen.

Die Datenleitungen dieser 16 Chips werden zu einem 16-Bit-Datenbus zusammengefaßt, denn die CPU kann ja 16 Bit auf einmal verarbeiten. Für die Adressierung der 262144 Speicherstellen in den Speicherchips wird ein gemultiplexter Adreßbus verwandt. Man legt dazu zeitlich nacheinander erst die untere und dann die obere Hälfte der in diesem Fall benötigten Adreßinformation an die Speicherchips.

Deshalb genügen dann für einen 256 KBit-Chip auch statt der sonst erforderlichen 18 (2^18 = 262144) nun 18/2 = 9 Adreßleitungen und je eine sogenannte RAS- und CAS-Leitung. Die RAS- und CAS-Leitung (RAS = Row Adress Strobe = Zeilen-Adreßimpuls / CAS = Column

Adress Strobe = Spalten-Adreßimpuls) signalisieren der Logik in den Speicherchips, welcher Teil der benötigten Adresse gerade an den Adreßeingängen des Chips anliegt.

Man spart so Platz, Anschlüsse und Leitungen, benötigt aber etwas zusätzliche Logik für das Multiplexen und das Timing. Im ST befindet sich diese Logik in der sogenannten Memory-Management-Unit (MMU).

Standardchips erleichtern den Speicherausbau

Bereits bei der Entwicklung der ST-Serie wurde von ATARI Wert auf eine möglichst einfache Verwendbarkeit von Standard-RAM-Chips gelegt.

So unterstützt die derzeit in den ST verwendete MMU zwei RAM-Speicherbänke mit dynamischem RAM zu max. 2 MByte pro Bank. Die STs des Typs 260ST und 520ST(FM) besitzen nur eine Speicherbank mit 16 Chips des 256KBit-Typs und kommen so auf 512 KByte RAM.

Um nun bei den Geräten des Typs 520ST+ und 1040ST(FM) auf einen Speicherausbau von 1 MByte RAM zu kommen, wurde einfach die zweite Speicherbank mit weiteren 16 Chips des 256 KBit-Typs eingebaut. Diese Aufrüstung ist relativ problemlos, denn außer den Chips braucht man keine zusätzlichen Multiplexer und Adreßdecoder. Die MMU liefert bereits von Haus aus die hierfür zusätzlich erforderlichen RAS- und CAS-Signale für die zweite Speicherbank.

Für die zusätzlich erforderlichen 16 Chips der zweiten Bank ist das Platinenlayout des 1040ST(FM) bereits vorgesehen. Beim 520ST+ (oder nachträglich beim 260ST oder 520ST(FM)) lassen sich die Chips für die zweite Speicherbank einfach auf die Chips der ersten Bank "huckepack" auflöten. Lediglich die RAS- und CAS-Anschlüsse werden nicht direkt mit den darunterliegenden Chipanschlüssen verlötet, sondern separat mit der MMU verbunden.

Dabei werden alle RAS-Anschlüsse (Pin 4 der Speicherchips) der neuen Bank verbunden und an den Anschluß RAS1 (Pin 18) der MMU geführt. Die CAS-Anschlüsse (Pin 15 der Speicherchips) der acht Chips, die mit den Datenleitungen D0...D7 des Datenbus (niedrigwertiges Byte) verbunden sind, werden gebrückt und über einen 68 Ohm-Widerstand an CAS1L (Pin 21) der MMU gelegt.

Entsprechend ist mit den CAS-Anschlüssen der acht Chips zu verfahren, die an die Datenleitungen D8...D15 (höherwertiges Byte) des Datenbusses angeschlossen sind. Sie werden ebenfalls gebrückt und auch über einen 68 Ohm-Widerstand mit CAS1H (Pin 22) der MMU verbunden. Für den Aufbau der Speicherbänke können verschiedene Typen von dynamischen RAM-Chips verwendet werden, wie Abbildung 1.3 zeigt.

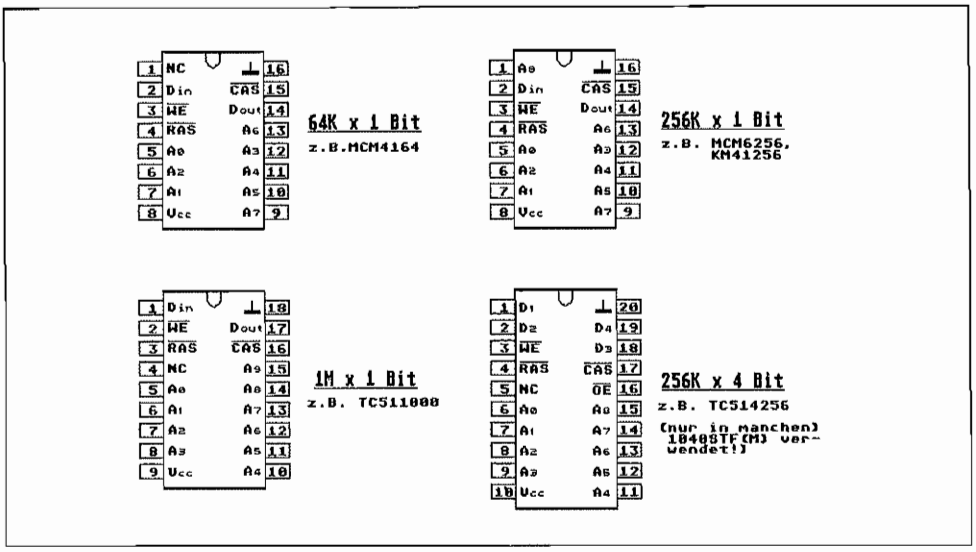


Abb. 1.3: Die im ST verwendbaren RAM-Chips

1 MBit-Chips passen leider nicht!

Die 64 KBit- und 256 KBit-Chips sind untereinander pinkompatibel und können direkt ausgetauscht werden. Ein direkter Tausch der 256 KBit- gegen 1 MBit-Typen ist aber leider nicht möglich (unterschiedliches Pin-Layout!).

Will man einen "Nicht-MEGA ST" mit 1 MBit-Chips bestücken, so sind einige Modifikationen bei den Chipanschlüssen erforderlich, die einen versierten Hardware-Bastler jedoch nicht vor unüberwindliche Probleme stellen dürften! Auch werden inzwischen von verschiedenen Firmen Aufrüstplatinen angeboten, die mit oder ohne Entfernen der alten Speicherbank eingebaut werden können und wahlweise eine Bestückung mit 256 KBit- oder mit Megabit-Chips erlauben. Aufrüstungen auf bis zu 4 MByte sind auch für "Nicht-MEGA ST-Computer" möglich. Grenzen sind nur durch den eigenen Geldbeutel gesetzt. Die Geräte der MEGA ST-Serie besitzen von Haus aus die Megabit-Chips und kommen somit auf einen Speicherausbau von 2 MByte bei einer Bank bzw. 4 MByte, wenn auch die zweite Bank auf der Hauptplatine bestückt ist.

Wegen der 1988/89 teilweise auftretenden Speicherknappheit hat ATARI insbesondere bei den Rechnern der 1040er-Serie verschiedene Platinenlayouts verwendet, um eben alle nur möglichen Speicherchips verwenden zu können. So kamen unter anderem auch 1 MBit-

RAMs der Organisation 256K x 4 Bit zum Einsatz. Die Bestückung der damit ausgerüsteten 1040er sieht dann so aus, daß pro Speicherbank vier Chips zum Einsatz kommen und damit 512 KByte Kapazität aufweisen.

Um auf die Standardkapazität von 1 MByte pro Rechner zu kommen, wurde die zweite Bank genauso bestückt. Acht Chips ergeben somit eine Kapazität von 1 MByte! Eine Erweiterung auf 2 MByte oder 4 Mbyte ist bei diesen Rechnern aber nicht mehr im einfachen "Huckepack-Verfahren" möglich.

Bei einer evtl. Speichererweiterung ist darauf zu achten, daß jede Bank einheitlich mit den gleichen Chips bestückt werden muß.

Es ist aber ohne weiteres möglich, in den beiden möglichen Speicherbänken unterschiedliche Chiptypen zu verwenden. So kann man z. B. die erste Bank mit 256 KBit-Chips bestücken und kommt dann auf 512 KByte RAM. Die zweite Bank kann mit Megabit-Chips bestückt sein und weist dann eine Kapazität von 2 MByte auf.

Zusammen sind das dann stattliche 2,5 MByte! (Leider gibt es einige STs mit MMUs der Firma IMP, die eine ungleiche Bestückung der beiden Speicherbänke nicht erlauben!) Durch diese Möglichkeit der gemischten Bestückung ergeben sich eine Reihe von möglichen Speicherkonfigurationen.

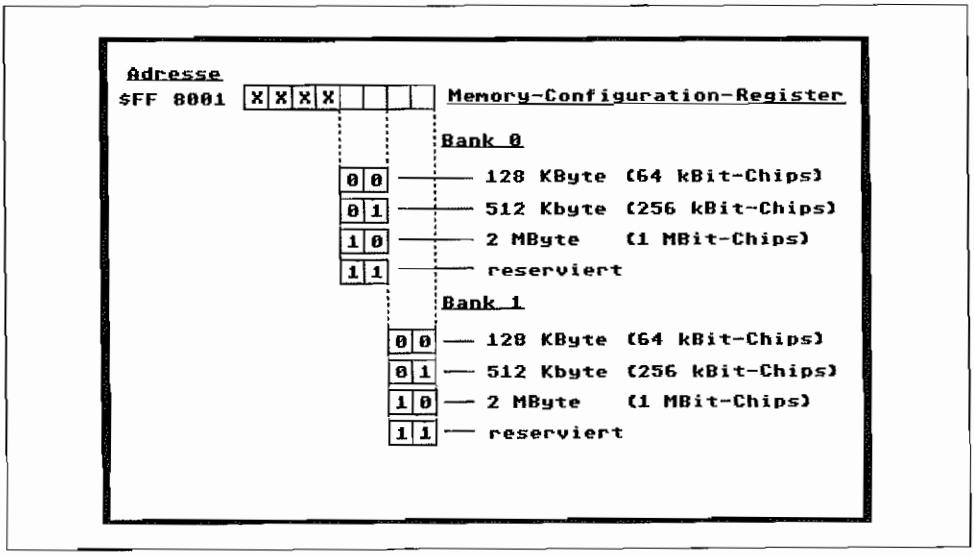


Abb. 1.4: So programmiert der ST seinen Speicherausbau

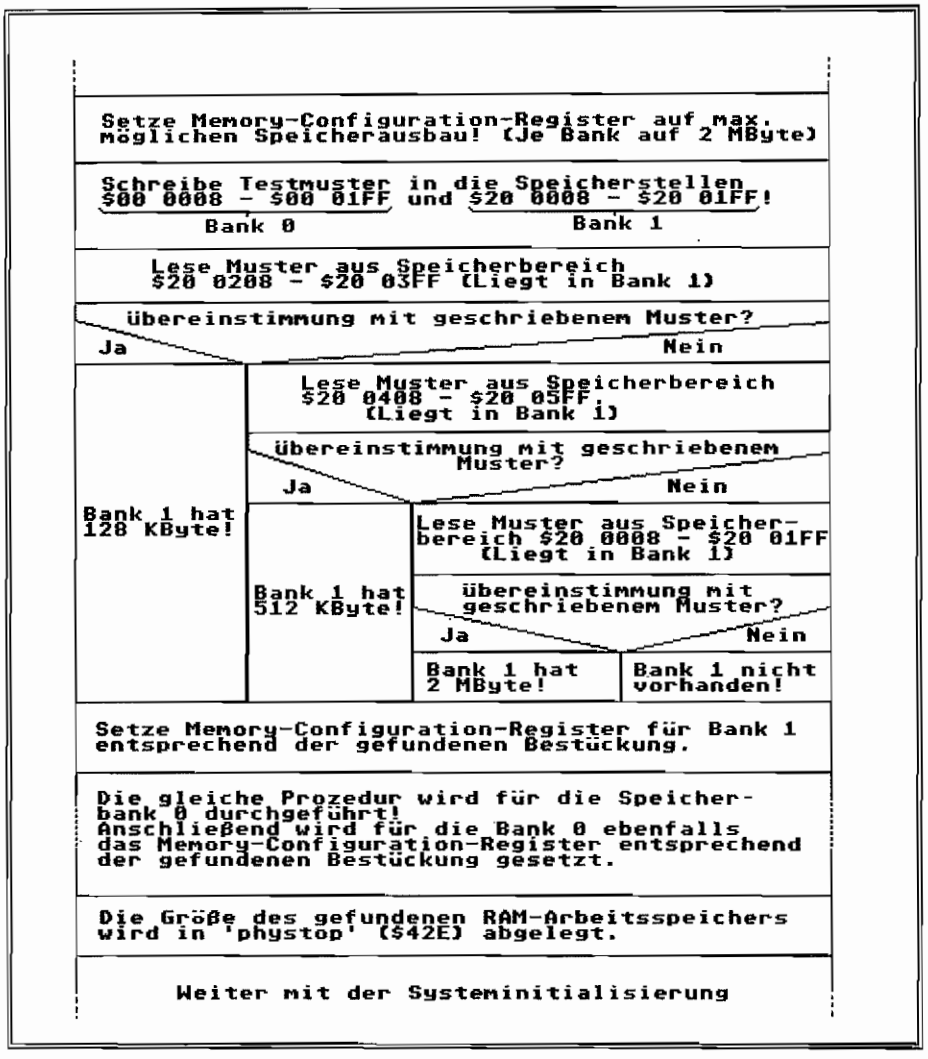


Abb. 1.5: So stellt der ST beim Systemstart seinen Speicherausbau fest

Wieviel RAM ist denn drin?

Um festzustellen, welcher Speicherausbau denn nun vorliegt, wird zu Beginn der Systeminitialisierung ein Speichertest ausgeführt, der Aufschluß darüber gibt, welche Speicherchips

vorhanden sind. Der Programmteil liegt im ROM und wird nahezu unmittelbar nach dem Systemstart ausgeführt.

Hierbei wird folgender Effekt ausgenutzt:

Man geht zunächst beim Speichertest davon aus, daß beide Speicherbänke vorhanden sind und mit Megabit-Chips bestückt sind.

Die ersten zwei Pages ($0...$1ff) jeder Bank werden nun mit einem Testmuster beschrieben. Befinden sich jedoch statt der Megabit-RAMs, die ja unterstellt werden, solche des 64 KBit- oder 256 KBit-Typs im Rechner, so werden durch eine Art Falschdecodierung der Adreßleitungen zu den Speicherchips nicht nur die ersten beiden Pages der Speicherbank mit dem Testmuster beschrieben, sondern je nach verwendeten Chips noch andere Pages!

Aus der Information, welche Pages noch beschrieben wurden, kann auf die Art der eingebauten Speicherchips geschlossen werden.

Das sogenannte Memory-Configuration-Register (Label: "memconf") bei Adresse $FF 8001 wird dann entsprechend gesetzt, so daß auch bei unterschiedlicher Bestückung der beiden Bänke ein durchgehender RAM-Speicherbereich vorliegt. Das "Schattenregister" des "memconf"-Registers befindet sich, für den Programmierer besser zugänglich, bei den Systemvariablen als Bytewert (jedoch nur unteres Nibble interessant!) an Speicherstelle $424 (Label: "memcntrl"). Soviel zum RAM des ST!

Nichtflüchtige Speicherbereiche

Wie bereits aus dem Bild zur Speicheraufteilung des ST hervorgeht, liegen im Adreßraum der CPU zwei Speicherbereiche, die für ROM-Bestückung vorgesehen sind. Als erstes soll der Bereich ab Adresse $FC 0000 betrachtet werden. Hier ist ein Speicherbereich von 192 KByte für das Betriebssystem (TOS) des ST vorgesehen. Inzwischen werden alle ST-Systeme mit Betriebssystem-ROMs ausgeliefert.

Die 260ST- und 520ST+ -Rechner enthielten nur zwei 8Kx8 Bit "Boot-ROMs" in zwei der insgesamt sechs vorgesehenen Fassungen für die Betriebssystem-ROMs. Damit konnte das eigentliche TOS von einer Systemdiskette in den RAM-Arbeitsspeicher geladen (gebootet) und gestartet werden. Auf diese Weise gehen aber immerhin 192 KByte vom RAM für das Betriebssystem drauf! Die Boot-ROMs belegen den Adreßraum von $FC 0000 bis $FC 3FFF.

In diesen 16 KByte ROM ist ein Programmabschnitt für die Grundsysteminitialisierung enthalten sowie Daten für die Bildinformation, die vom angeschlossenen Monitor wiederge-

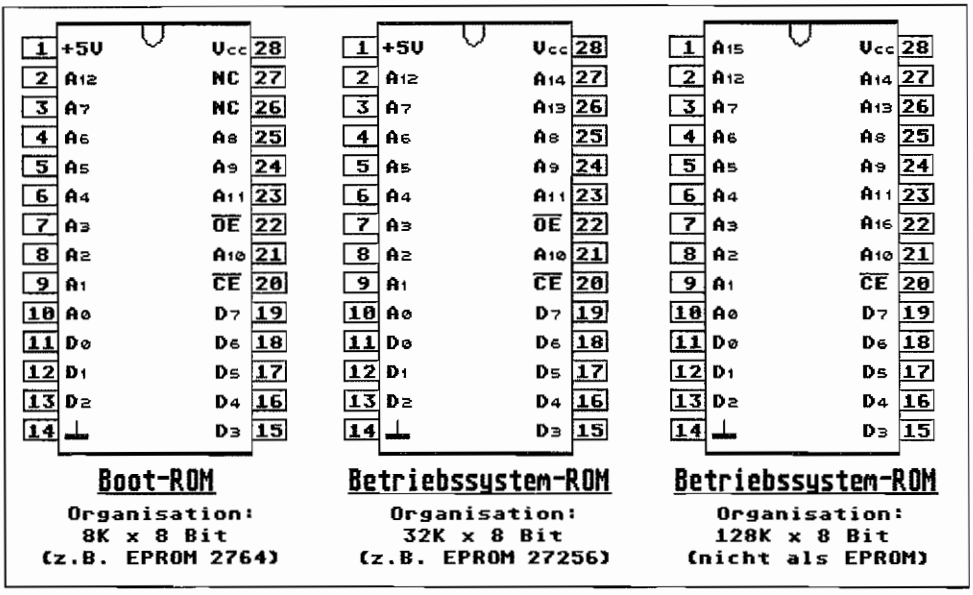


Abb. 1.6: Pinbelegung der im ST verwendbaren ROMs

geben wird, während das eigentliche TOS von der Systemdiskette in den Arbeitsspeicher geladen wird.

Geballte "Intelligenz" sofort verfügbar – TOS im ROM

Bei den von ATARI ab Ende 1986 ausgelieferten Rechnern der ST-Serie ist das komplette Betriebssystem in sechs ROM-Chips der Organisation 32K x 8 Bit oder in zwei ROM-Chips der Organisation 128K x 8 Bit enthalten. Die ROM-Chips sind von ATARI auch als Set separat erhältlich, so daß auch ältere ST-Rechner mit ROM-TOS nachgerüstet werden können, die Steckplätze sind ja vorhanden.

Mit Erscheinen der MEGA ST-Serie wurde erstmals eine neue Version von ROM-TOS (Version 1.02) ausgeliefert, welche den BLITTER (deshalb auch als BLITTER-TOS bezeichnet) und den neuen Hardware-Uhr-Chip unterstützt. Eine weitere, überarbeitete TOS-Version (1.04) steht seit Mitte März 1989 auf ROMs zur Verfügung.

In neueren ST-Systemen (sowohl MEGA ST als auch 1040ST(FM) können durch geänderte Platinen-Layouts erstmals auch Megabit-ROM-Chips verwendet werden.

Das bedeutet, daß für das TOS nur noch zwei Chips mit je 128K x 8 Bit Kapazität erforderlich sind und diese noch nicht einmal voll ausgenutzt werden! In Abbildung 1.7 ist der Schaltungsauszug wiedergegeben, der zeigt, wie die Einbindung der Betriebssystem-ROMs beim MEGA ST und bei Geräten der 1040er-Serie vorgenommen wurde.

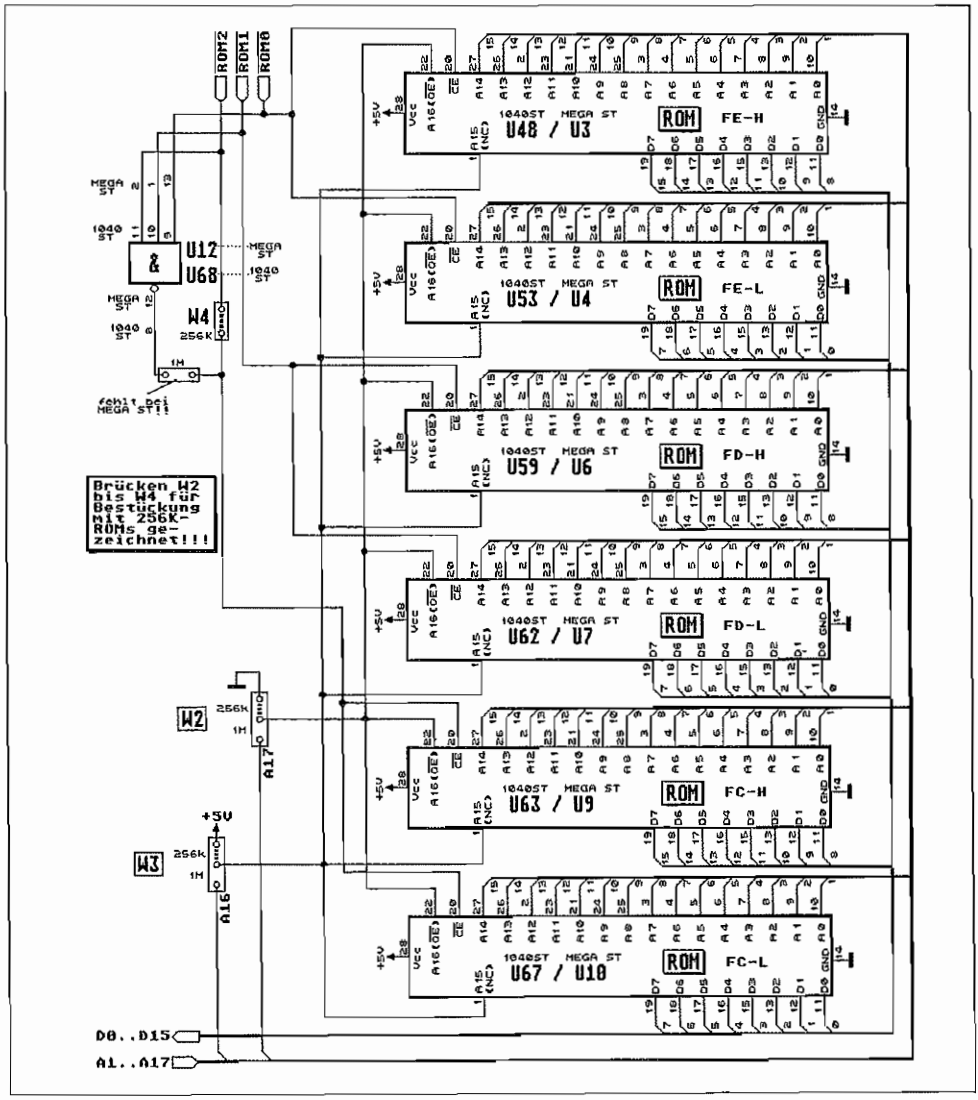


Abb. 1.7: Schaltung der ROM-Bank für MEGA ST und 1040STF(M)

Da je ROM-Chip ein Byte pro Adresse angesprochen wird, bilden immer zwei Chips ein Pärchen, um den 16-Bit-Datenbus mit ROM-Informationen zu versorgen. Wenn 256 KBit-ROMs (oder EPROMs) verwendet werden, sind alle sechs Chipfassungen belegt. Dabei ist die Aufteilung des Adreßraums entsprechend der nachfolgenden Tabelle vorgenommen:

Adreßbereich	Chip-# MEGA ST	Chip-# 1040ST	Chip aktiviert bei Low auf	Inhalt
$FC 0000 bis	U 10	U 67	_ROM2	Low-Byte
$FC FFFF	U 9	U 63	_ROM2	High-Byte
$FD 0000 bis	U 7	U 62	_ROM1	Low-Byte
$FD FFFF	U 6	U 59	_ROM1	High-Byte
$FE 0000 bis	U 4	U 53	_ROM0	Low-Byte
$FE FFFF	U 3	U 48	_ROM0	High-Byte

Die Chip Enable(_CE)-Information auf den Leitungen _ROM0.._ROM2 werden dabei vom GLUE-Chip geliefert. Je nach "gewünschtem" Adreßbereich wird somit vom GLUE das entsprechende ROM-Chip-Pärchen aktiviert.

Bei Verwendung von Megabit-ROMs kommt man locker mit zwei Chips (2 x 128 KByte = 256KByte) aus, um das komplette TOS darin unterzubringen. Allerdings muß dazu natürlich dafür gesorgt werden, daß sich dieses Chip-Pärchen über den gesamten ROM-Adreßbereich von $FC 0000..$FE FFFF "angesprochen" fühlt und auch alle Adreßleitungen am Chip vorhanden sind. Das Platinenlayout beim MEGA ST und 1040 STF(M) ist entsprechend ausgelegt. Die Mega-Bit-ROMs müssen dazu in die Fassungen für U63/U9 und U67/U10 eingesetzt sein. Über entsprechende Lötbrücken wird die korrekte Einbindung der ROM-Chips dann folgendermaßen vorgenommen (siehe dazu Abbildung 1.7!):

- Brücke "W2" auf Stellung "1M" versetzen, um die Adreßleitung A 17 der CPU an die Megabit-ROM-Chips (Pin 22 = A 16-Anschluß des ROMs) zu führen. (Bei 256KBit-ROMs ist der Chipanschluß (Pin 22) das "Output Enable (_OE)"-Signal und fest auf Low zu legen.)

- Brücke "W3" ebenfalls auf Stellung "1M" bringen. Damit wird die Adreßleitung A 16 der CPU an Pin 1 (= A 15-Anschluß) der Megabit-ROM-Chips geführt. (Bei 256KBit-ROMs ist Pin 1 des Chips intern nicht angeschlossen, bzw. bei EPROMs muß dort +5V angelegt werden. Bei eingelegter Brücke "W3" wird an alle Pin 1 der ROM-Fassungen +5V angelegt!)

– Die Brücke "W4" darf nicht eingelegt sein. Das 3fach NAND-Gatter in U12/U68 faßt dann die Signale _ROM0.._ROM2 zu einem einzigen "Chip Enable (_CE)"-Signal für die Megabit-ROM-Chips zusammen. Damit ist gewährleistet, daß die beiden Megabit-ROMs im gesamten TOS-ROM-Adreßbereich aktiviert sind! (Bei 256KBit-ROMs braucht jedes ROM-Pärchen ein eigenes "Chip Enable (_CE)"-Signal (_ROM0.._ROM2). Die Funktion von U12/U68 ist deshalb durch Auftrennen der Brücke am Ausgang des NAND-Gatters aufzuheben. Wichtig! Leider ist durch einen Platinen-Layout-Fehler bei den MEGA STs diese Brücke nicht vorhanden. Trennen geht also hier nur durch z. B. Auslöten des gesamten Gatterbausteins U12 oder durch Kappen des Pins 12 von U12!)

In Abbildung 1.7 ist die Bestückung der Brücken (sie werden im MEGA ST übrigens durch 0 OHM-Widerstände realisiert!) für 256KBit-ROMs eingezeichnet. Wo findet man nun den Gatterbaustein U12 (MEGA ST) bzw. U68 (1040STF(M)) auf der Platine?

Beim MEGA ST befindet sich U12 in der linken, unteren Ecke der Platine, direkt links neben der Quadratischen MMU. In der Nähe befinden sich auch die Brücken "W2" bis "W4"! Wegen der vielen Platinenversionen beim 1040STF(M) kann man als Anhaltspunkt nur sagen: In der Nähe der ROM-Fassungen auf die Suche gehen. Die Brücken sind beim 1040STF(M) durch Lötspiegel realisiert und wegen der Beschriftung der Platine relativ gut zu finden!

Würde ATARI einen anderen GLUE-Chip verwenden (dieser liefert unter anderem die Chip-Select-Signale _ROM0.._ROM2 für die ROM-Chips), so ließen sich in den 6 Chip-Fassungen 768 KByte ROM unterbringen. Die Schaltung und das Platinenlayout sind dafür bereits ausgelegt!

Nach einem RESET oder bei einem Kaltstart ist ein Rechner mit ROM-TOS also sofort "voll da", während dieser Vorgang mit einem von Diskette zu ladenden TOS ca. 35 Sek. dauert.

Außerdem steht bei TOS im ROM dann natürlich ein schönes Stück RAM mehr zur Verfügung, welches ja sonst vom einzuladenden TOS-Betriebssystem belegt wird. Es ist aber auch mit TOS im ROM weiterhin möglich, Boot-Disketten mit z. B. modifizierten oder neuen TOS-Versionen zu starten.

Das Cartridge-System des ST

Aber nicht nur das Betriebssystem kann in ROMs untergebracht werden! Der Adreßraum von $FA 0000 bis $FB FFFF kann durch ein ROM-Cartridge (Einsteckplatine mit ROM- bzw. EPROM-Chips) belegt werden, welches ein oder mehrere Programme enthalten kann. Es stehen 128 KByte Speicherplatz für ein solches Cartridge zur Verfügung. Die eventuelle Verwendung solcher Cartridges ist im Betriebssystem schon vorgesehen worden.

Der Vorteil von Software auf Cartridge liegt einmal in der schnellen Handhabung, denn das Laden vom Massenspeicher entfällt.

Als weiterer Pluspunkt ist anzuführen, daß ja kein Platz im RAM für diese Software benötigt wird. Dies gilt jedoch nur dann, wenn die Programme auch wirklich im Cartridge-ROM ablaufen und nicht erst über ein Treiberprogramm ins RAM kopiert und dann dort gestartet werden (wie es z. B. bei einer RAM-Disk gemacht wird)!

Ein neues Betriebssystem für den ST?

Entsprechend der Behandlung des Cartridge-Anschlusses in der Betriebssystem-Software ist es sogar möglich, mittels Cartridge ein neues Betriebssystem auf dem ST zu fahren! So existiert bereits als Cartridge das Betriebssystem "RTOS-UH" (Realtime Operating System – Uni Hannover) mit Compiler für die Programmiersprache "PEARL". Dieses Betriebssystem erlaubt das komfortable Bearbeiten von Echtzeit-Steuerungsaufgaben bei "gleichzeitiger" Ausführung mehrerer Aufgaben (Multitasking).

Ein sogenannter MAC-Emulator, mit dem das Betriebssystem des Apple-Macintosh erfolgreich auf dem ST nachgebildet wird, ist auch mit Cartridge-Unterstützung erhältlich! Dazu werden die Original Macintosh-ROMs in das Cartridge eingesetzt und mit einer Anpassungssoftware das Betriebssystem des MAC gestartet.

Anschlußplatinen für den Einsatz von "selbstgebrannten" EPROMs mit eigener Software sowie EPROM-Programmiergeräte zum Anschluß an den Cartridge-Slot sind ebenfalls von Drittanbietern lieferbar.

Weiterhin gibt es bereits Anschlußplatinen für den Cartridge-Port, mit denen die 128 KByte-Grenze "durchstoßen" wird. 512 KByte und mehr an residenter Software sind mittels Bank-Switching-Technik am Cartridge-Port abrufbar. Man unterteilt dazu den Speicherraum von 512 KByte oder mehr in einzelne Blöcke von z. B. 64 KByte. Eine entsprechende Treibersoftware (ebenfalls auf dem Cartridge) sorgt dann dafür, daß die CPU im Cartridge-Adreßraum immer den gerade notwendigen Block "zu sehen" bekommt, der gerade angesprochen werden soll.

Im TOS-Magazin 8/90 und 9/90 habe ich außerdem beschrieben, wie man den Cartridge-Port des ST als I/O-Port für einfache Steuer- und Überwachungsaufgaben verwenden kann. Folgende Signale stehen am Cartridge-Port zur Verfügung:

D0...D15 Der 16 Bit breite Datenbus (ungepuffert!) der CPU

A1...A15 Die unteren 15 Adreßleitungen (ungepuffert!) der CPU

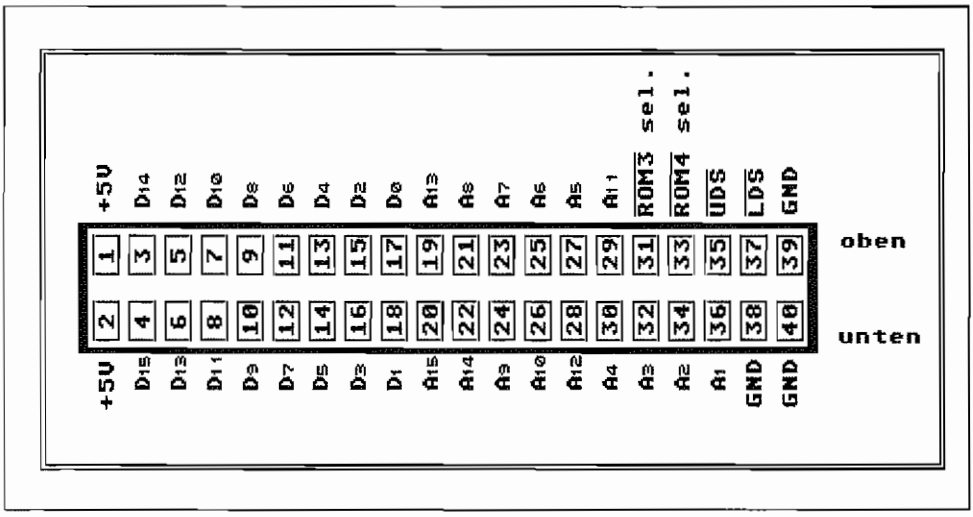


Abb. 1.8: Die Belegung des Cartridge-Anschlusses (von außen auf den Anschluß gesehen)

LDS/UDS	Lower-/Upper-Data Strobe (Aktiv=low!). Dient zur Decodierung, ob Wort- oder Einzelbyte-Zugriff erfolgen soll.
ROM3 sel.	Wird Low, sobald eine Adresse im Bereich von $FB 0000 bis $FB FFFF angesprochen wird (64 KByte-Block). Als "Chip-Select"-Signal für die ROM-Chips eines Cartridges benutzbar.
ROM4 sel.	Wird Low, wenn eine Adresse im Bereich von $FA 0000 bis $FA FFFF ausgelesen wird (64 KByte Block). Ebenfalls als "Chip-Select" für Cartridge-ROM-Chips benutzbar.
+5V	Versorgungsspannung für das Cartridge
GND	Masseanschluß für das Cartridge

Wer bereits mit den 8-Bit-Computern von ATARI gearbeitet und sich mit dem dort ebenfalls schon realisierten Cartridge-Konzept beschäftigt hat, dem wird das Prinzip vertraut vorkommen.

Gegenüber dem Cartridge-Slot der 8-Bit-Rechner hat man leider beim ST nicht mehr die Möglichkeit, in das Cartridge zu schreiben. Es ist aber nicht so, daß für die R/W-Leitung kein Platz mehr gewesen oder diese nur "vergessen" worden wäre.

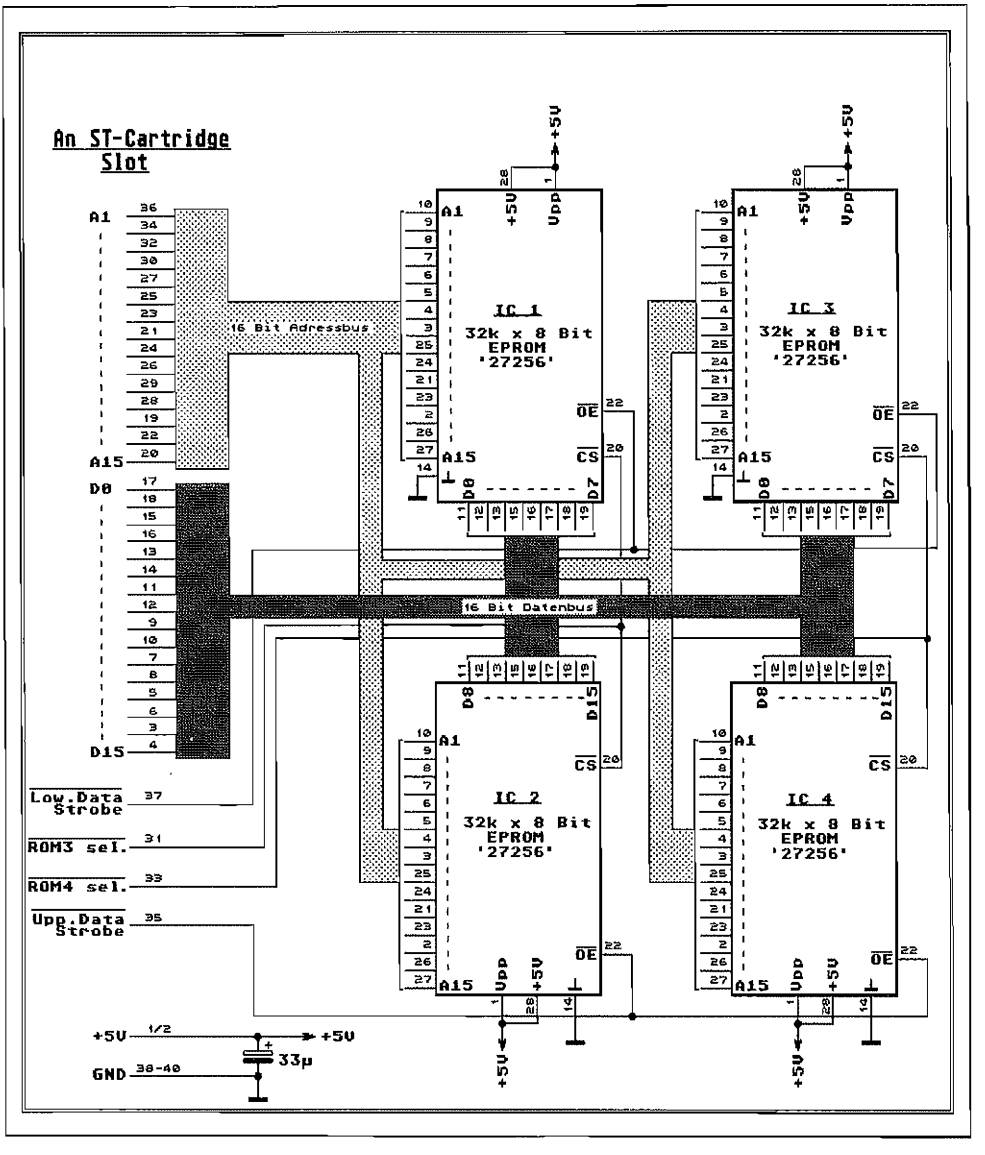


Abb. 1.9: So könnte die Schaltung für ein 128 KByte-Cartridge aussehen

Bei jedem Schreibversuch in den Adreßbereich $FA 0000 bis $FE FFFF meldet der GLUE-Chip der CPU einen "BUS-ERROR". Schreibzugriffe in diesen Bereich sind also nicht

erwünscht und vorgesehen! Außerdem hat ATARI bei der Realisierung des Cartridge-Slots für die Direktsteckvorrichtung ein unübliches Rastermaß von 2 mm verwandt. Warum man hier nicht auf dem sonst üblichen Maß von 1/10 Inch = 2,54 mm geblieben ist, ist nicht ohne weiteres nachvollziehbar.

Von Drittanbietern sind aber Adaptersteckverbinder von 2mm –> 2,54mm Raster mit zum Teil Pufferbausteinen für die Adreß- und Datenbussignale lieferbar.

Außerdem kann der Hardewarebastler auf Leerplatinen zurückgreifen, die zum Aufbau eigener Schaltungen am Cartridgeport verwendet werden können.

Die Behandlung von Cartridge-Software durch das Betriebssystem

Wie bereits erwähnt, wird vom Betriebssystem das Vorhandensein eines Cartridges schon bei der Systeminitialisierung berücksichtigt. So bestehen verschiedene Möglichkeiten, aus dem normalen "Hochfahren" des Computersystems "auszusteigen" und die weitere Kontrolle über den ST an die Cartridge-Software zu übergeben.

Die "Anwesenheit" eines Cartridges wird vom Betriebssystem durch Lesen des ersten Langwortes ($FA 0000) im Cartridge-Speicherbereich überprüft. Abhängig von dem Wert dieses "magischen Langworts" wird evtl. schon direkt nach einem RESET die Kontrolle an das Cartridge übergeben oder erst zu einem späteren Zeitpunkt. Lautet der Inhalt des magischen Langworts auf $FA52 235F, so wird nahezu unmittelbar nach einem System-RESET nach Adresse $FA 0004 im Cartridge verzweigt. In Register A6 der CPU wird jedoch für den Fall, daß später die normale Systeminitialisierung fortgesetzt werden soll, eine RETURN-Adresse mit an die Cartridge-Software übergeben.

Cartridge-Software, welche schon zu diesem Zeitpunkt die "Herrschaft" über den ST übernimmt, muß alle Hardware-Komponenten des ST selbst initialisieren, die benutzt werden sollen. Anwendung findet dieser frühzeitige Ausstieg aus der Systeminitialisierung z. B. beim Geräte-Test im Reparaturservice.

Mittels eines sogenannten Diagnostic-Cartridge ist es so möglich, den ST durchzuchecken. Es werden durch die Diagnose-Software alle Systemteile angesprochen, und aus entsprechenden Rückmeldungen (oder eben auch nicht!) kann auf die Funktionsfähigkeit geschlossen werden. Im Regelfall wird Cartridge-Software jedoch auf ein voll initialisiertes System zurückgreifen wollen. Außerdem lassen sich auf einem Cartridge mehrere Programme unterbringen.

In so einem Fall muß bei Adresse $FA 0000 der Inhalt des magischen Langwortes auf $ABCD EF42 lauten. Außerdem müssen im Cartridge noch einige Angaben mehr enthalten

sein. So ist für jedes Programm ein sogenannter "Programm-Vorspann" (Cartridge-Application-Header) erforderlich, der folgendermaßen aufgebaut sein muß:

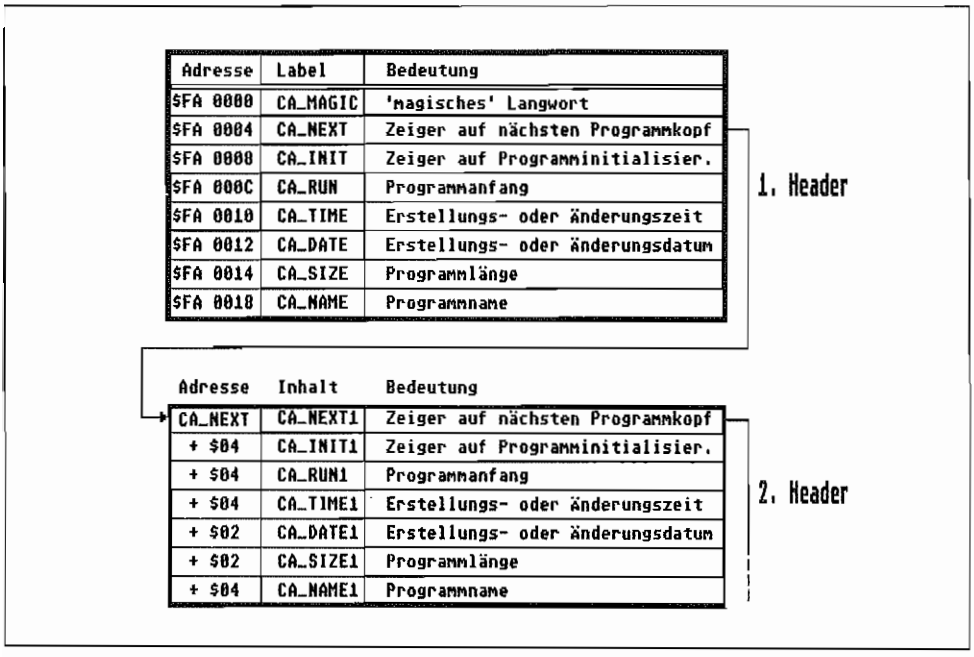


Abb. 1.10: So sind die Programm-Vorspänne beim ST-Cartridge aufgebaut

Die Label haben dabei im einzelnen folgende Bedeutung:

CA_NEXT Wenn mehrere Programme auf dem gleichen Cartridge enthalten sind, enthält CA_NEXT einen Pointer auf den nächsten Programmvorspann. Ist nur ein Programm vorhanden oder ist der betrachtete Programmvorspann der letzte auf dem Cartridge, steht hier $0000 0000!

CA_INIT Anfangsadresse einer evtl. Cartridge-Programm-Initialisierung. Die Bits 24..31 bleiben für die Adressierung unberücksichtigt und enthalten Informationen, zu welchem Zeitpunkt der ST-Systeminitialisierung nach CA_INIT verzweigt werden soll. Ist das entsprechende Bit gesetzt, so erfolgt Ausführung.

Bit 24 Initialisierung oder Start der Cartridge-Software nach erfolgter Hardwareinitialisierung. Die Betriebssystemvariablen und Interruptvektoren sind schon ge-

setzt. Die Bildschirmauflösung ist bereits ermittelt und eingestellt. Die Interrupts sind jedoch in der CPU noch gesperrt (Interrupt-Priority-Level auf 7).

Bit 25 Initialisierung oder Start der Cartridge-Software wie bei gesetztem Bit 24, jedoch erst unmittelbar nach Freigabe der Interrupts in der CPU (Interrupt-Priority-Level ist dann auf 3 gesetzt) und vor der GEMDOS-Initialisierung.

Bit 26 Initialisierung oder Start der Cartridge-Software etwas früher, als für Bit 24 beschrieben. Einstellung der aktuellen Bildschirmauflösung ist noch nicht erfolgt.

Bit 27 Cartridge-Programm-Initialisierung oder -Start erfolgt erst unmittelbar vor einem evtl. Disk-Boot. Sonst wie bei Bit 25.

Die Bits 29 .. 31 sind nur dann von Bedeutung, wenn Cartridge-Programme als GEMDOS-Anwendungen laufen. (Es ist mir allerdings noch nicht gelungen, Parameter an eine TTP-Anwendung im Cartridge zu übergeben, wie man das bei "normalen" TTP-Programmen problemlos mittels der Dialogbox bewerkstelligen kann. Die Angaben für die Bits 29...31 beruhen auf Informationen aus dem "Hitchhiker's guide to the BIOS" von ATARI.)

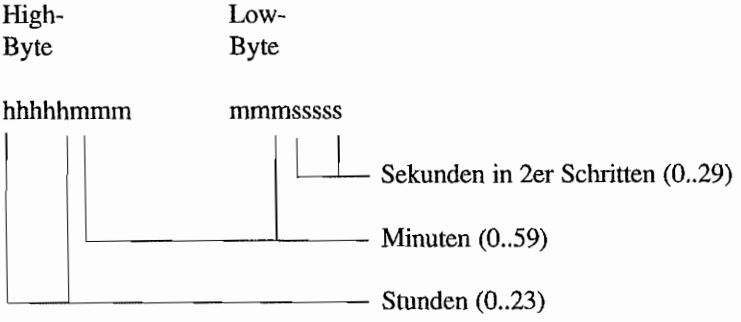
Bit 29 Programm gehört zum Desktop (Accessory)

Bit 30 TOS-Applikation

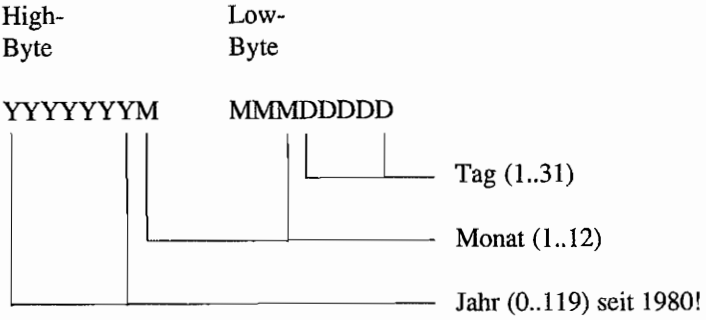
Bit 31 TOS-Applikation mit der Möglichkeit, Parameter an das Programm zu übergeben. (TTP-Anwendung =TOS takes Parameters)

CA_RUN Adresse des Programmanfangs eines Cartridge-Programms. An der hier angegebenen Adresse beginnt das eigentliche Programm. Alle evtl. nötigen Initialisierungen sind schonerfolgt. Nur nötig, wenn Programm unter GEMDOS läuft.

CA_TIME Uhrzeit der Programmerstellung im Format (dient nur zur Information des Benutzers).



CA_DATE Datum der Programmerstellung im Format (dient ebenfalls nur zur Information des Benutzers.)



CA_SIZE Größe des Einzelprogramms. Wird eigentlich nicht benötigt!

CA_NAME Name des Programms im GEMDOS-Format (FILENAME.EXT+$00), also max. acht Zeichen für den Programmnamen, evtl. drei Zeichen für die Erweiterung mit dazwischenliegendem Trennungspunkt, abgeschlossen durch $00. Nur zur Information des Benutzers.

Cartridge-Programme, die unter GEMDOS laufen, sind auf dem virtuellen Laufwerk c: zu finden (der Laufwerksbezeichner ist ein kleines "c").

Gestartet werden die Programme mit dem gewohnten Doppelklick auf das gewünschte Programmsymbol. Software, die im Cartridge laufen soll, muß für den entsprechenden Adreßbereich (ROM!) geschrieben sein, in dem sie später ablaufen soll, oder voll relokatibel angelegt sein. GEMDOS hat ja keine Möglichkeit, anhand eines Programm-Headers irgendwelche Adressen im TEXT- oder DATA-Bereich im Cartridge (ROM!) zu relozieren.

Cartridge-Programme, die RAM-Speicher benötigen (Pufferspeicher u.ä.), müssen sich diesen Speicher "zuteilen" lassen (über die GEMDOS-Funktion "Malloc") und selbst verwalten. Ein BSS-Bereich (Bereich für nichtinitialisierte Daten, wie er z. B. für Zwischenspeicherung von Benutzereingaben benutzt wird) ist ja im ROM schlecht zu realisieren!

Beim Aufruf eines Cartridge-Programms legt GEMDOS auch eine Basepage an, die aber im wesentlichen nur die Startadresse des Programms und einen Pointer auf den Anfang der eigenen Basepage sowie die Basepage des aufrufenden Programms enthält.

Soviel zum Cartridge-Slot des ST.

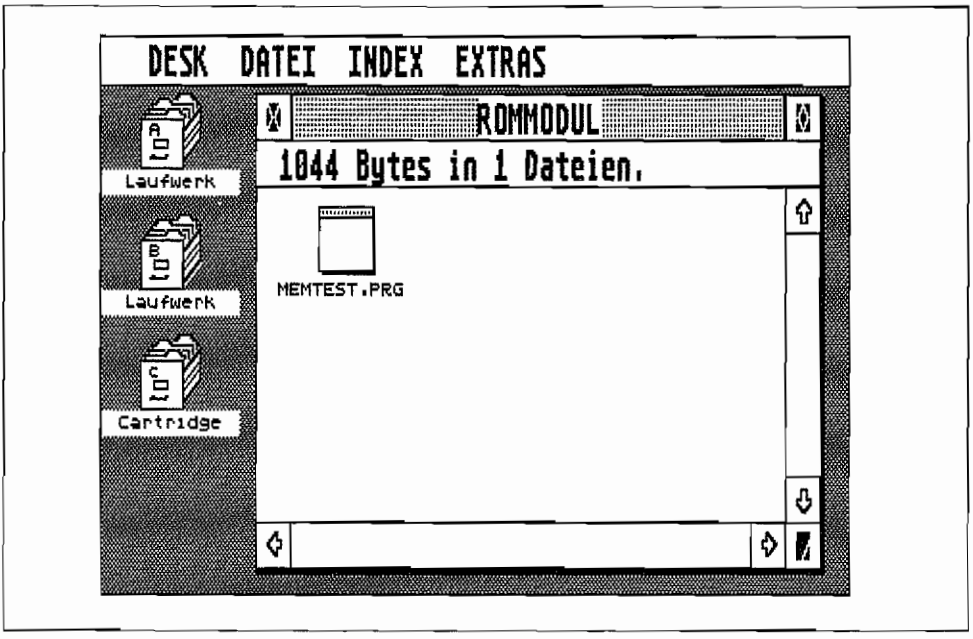


Abb. 1.11: So erscheinen Cartridge-Programme auf dem Desktop

DMA im ST

Die Konstrukteure des ATARI ST haben der CPU einen tatkräftigen Helfer zur Seite gestellt, der zwar nicht besonders viel "Intelligenz" besitzt, dafür aber ganz schön schnell arbeiten kann.

Es handelt sich hierbei um eine DMA-Einheit, die von ATARI ebenfalls speziell für die ST-Serie entwickelt wurde. Mittels DMA = Direct Memory Access (zu deutsch etwa: direkt auf den Hauptspeicher zugreifend) ist es möglich, große Datenmengen von und zu Peripheriegeräten wie z. B. Floppy-Disk- und Hard-Disk-Stationen ohne Umwege über die CPU und deren Register zu transportieren. Der DMA-Baustein kann dabei gleichberechtigt mit der CPU auf das RAM zugreifen. Wer zuerst kommt, darf an die Daten!

Datentransfer mit Turbolader

Durch den DMA-Baustein bedient der ST den Floppy-Disk-Controller (FDC) und das sogenannte ACSI (ATARI Computer System Interface). Der Datenstrom vom und zum Floppy-

Disk-Controller könnte mit einer Transferrate von bis zu 500 KBit/Sekunde "fließen", wobei der eingebaute FDC jedoch nur eine Übertragungsrate von 250 KBit/s verkraftet. Beim ACSI handelt es sich um einen 8 Bit breiten Bus zum Anschluß von schnellen Peripheriegeräten wie z. B. eine Festplatte oder einen Laserdrucker. Die Datentransferrate beträgt hierbei immerhin 8 MBits/Sek (bei der Harddisk– ca. 12 MBits/Sek sind aber durchaus noch drin, wenn das Peripheriegerät da mitkommt!).

Die CPU allein würde dabei schon ganz schön ins Schwitzen geraten und hätte kaum noch Zeit für andere "Hobbys". Durch Verwendung der DMA-Einheit ist es nun nur noch erforderlich, diese entsprechend zu "informieren", wo die zu transportierenden Daten im RAM abgelegt oder geholt werden (Startadresse) und wie viele Daten (Blocks zu je 512 Bytes) zum oder vom Peripheriegerät übertragen werden sollen.

Damit die DMA-Einheit überhaupt wirkungsvoll arbeiten kann, ist sie darauf ausgelegt, Datenblöcke mit 512 Bytes (beim ST ist das sinnvollerweise die Größe eines Datensektors auf Disk) oder einem Vielfachen davon zu übertragen.

Die weitere Abwicklung der Datenübertragung liegt nun bei der DMA-Einheit, so daß die CPU mit anderen Aufgaben beschäftigt werden kann (vom TOS wird diese Möglichkeit leider nicht ausgenutzt!).

Die Hälfte reicht auch

Wie das Blockschaltbild zeigt, wird der 16 Bit breite Datenbus des Hauptspeichers durch die DMA-Einheit auf einen 8 Bit breiten Datenbus reduziert. Die Umsetzung eines 16 Bit-Datenwortes in zwei aufeinanderfolgende 8-Bit-Datenworte erledigt die DMA-Einheit automatisch. Ein 16 Bit breiter Bus auf der Peripheriegeräteseite ist auch nicht erforderlich, da der Floppy-Disk-Controller (FDC) nur über einen 8-Bit-Datenbus verfügt.

Der ACSI-Anschluß ist wegen seiner Anlehnung an den verbreiteten SCSI-Bus (SCSI = Small Computers System Interface) ebenfalls nur 8 Bit breit ausgefallen.

Die DMA-Einheit verfügt intern über fünf Register, mit denen die Steuerung des Datentransfers zum/vom Floppy-Disk-Controller und dem ACSI-Bus eingestellt werden kann. Ein 32 Byte großer Puffer nach dem FIFO-Prinzip (First In, First Out = zuerst eingebrachte Daten werden auch zuerst abgeholt) in der DMA-Einheit übernimmt die Pufferung zwischen dem Hauptspeicher und dem Floppy-Disk-Controller/ACSI-Bus.

Außerdem geschieht dort die Umsetzung eines 16-Bit-Datenwortes in zwei 8-Bit-Datenworte und umgekehrt.

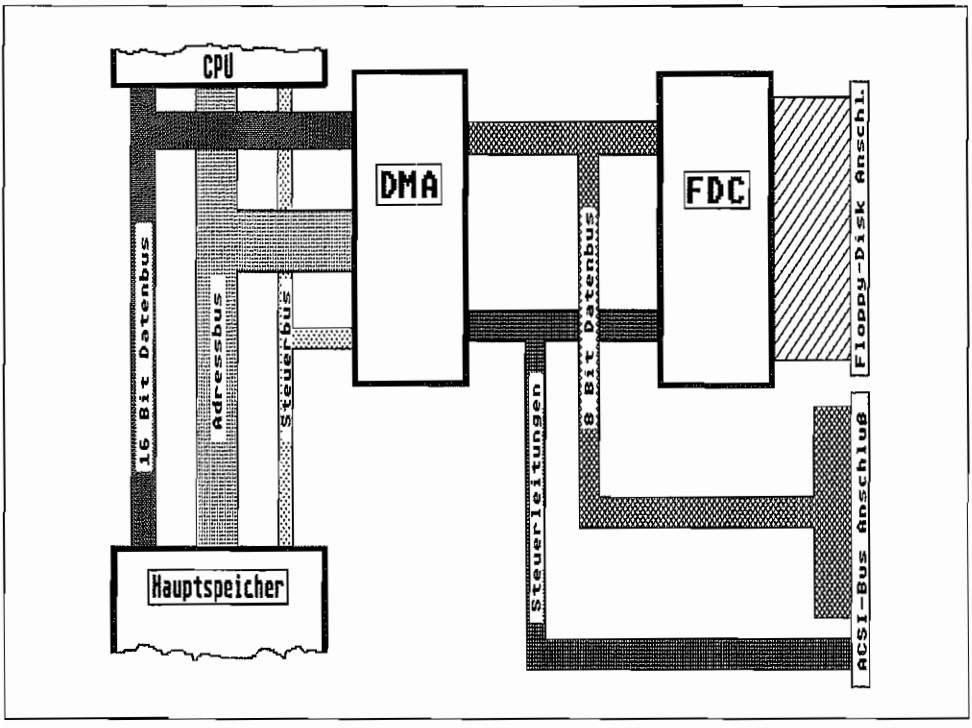


Abb. 1.12: Die DMA-Einheit bedient den FDC und den ACSI-Bus

Hierbei ist zu beachten, daß ein Transfer zwischen DMA-Einheit und Hauptspeicher immer erst dann erfolgt, wenn der Fifo-Buffer zu mehr als zur Hälfte gefüllt ist.

Die DMA-Einheit versucht dann, die Buskontrolle zu übernehmen und 16 Bytes zu übertragen. Eine Übertragung erfolgt also immer in Bursts zu je 16 Byte! Bei Datentransfers in Sektorgröße (512 Bytes) braucht man darauf nicht so sehr zu achten.

Will man jedoch weniger als einen Sektor (512 Bytes) von Disk lesen, also z. B. nur Adreßfelder (die sind nur 6 Bytes lang), so sollten mindestens drei dieser Felder gelesen werden, um auf eine Bytezahl von über 16 zu kommen. Sonst bleiben die gelesenen Daten im Fifo-Buffer hängen, und man wundert sich nach einer anscheinend doch korrekt ausgeführten DMA-Operation, daß keine Daten im Hauptspeicher angekommen sind.

So wie in der Abbildung 1.13 dargestellt, kann man sich ungefähr das Innenleben der DMA-Einheit des ST vorstellen.

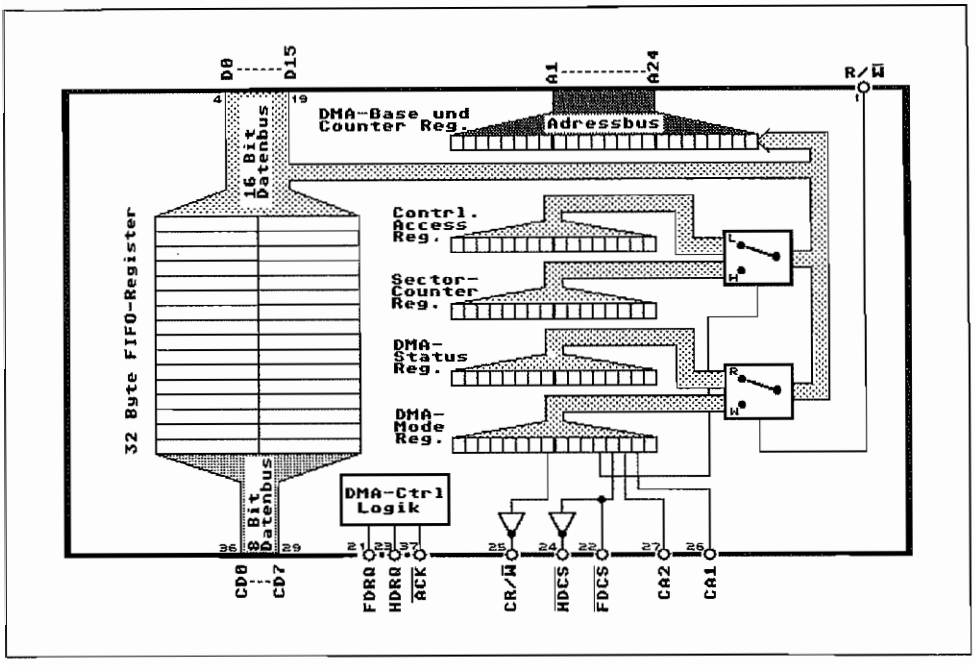


Abb. 1.13: So ist die DMA-Einheit im Prinzip aufgebaut

Für die Steuerung der Peripheriebausteine wie FDC und ACSI-Bus sind unter anderem die CR/W-Leitung (Controller Read/Write = Lesen/Schreiben-Signalisierung für FDC oder ACSI-Bus (bei Low erfolgt Schreibzugriff)) und die HDCS-Leitung (Hard Disk Controller Select = ACSI-Bus-Zugriffssignalisierung) nötig.

Über die FDCS-Leitung wird dem Floppy-Controller-Chip ein Zugriff signalisiert (Floppy-Disk-Controller-Select). Die CA1/CA2-Leitungen dienen als Adreßleitungen zur Registerauswahl beim FDC-Chip oder zur Command-Byte-Signalisierung am ACSI-Bus.

Wie man sieht, wird durch den Zustand der R/W-Leitung am Eingang der DMA-Einheit entschieden, ob ein Zugriff auf das DMA-Mode-Register oder das DMA-Status-Register erfolgt. So kann das DMA-Status-Register nur ausgelesen und das DMA-Mode-Register nur beschrieben werden. Beide Register sind im Speicherraum des ST unter der gleichen Adresse zu finden!

Durch das Bit 4 im DMA-Mode-Register läßt sich steuern, ob auf das Controller-Access-Register oder das Sector-Counter-Register zugegriffen werden soll. Diese beiden Register

können sowohl beschrieben als auch ausgelesen werden und belegen ebenfalls nur eine Adresse.

Die einzelnen Register haben nun folgende Funktionen:

DMA-Base- und Counter-Register

Adresse	Zugriff	Funktion	Label
$FF 8609	R/W	High-Byte DMA-Base- und Counter-Register	"dmahigh"
$FF 860B	R/W	Mid-Byte DMA-Base- und Counter-Register	"dmamid"
$FF 860D	R/W	Low-Byte DMA-Base- und Counter-Register	"dmalow"

Diese drei jeweils 8 Bit breiten Register werden zu Beginn des Datentransfers mit der Startadresse des zu übertragenden Datenbereichs geladen. (Wichtig! Reihenfolge des Zugriffs: Low-, Mid-, High-Byte!)

Im DMA-Betrieb werden die Register von der DMA-Einheit nach jedem übertragenen Datenwort automatisch weitergezählt. Die drei Register bilden zusammen also einen Pointer auf das jeweils nächste zu übertragende Datenwort vom bzw. zum Hauptspeicher.

Controller-Access-Register

Adresse	Zugriff	Funktion	Label
$FF 8604	R/W	Durchgriff auf FDC-Register/ACSI-Bus	"diskctl"

Dieses Register ist nur zugänglich, wenn Bit 4 im DMA-Mode-Register "0" ist! Wenn man hier Daten einschreibt, werden diese von der DMA-Einheit an eines der FDC-Register weitergereicht oder auf den ACSI-Bus ausgegeben. Ein Lesezugriff auf diese Adresse ergibt den Inhalt eines der FDC-Register oder eines Bytes vom ACSI-Bus.

Ein Durchgriff auf FDC-Register erfolgt, wenn Bit 3 im DMA-Mode-Register "0" ist. Welches FDC-Register man im Zugriff hat, legen dann Bit 1 und 2 im DMA-Mode-Register fest (diese steuern die "Adreßleitungen" CA1 und CA2 für die FDC-Register). Das Controller-Access-Register hat zwar Wortbreite, für FDC-Register-Zugriffe sind jedoch nur die unteren 8 Bit relevant (8-Bit-Bus zum FDC)!

Für einen Zugriff auf den ACSI-Bus muß Bit 3 des DMA-Mode-Register gesetzt sein. Beim Durchgriff auf den ACSI-Bus ist auch hier wieder nur das niederwertige Byte des Registers "diskctl" relevant!

Sector-Counter-Register

Adresse	Zugriff	Funktion	Label
$FF 8604	R/W	Zähler für Datenblöcke in Sektorgröße	"diskctl"

Über dieses Register wird der DMA-Einheit mitgeteilt, wie viele Blöcke (Sektoren) zu je 512 Bytes vom bzw. zum FDC/ACSI-Bus übertragen werden sollen. Der Zugriff auf dieses Register ist nur möglich, wenn im DMA-Mode-Register das Bit 4 gesetzt ist! Da dieses Register nur 8 Bit breit ist, kann man in einem Rutsch nur 255 * 512 = 127,5 KBytes übertragen.

Bei größeren Datenmengen muß man also mehrere "Pakete" direkt hintereinander versenden. Durch das Beschreiben dieses Registers wird der DMA-Vorgang intern schon gestartet. So wird die DMA-Einheit bei einem Schreibvorgang auf Floppy oder ACSI-Bus schon mal beginnen, den Fifo-Buffer aus dem Hauptspeicher mit Daten zu füllen.

Wenn dann die DMA-Freigabe durch Löschen von Bit 7 im DMA-Mode-Register erfolgt, reagiert die DMA-Einheit auch auf Datenanforderungen vom FDC (FDRQ = Floppycontroller-Data-Request) oder ACSI-Bus (HDRQ = Harddiskcontroller-Data-Request).

DMA-Status-Register

Adresse	Zugriff	Funktion	Label
$FF 8606	R	Status der ausgeführten DMA-Operation	"fifo"

Nur die unteren 3 Bits dieses 16-Bit-Registers werden für Zustandsinformationen über eine DMA-Operation benutzt.

Die Bits sind Low-aktiv, d. h., wenn der genannte Zustand aufgetreten ist, wird das Bit auf "0" zurückgesetzt!

Bit	Bedeutung
0	Der "Turbolader" hatte eine Funktionsstörung. Es ist ein Fehler während der DMA-Operation aufgetreten.
1	Das Sector-Counter-Register ist bis auf Null heruntergezählt.
2	DATA REQUEST von FDC/ACSI-Bus ist inaktiv.

Die Bits 1 und 2 werden jedoch vom Betriebssystem nicht benutzt. Vor einem Zugriff auf das DMA-Status-Register sollte lt. ATARI-Information immer erst das Sector-Counter-Register selektiert werden (Bit 4 in DMA-Mode-Register setzen)!

DMA-Mode-Register

Adresse	Zugriff	Funktion	Label
$FF 8606	W	"Kommando"-Register für die DMA-Einheit	"fifo"

Die Bedeutung der einzelnen Bits ist in Abbildung 1.14 erläutert.

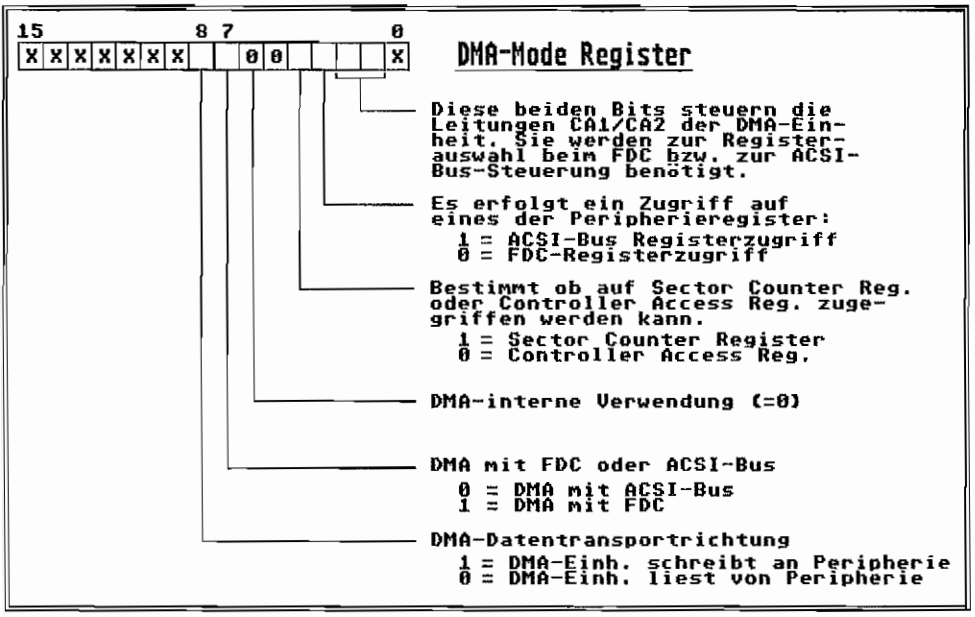


Abb. 1.14: Das DMA-Mode-Register

Das Bit 8 (DMA-Datentransport-Richtung) hat noch eine besondere Funktion in Verbindung mit dem FIFO-Buffer. Da der FIFO-Buffer nicht automatisch zu Beginn einer jeden neuen DMA-Operation gelöscht wird, hat der Programmierer selbst für "saubere" Verhältnisse zu sorgen. Durch "Klappern" (Wechseln des Zustands) mit Bit 8 des DMA-Mode-Registers wird nämlich der Fifo-Buffer gelöscht und die DMA-Einheit auf einen DMA-Transfer vorbereitet!

Das Löschen des FIFO-Buffers wird sinnvollerweise gleichzeitig mit dem Einstellen der DMA-Datenübertragungs-Richtung durchgeführt. Beispiel:

```
move.w  #$090,dma_mode   ; Diese Befehlssequenz löscht den
move.w  #$190,dma_mode   ; FIFO-Buffer in der DMA-Einheit
move.w  #$090,dma_mode   ; und schaltet diese auf Lesen
```

(Bei Schreibzugriffen ist entsprechend gegensätzlich zu verfahren!) Soviel an dieser Stelle zur DMA-Einheit des ATARI ST. Weitere Informationen sind bei der Beschreibung des Floppy-Disk-Interface und des ACSI-Busses zu finden.

Der Systembus des MEGA ST

Erstmals bei den Geräten der MEGA ST-Serie hat ATARI die Möglichkeit vorgesehen, Erweiterungen des Systems vorzunehmen. So wurde im Gehäuse des MEGA ST Platz gelassen, um eine zusätzliche Erweiterungskarte unterzubringen. Die Karte wird liegend über die Hauptplatine (Mainboard) montiert (in der linken Gehäusehälfte). Dabei können zwei verschiedene Typen von Erweiterungskarten eingebaut werden. Die sogenannte "Half Card" hat die Abmessungen 137 x 145 mm, während die "Full Card" ca. 137 x 280 mm groß ist.

In beiden Fällen ist an der Gehäuserückseite eine Befestigungsmöglichkeit für die Karten vorgesehen. Außerdem kann ein bereits vorbereiteter Ausschnitt der Gehäuserückwand entfernt werden, um so Platz für Anschlußbuchsen oder -kabel zu erhalten.

Weitere Stabilität für die Erweiterungskarte ergibt sich durch die Befestigung der Karte mit Abstandshaltern im Gehäuseboden des MEGA ST. Entsprechende Aussparungen in der Hauptplatine des MEGA ST und Löcher zur Aufnahme der Befestigungsschrauben im Gehäuseboden sind bereits vorgesehen.

Die elektrische Verbindung erfolgt über eine 64pol. Steckverbindung. An der Erweiterungskarte sitzt auf der Lötseite eine zweireihige Federleiste (Buchse) nach DIN 41612, Bauform B. Auf dem Mainboard des MEGA ST befindet sich das Gegenstück dazu, eine zweireihige Messerleiste.

Pinbelegung des MEGA ST-Systembusanschlusses

An den 64 Anschlüssen des Erweiterungsanschlusses stehen im Prinzip alle Signale der 68000er-CPU zur Verfügung. D. h., daß sowohl der komplette Adreß- als auch der Datenbus und alle wesentlichen Steuer- und Meldeleitungen herausgeführt sind!

Die Einbindung des Systembusanschlusses (im folgenden kurz Megabus genannt) in die Hardware des MEGA ST zeigt der Schaltungsauszug in Abbildung 1.15. Alle Ausgangssignale sind ungepuffert und können einen LS-TTL-Eingang treiben.

Das Signaltiming am Megabus ist das einer normalen 68000er-CPU mit einem Takt von 8 MHz. Deshalb können periphere Einheiten auch ohne Probleme so angeschlossen werden, wie sie an eine normale 68000er-CPU angeschlossen würden. Natürlich kann die Logik auf der Erweiterungskarte auch die Bussteuerung übernehmen, wenn sie sich an das gleiche Protokoll hält, als wenn sie an eine 68000er-CPU angeschlossen würde.

Dazu muß dem augenblicklich aktiven Busmaster (das ist die Einheit, welche gerade die Kontrolle über die Bussteuerung besitzt, also in der Regel die CPU) mit einem Low auf der BR-Leitung (BR = Bus-Request = Anfrage für Busübernahme) signalisiert werden, daß eine andere intelligente Einheit die Bussteuerung übernehmen will.

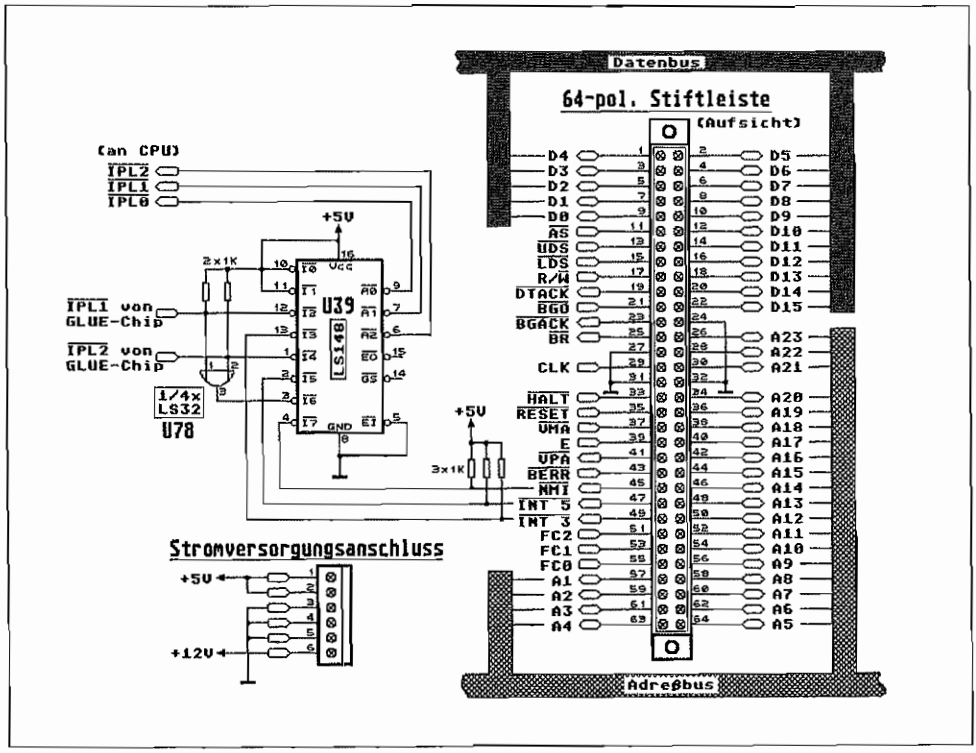


Abb. 1.15: Anschlußbelegung des Systembusses im MEGA ST

Der Busmaster gibt mit einem Low auf der BG-Leitung (Bus-Grant = Buszuteilung) an die anfragende Einheit sozusagen sein Einverständnis, daß nach Beendigung des gerade laufenden Buszyklusses die Bussteuerung abgegeben wird. Jetzt wartet die anfragende Einheit, bis der laufende Buszyklus beendet ist (AS = Adress-Strobe geht dann auf High, und DTACK muß ebenfalls auf High sein!), und übernimmt dann die Bussteuerung, indem sie als neuer Busmaster jetzt ein Low auf die BGACK-Leitung (Bus-Grant-Acknowledge = Quittung für Buszuteilung) legt und die BR-Leitung wieder auf High setzt. BGACK muß für mindestens einen Buszyklus des neuen Busmasters auf Low bleiben. Der neue Busmaster behält die Bussteuerung so lange, wie er BGACK auf Low hält.

Im MEGA ST sind bereits zwei Einheiten an der Arbeit (DMA-Baustein und BLITTER), welche ebenfalls von Zeit zu Zeit als Busmaster auftreten. Deshalb muß eine weitere Einheit, die in der Lage ist, die Bussteuerung zu übernehmen, sich den Bus mit den bereits vorhandenen Einheiten teilen. Dazu wird das Signal BGO (Bus-Grant-Out) an Pin 21 des Megabusanschlusses benutzt. Das BGO-Signal am Megabus wird nicht direkt von der CPU, sondern vom GLUE erzeugt und berücksichtigt bereits die anderen DMA-Bausteine im Mega ST, die ebenfalls die Bussteuerung übernehmen können (Daisy-Chaining = Verkettung unter Berücksichtigung der Prioritäten von ST-DMA-Einheit, BLITTER und intelligenter Einheit am Megabus).

Sind die Interrupts für die 68000er-CPU nicht gesperrt, sollte gewährleistet sein, daß ein anderer Busmaster als die CPU den Bus nicht länger als jeweils ca. 50 Buszyklen beansprucht, um noch eine einigermaßen niedrige Reaktionszeit auf Interruptanforderungen zu erhalten!

Liegt die Bussteuerung bei der Erweiterung am Megabus, müssen die Buszyklen exakt wie die einer mit 8 MHz arbeitenden 68000er-CPU ausgeführt werden, da sonst Störungen mit dem Video-Controller auftreten können. Die Zugriffe des Video-Controllers auf das RAM müssen auch unter einem anderen Busmaster wie gewohnt ablaufen können.

Nun noch eine kurze Beschreibung der einzelnen Anschlüsse des Megabusses:

CLK	1:1 Rechteck-TTL-Taktsignal mit einer Frequenz von 8.0106 MHz.
RESET, HALT	Diese Anschlüsse dienen zur Initialisierung des Prozessors und des gesamten Systems. Die Anschlüsse sind bidirektional. RESET ist auf dem Mainboard mit einem 1K-Pullup-Widerstand abgeschlossen, HALT mit einem 4K7-Pullup-Widerstand.

Führt die 68000er-CPU den Befehl RESET (CPU-Status und -Register bleiben unbeeinflußt) aus, wird die RESET-Leitung für ca. 15 µSek. (124 Taktzyklen) auf LOW gezogen und setzt damit externe Bausteine zurück. Kommt das RESET-Signal von einem anderen Baustein, so wird die CPU zurückgesetzt.

Die CPU zieht den HALT-Anschluß auf LOW, um damit auf ihren Funktionsstillstand hinzuweisen, den sie einnimmt, weil z. B. ein doppelter Busfehler aufgetreten ist.

Wird der HALT-Anschluß von außen auf LOW gebracht, stellt die CPU am Ende eines laufenden Buszyklus alle weiteren Tätigkeiten ein und versetzt alle Leitungen in den hochohmigen Zustand.

Ein System-RESET erfolgt durch LOW-Setzen beider Leitungen RESET und HALT beim Einschalten oder durch Druck auf den RESET-Knopf. Der Einschaltreset oder der Druck auf den RESET-Knopf veranlaßt, daß beide Leitungen für ca. 130 msec auf LOW gezogen werden.

A1 – A23	Der 23-Bit-Adreßbus direkt von der CPU. Auf dem Mainboard sind die Adreßleitungen mit 4K7-Pullup-Widerständen abgeschlossen.
D0 – D15	Der 16-Bit-Datenbus (bidirektional) der CPU. Auf dem Mainboard mit 10K-Pullup-Widerständen abgeschlossen.
FC0, FC1, FC2	Funktionscode-Leitungen von der CPU. Sie geben den Betriebszustand des Prozessors für den laufenden Buszyklus an. Abgeschlossen sind diese direkt von der CPU kommenden Leitungen auf dem Mainboard mit 10K-Pullup-Widerständen.
AS	Adress-Strobe. Bei LOW signaliert diese Leitung gültige Adreßinformationen auf dem Adreßbus. 4k7-Pullup auf Mainboard!
R/W	Read/Write-Leitung. LOW zeigt einen Schreibzyklus, HIGH einen Lesezyklus an. 4k7-Pullup auf Mainboard!
UDS, LDS	Upper-Data-Strobe, Lower-Data-Strobe. LOW an UDS signalisiert gültige Informationen auf dem höherwertigen Byte des Datenbusses. LOW an LDS zeigt gültige Daten auf dem niederwertigen Byte des Datenbusses an. Bei einem Wortzugriff sind beide Leitungen auf LOW. Beide Leitungen sind mit 4k7-Pullups auf dem Mainboard abgeschlossen!
DTACK	Data-Transfer-ACKnowledge. Bidirektionaler Anschluß (Open Collector). Mit einem LOW wird ein Datentransfer bestätigt. Ein Erweiterungsbaustein, dem Daten von der CPU zugeschrieben werden, muß der CPU den Empfang der Daten mit einem LOW auf dieser Leitung bestätigen.

Ist die periphere Einheit auf der Erweiterungskarte Busmaster, so muß diese an der DTACK-Leitung "lauschen", ob der Datenaustausch korrekt abgelaufen ist. Mit einem 1K-Pullup-Widerstand ist diese Leitung auf dem Mainboard abgeschlossen.

BERR — Bus Error. Zeigt der CPU oder einer DMA-Einheit an, das keine periphere Einheit auf den laufenden Buszyklus reagiert hat. Beim ST wird automatisch nach 64 Taktzyklen diese Leitung auf LOW gesetzt, wenn bis dahin keine Reaktion von einem angesprochenen Peripheriebaustein erfolgt ist. Diese Open-Collector-Leitung der CPU ist mit einem 4K7-Pullup-Widerstand auf dem Mainboard abgeschlossen.

E — Enable Clock. Dient zur Synchronisation beim Anschluß von Peripheriebausteinen der 6800-Serie. Dieser Takt ist mit dem 8 MHz-Takt fest verknüpft. Für sechs Taktzyklen des CLK-Signals ist E auf LOW, dann wieder auf HIGH für vier Taktzyklen (also ein Takt von ca. 0,8 MHz).

VPA — Valid Peripheral Address. Der CPU wird mit einem LOW über diese Leitung mitgeteilt, das ein synchron arbeitender Baustein (z. B. Peripheriebaustein der 68XX-Serie) adressiert wurde. Abschluß mit 2K2-Pullup-Widerstand auf dem Mainboard.

VMA — Valid Memory Address. Die CPU reagiert damit auf die Anforderung (durch LOW an VPA) nach einem synchronen Buszyklus und meldet mit einem LOW über VMA, daß sie nun auf den E-Takt synchronisiert ist. 4K7-Pullup auf Mainboard!

BR — Bus Request. Mit einem LOW wird der CPU hier mitgeteilt, daß ein anderer Baustein die Bussteuerung übernehmen möchte. Auf dem Mainboard ist die Leitung mit einem 4K7-Pullup-Widerstand abgeschlossen. ATARI empfiehlt ebenfalls einen 4K7-Pullup-Abschlußwiderstand auf der Erweiterungskarte.

BGO — Bus Grant Out. Der Einheit auf der Erweiterungskarte, welche die Bussteuerung angefordert hat (mit LOW auf der BR-Leitung) wird mit einem LOW signalisiert, daß sie die Bussteuerung nach Abschluß des laufenden Buszyklusses übernehmen kann.

Dieses Signal wird vom GLUE geliefert und ist mit den anderen Bausteinen des Mega ST (BLITTER und DMA-Einheit), die ebenfalls die Bussteuerung von Zeit zu Zeit übernehmen können, "abgestimmt".

BGACK — Bus Grant ACKnowledge. Mit einem LOW auf dieser Leitung bestätigt der anfordernde Baustein, daß er die Bussteuerung übernommen hat (Buszuteilungsquittung).

Als Open-Collector-Leitung ist dieser Anschluß auf dem Mainboard mit einem 4K7-Pullup-Widerstand abgeschlossen, was ATARI auch für den Abschluß auf der Erweiterungskarte empfiehlt.

NMI, INT 5, INT 3 — Je nachdem, welche dieser drei Leitungen von der Logik der Erweiterungskarte auf LOW gezogen wird, wird ein Interrupt der Priorität 7 (Non Maskable Interrupt), Priorität 5 oder Priorität 3 ausgelöst. Sollten alle drei Interrupts gleichzeitig auftreten, wird natürlich der höchstwertige (Priorität 7) zuerst berücksichtigt. Die periphere Einheit am Megabus muß nun korrekt auf die Interruptbestätigung durch die 68000er-CPU reagieren (siehe auch Teil II, Kapitel 4, Abschnitt "Interrupt-Steuerung").

Stromversorgung der Erweiterungskarte

Die Stromversorgung der Erweiterungskarte erfolgt über eine 6pol. Steckverbindung (siehe Abbildung 1.15). Die +5V können dabei mit bis zu 750 mA und die +12V mit bis zu 500 mA belastet werden.

Speicherraumreservierungen für die Erweiterungskarte

Nach ATARI-Informationen können der Adreßbereich von $C0 0000 ... $CF FFFF, $FF 0000 ... $FF 7FFF und $FF FE00 ... $FF FFFD durch die Erweiterungskarte belegt werden. Für die Benutzung aller weiteren Adreßbereiche behält sich ATARI das Recht auf eine eigene Benutzung vor.

Da für den Megabus inzwischen einige Erweiterungen erhältlich sind, wie z. B. eine Floating-Point-Coprocessor-Karte von ATARI, Einbau-Festplatten mit Controller, Netzwerkkarten und Ansteuerkarten für Großbildschirme, ist der eine oder andere Besitzer eines "kleinen" STs (260ST/520ST+ oder 520ST(M)) nicht abgeneigt, seinem System ebenfalls einen Megabus-Anschluß zu spendieren.

Im ST Magazin, Ausgabe 10/88, ist aufgezeigt, wie unerschrockene Hardware-Bastler ihren "kleinen" ST mit einem Megabus-Anschluß nachrüsten können!

Kapitel 2: Das Grafiksystem

Die Computer der ATARI ST-Serie kennen vom Prinzip her zwei Grafikbetriebsarten:

- Monochrombetrieb
- Colorbetrieb

Nur schwarz/weiß, aber gestochen scharf – Der Monochrombetrieb

Der Monochrombetrieb eignet sich durch seine hohe Grafikauflösung von 640 * 400 Punkten und der schwarz-auf-weiß Darstellung hervorragend für die Textverarbeitung und den Programmierbetrieb. Der von ATARI für den Monochrombetrieb vorgesehene Monitor SM124 / SM125 liefert ein scharfes und flimmerfreies Bild.

Das wird durch eine Bildwiederholfrequenz von ca. 70 Hz erreicht, während diese bei normalen Fernsehern und Monitoren üblicherweise bei 50 Hz (50 Halbbilder/s.) liegt. Der ST stellt 70 Vollbilder/s mit 400 sichtbaren Bildschirmzeilen dar. Dazu muß die Horizontalfrequenz (Zahl der Bildschirmzeilen/s) auf einen Wert von ca. 36 kHz gebracht werden. Der ST arbeitet

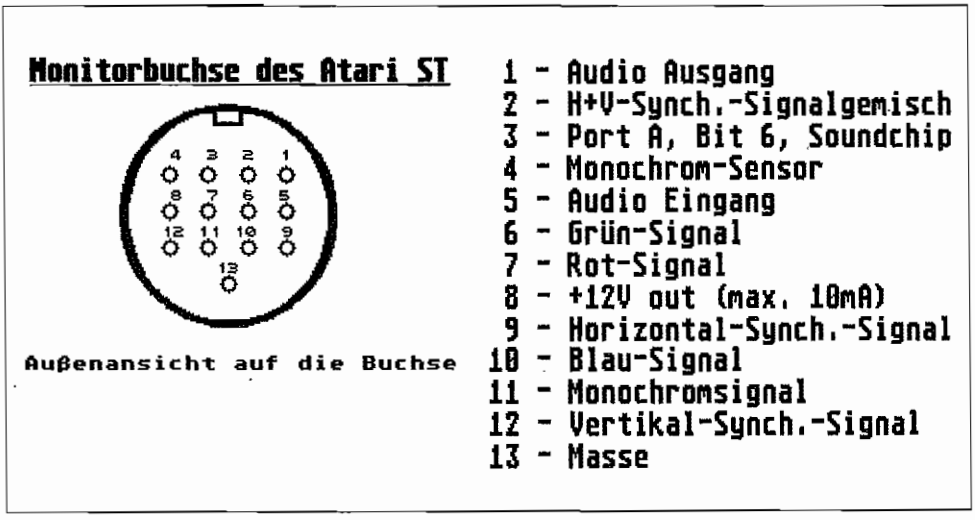


Abb. 2.1: Die Monitorbuchse des ATARI ST

mit einer H-Ablenkfrequenz, die also nahezu 2,5mal so hoch wie bei "normalen" Fernsehern und Monitoren (ca. 15,6 kHz) liegt. Ein "normaler" Monitor oder Fernseher mit Videoeingang ist da klar überfordert und kann evtl. Schaden erleiden, da die Hochspannungserzeugung in Fernsehern und Monitoren üblicherweise vom Horizontaloszillator gesteuert wird.

Angeschlossen wird der ATARI-Monochrommonitor an die 13polige Monitorbuchse des ST.

Die Bildinformation wird dem Monitor bei Monochrombetrieb über den Anschlußpin 11 (Monochrom-Signal) der Monitorbuchse angeboten. Das Signal kennt, ähnlich dem Ausgang eines Digitalbausteins, nur zwei Zustände. Der ST liefert ein Signal von 1Vss an 75 Ohm.

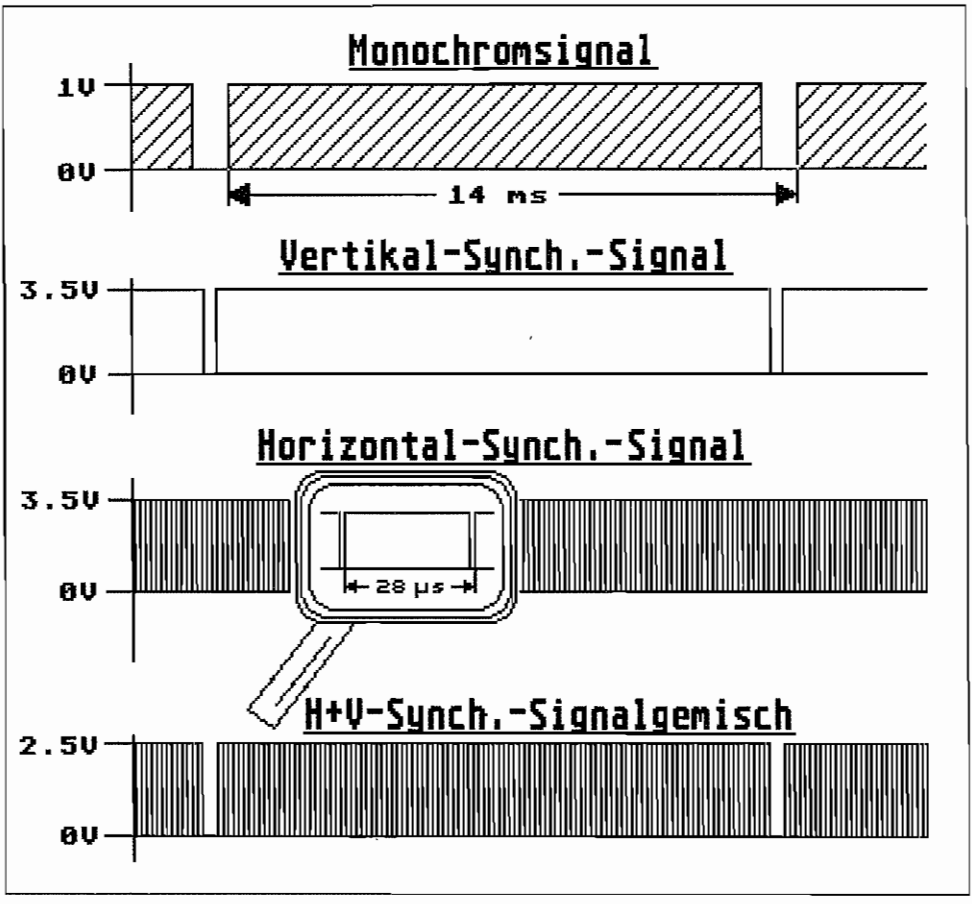


Abb. 2.2: Die Video-Ausgangssignale bei Monochrombetrieb

Ein Spannungswert von +1 V gegen Masse (liegt an Pin 13 der Monitorbuchse) entspricht dabei der Bildinformation "weiß", und 0V bedeutet somit "schwarz". Zwischenwerte (Grauabstufungen) sind nicht möglich.

Die Synchronisationssignale für das entsprechende Strahlrücklauf-Timing im Monitor werden für Horizontal- und Vertikalablenkung sowohl getrennt als auch in Form eines Signalgemischs an der Monitorbuchse angeboten. Sie haben TTL-Pegel und sind Low-aktiv, was bedeutet, daß die Aktion (Elektronenstrahlrücklauf im Monitor zum Zeilen-/Bild-Anfang) ausgelöst wird, wenn das Signal auf Low-Pegel geht.

Der Monitor bestimmt die Betriebsart

Über Pin 4 der Monitorbuchse "erfühlt" der ST, welche Art von Monitor angeschlossen ist. Durch einen Low-Pegel wird "mitgeteilt", daß im hochauflösenden Monochrom-Modus gearbeitet werden soll. Der Anschluß führt im ST auf den I/O-Port, Bit 7, des Multi-Function-Peripheral-Bausteins (MFP) 68901.

Bei der Systeminitialisierung wird durch Abfrage des eben erwähnten Port-Anschlusses 7 des MFP festgestellt, ob im Monochrom- oder Colorbetrieb gearbeitet wird.

Ein Pegelwechsel an diesem Eingang, z. B. durch Abziehen des Monitorsteckers, wird während des Vertikal-Blank-Interrupts festgestellt und führt zur Neuinitialisierung des Systems.

Jetzt wird's bunt – Die Colorbetriebsart

Im Colorbetrieb arbeitet der ST mit "normalen" RGB-Monitoren oder Farbfernsehern mit SCART-Eingang zusammen.

Hierbei wird die Farbinformation für jeden Bildpunkt in entsprechende Intensitätswerte der drei Primärfarben Rot, Grün und Blau zerlegt. Für jede Primärfarbe wird die Bildinformation über eine eigene Leitung dem Monitor/Fernseher zugeführt.

Der ST liefert RGB-Signale von max. 1 Vss an 75 Ohm, denen ein Gleichspannungspegel von ca. 0,5 Volt unterlegt ist, wobei ein Spannungswert von +1,5 Volt gegen Masse der höchsten Farbintensität entspricht. Ein Wert von 0,5 Volt (= unterlegter Gleichspannungspegel) bedeutet somit niedrigste Farbintensität.

Die Synchronisationssignale sind auch in dieser Betriebsart wieder getrennt von den Signalen für die Bildinformation geführt. Sie stehen sowohl getrennt nach H- und V-Signal als auch als

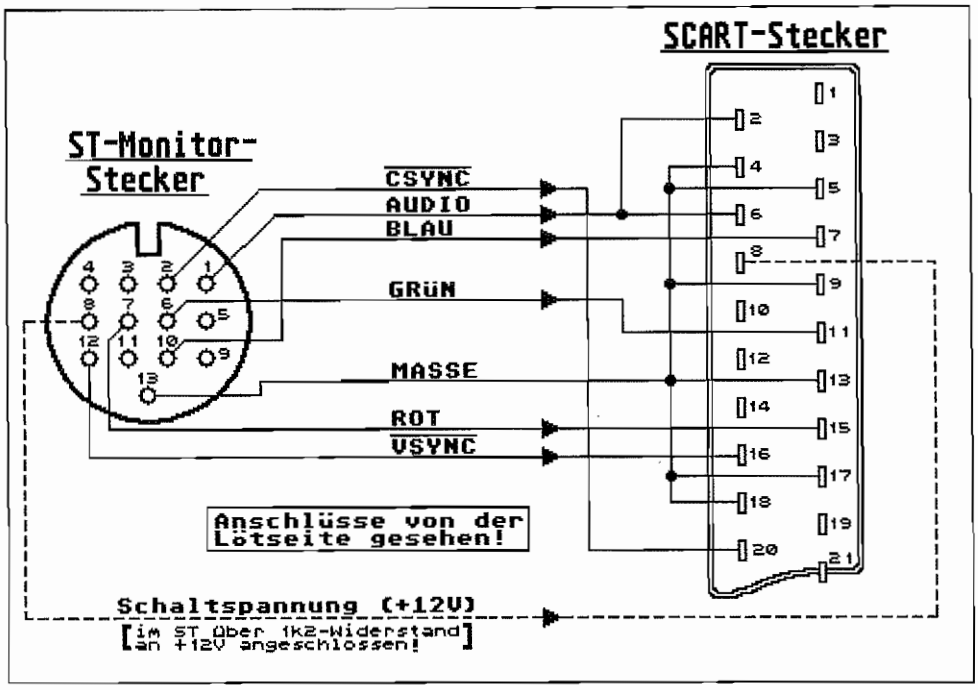


Abb. 2.3: Anschluß des ST an einen Farbfernseher mit SCART-Eingang

Synchronsignalgemisch an Pin 2 der Monitorbuchse zur Verfügung. (Achtung! Das Synchronsignalgemisch steht wegen des eingebauten Modulators bei älteren 520STM-Geräten nicht zur Verfügung!)

Durch die getrennte Zuführung von Rot-, Grün- und Blau-Signalen erhält man auch auf einem handelsüblichen Farbfernseher mit SCART-Eingangsbuchse ein besseres Farbbild als von Computern mit einem sogenanntem FBAS-Ausgang.

Ein Farb-Bild-Austast und Synchronsignal-Ausgang (FBAS-Ausgang) hat zwar den Vorteil, daß man nur eine Leitung (nun ja, die Masseleitung ist natürlich ebenfalls noch erforderlich) zwischen Computer und Monitor benötigt.

Alle Informationen wie Farbe, Farbintensität, Synchronisier- und Austastinformation (zur Dunkeltastung des Elektronenstrahls in der Bildröhre, während dieser vom Ende einer Bildschirmzeile zurück zum Anfang einer neuen Zeile springt) sind in einem Signal gewissermaßen verschlüsselt enthalten.

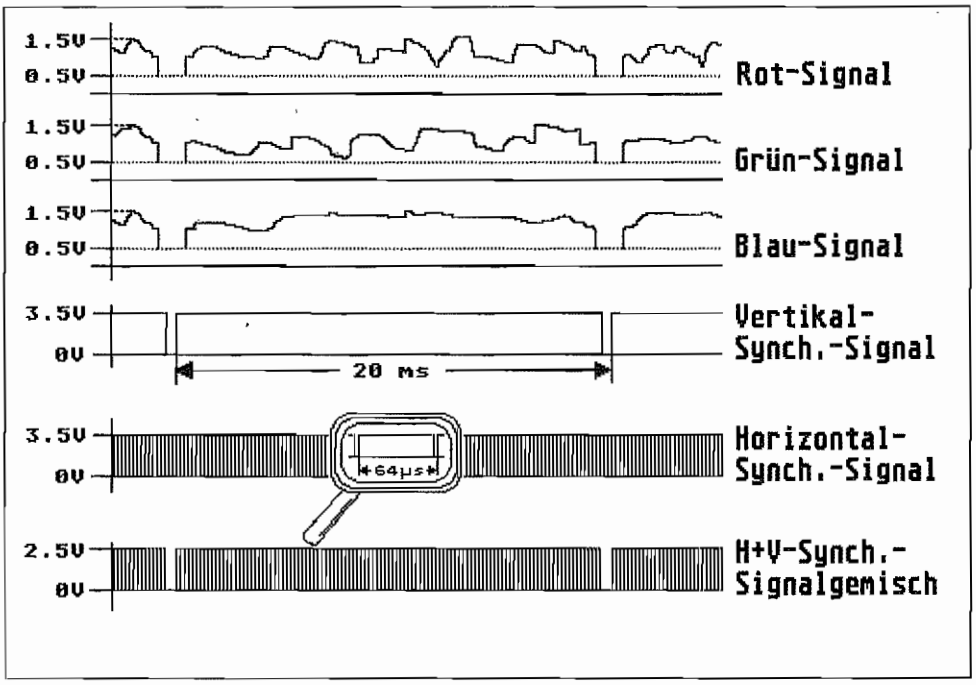


Abb. 2.4: Die Video-Ausgangssignale bei RGB-Betrieb

Sie müssen jedoch (aha! Jetzt kommt der Haken!) im Monitor erst wieder entschlüsselt werden. Dabei erleidet die eigentliche Bildinformation einige Qualitätseinbußen durch Dekodier- und Verzögerungsschaltungen.

Das bessere Farbbild – mit RGB-Monitor

Bei einem RGB-Anschluß werden die Bildinformationen dagegen schon direkt in ihre Primärfarben zerlegt angeboten, wie es der Monitor/Fernseher im Endeffekt ja eigentlich benötigt. Diese Signale erfahren im Monitor/Fernseher nur noch eine Pegelanpassung und gelangen dann direkt an die Bildröhre, welche für jede Primärfarbe ein eigenes Strahlerzeugungssystem besitzt. Die RGB-Bildinformation wird also kaum beeinflußt und verfälscht.

Man kann übrigens mit relativ geringem Aufwand einen handelsüblichen Monochrom-Monitor dazu "bewegen", als Sichtgerät für den Colorbetrieb zu arbeiten. Die einzelnen Farb-

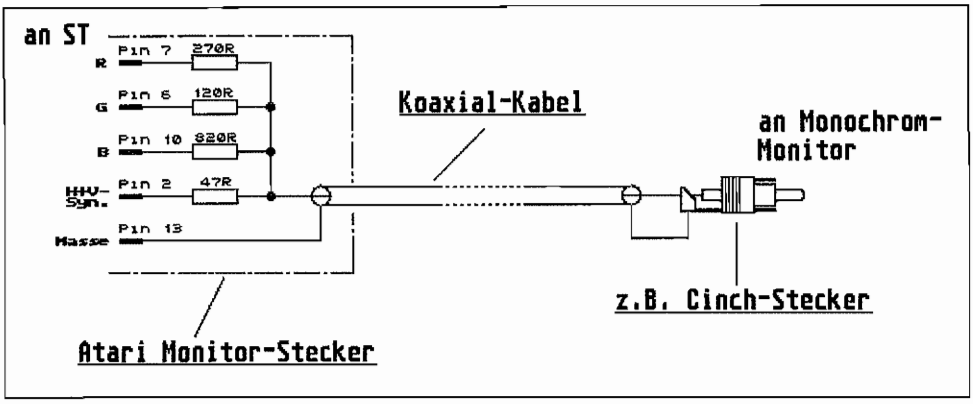


Abb. 2.5: So einfach läßt sich ein normaler Monochrom-Monitor als Sichtgerät für die Colorbetriebsart des ST anschalten

informationen und Farbintensitätswerte müssen nur in entsprechende Grauwerte umgesetzt werden.

Außer den Anschlußelementen an Computer und Monochrom-Monitor sowie Koaxialkabel entsprechender Länge benötigt man nur noch vier Widerstände, einen Lötkolben, Seitenschneider und ein wenig bastlerisches Geschick.

Soviel zur Video-Anschlußtechnik bei den ST-Computern. Erwähnt sei hier noch, daß der ATARI 520ST(F)M/1040STFM mit einem eingebauten Modulator ausgerüstet ist. Damit ist es möglich, einen Farbfernseher als Sichtgerät für Colorbetrieb zu benutzen. Aufgrund der hierbei auftretenden Qualitätsverluste des Bildinformationssignals durch Modulation im ST und Demodulation im Farbfernseher ist die Bildqualität natürlich nicht mit der eines RGB-Monitors zu vergleichen. Ein längeres Arbeiten mit einem über den Modulator angeschlossenen Farbfernseher ist insbesondere bei Textdarstellung nicht empfehlenswert.

Organisation des Bildschirmspeichers

Die Computer der ST-Serie besitzen keinen separaten Bildschirmspeicher wie z. B. viele "professionelle" Personal-Computer.

Vielmehr wird ein Teil des normalen RAMs dafür benutzt. Im Prinzip ist die Lage des Bildschirm-RAMs im Speicherraum des ST beliebig einstellbar, also nicht auf einen bestimmten Adreßbereich beschränkt.

Der Video-Hardware muß nur "gesagt" werden, ab welcher Adresse im RAM der Bildschirmspeicher beginnt. Dann werden die ab dieser Adresse folgenden 32000 Bytes im RAM vom Video-Controller als Bildschirminformation interpretiert und entsprechend umgesetzt.

Je nach Wahl der gewünschten Bildschirmauflösung werden die Daten im Bildschirmspeicher vom Video-Controller in ein Videosignal umgewandelt.

High resolution – Hohe Auflösung

Diese Darstellungsart ist nur in Verbindung mit dem hochauflösenden Monitor möglich. Es werden 640 Bildpunkte in 400 Zeilen und dabei mit zwei Farben dargestellt. Ein Bildschirmpunkt (Pixel = umgangssprachliche Kurzform von Picture cell = zu deutsch: Bildelement) kann hierbei nur zwei Zustände annehmen, entweder hell oder dunkel. Entsprechend der Abbildung 2.6 wird jedes helle Pixel durch ein gelöschtes Bit und ein dunkles Pixel durch ein gesetztes Bit im Bildschirmspeicher dargestellt.

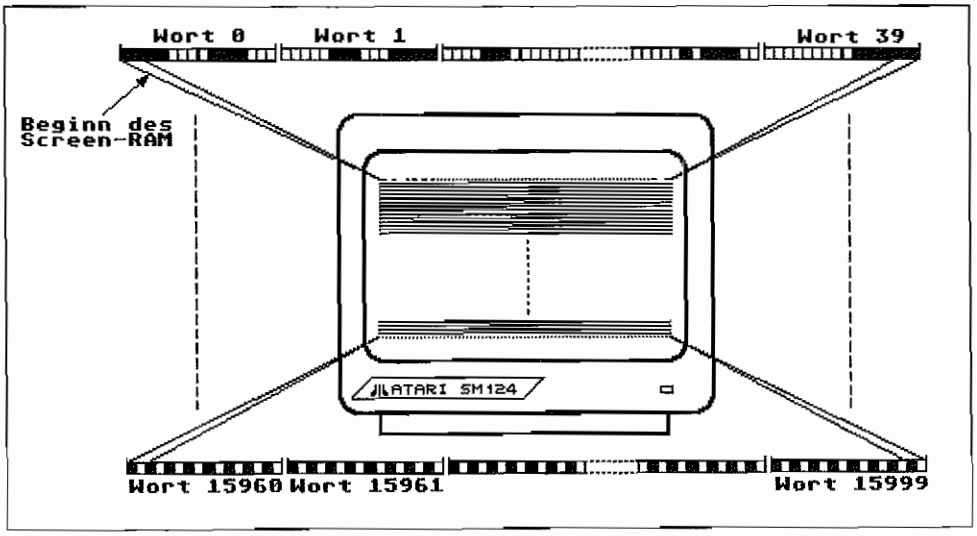


Abb. 2.6: Organisation des Bildschirmspeichers für die High-Resolution-Darstellung

Medium resolution – Mittlere Auflösung

Auf dem hochauflösenden Monitor ist diese Darstellungsart nicht möglich (zumindest nicht ohne kräftige bastlerische Klimmzüge wie z. B. in der "ST COMPUTER" 5/88 beschrieben).

Es handelt sich hierbei um eine Colorbetriebsart, die einen RGB-Monitor/Farbfernseher mit SCART-Eingang oder einen "normalen" Monochrommonitor/Fernseher mit entsprechendem Anschlußkabel erfordert. Die Bildschirmdarstellung umfaßt hierbei 200 Zeilen zu je 640 Pixel. Jedes Pixel kann hierbei vier Zustände, also vier Farben annehmen.

Der Bildschirmspeicher wird vom Video-Controller dabei als zwei sogenannte Bit-Planes (Bit-Ebenen) interpretiert, wobei die Kombination der jeweils "übereinanderliegenden" Bits der beiden Planes bestimmt, welche Farbe der zugehörige Bildschirmpunkt auf dem Monitor dann bekommen soll (siehe Darstellung in der Abbildung 2.7).

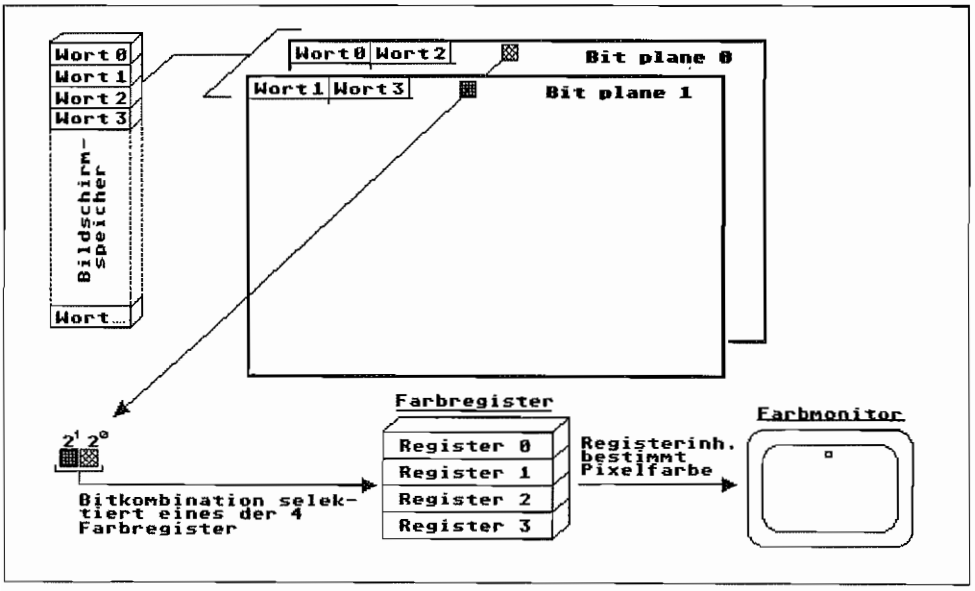


Abb. 2.7: Organisation des Bildschirmspeichers für die Medium-Resolution-Darstellung

Low resolution – Niedrige Auflösung

Auch hierbei handelt es sich um eine Colorbetriebsart. Es werden 320 Pixel in 200 Zeilen auf dem Bildschirm dargestellt.

Der Bildschirmspeicher wird jetzt in vier Bit-Planes organisiert. Ein Pixel wird also durch vier Bits beschrieben. Somit kann die Pixelfarbe aus einem der 16 zur Verfügung stehenden Farbregister ausgewählt werden.

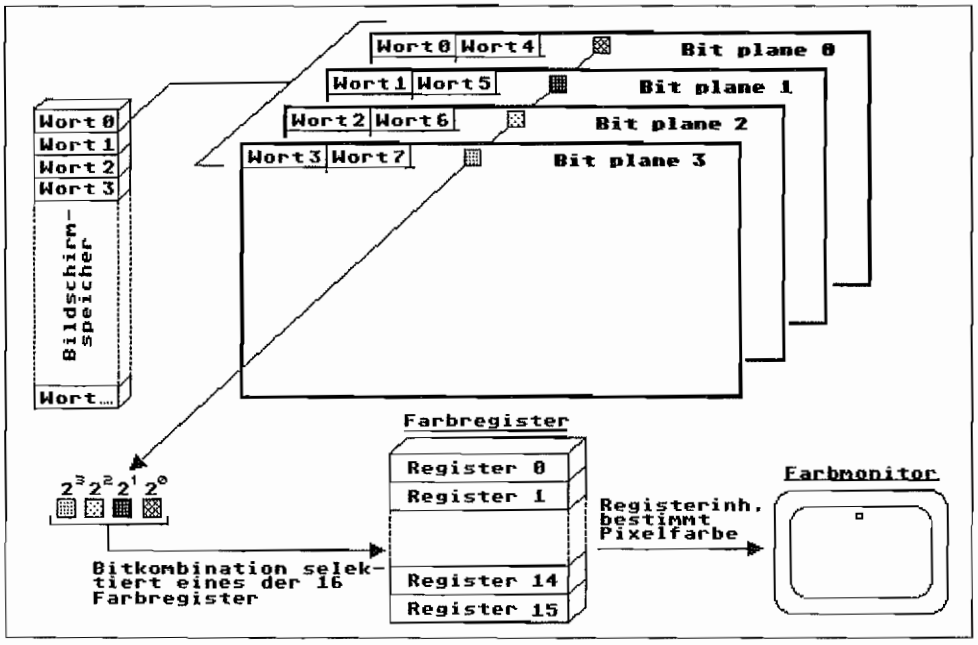


Abb. 2.8: Organisation des Bildschirmspeichers bei Low-Resolution

Indirekte Farbgebung

ATARI benutzt auch hier, wie schon bei den 8-Bit-Modellen 800XL/130XE, das Prinzip der Color indirection, d. h., die Bitkombination für ein Pixel legt noch nicht direkt eine bestimmte Farbe für den Bildschirmpunkt fest, sondern dient dem Video-Controller lediglich zur Auswahl eines Farbregisters. Dieses Farbregister enthält dann erst den entsprechenden Farbwert für das Pixel. Der ST besitzt insgesamt 16 solcher Farbregister, auch Palette-Register genannt, die mit unterschiedlichen Farbinformationen geladen werden können. Pro Pixel stehen bei Medium Resolution also zwei Bits zur Verfügung, die entsprechend ihrer Wertigkeit im Video-Controller eines von vier Palette-Registern 0...3 adressieren können. In Low Resolution kann mit vier Bits pro Pixel eines von 16 Farbregistern angewählt werden.

Der ATARI-Videocontroller

ATARI verwendet auch in den 16 Bit-Computern der ST-Serie keinen Standard-Video-Controller. Es handelt sich hierbei um eine Eigenentwicklung mit speziell auf den ST hin

ausgelegten Eigenschaften. Der Video-Controller läßt sich ebensowenig wie die anderen Systemkomponenten DMA-Einheit, Speicherverwaltung usw. in einem ganz bestimmten Chip auf der Platine des ST wiederfinden. Vielmehr bilden erst die Kombination von CPU und vier ATARI-spezifischen Chips, nämlich MMU, DMA, Shifter und GLUE, ein funktionsfähiges System.

Wenn also hier von dem ST-Video-Controller die Rede ist, so sind damit alle Register und Steuereinheiten in den vier ATARI Custom-Chips gemeint, die zusammen die Funktion des Video-Controllers erfüllen! Wesentliche Aufgaben bei der Erzeugung des Videosignals übernimmt hierbei der Shifter-Chip.

Für den Programmierer – auch den hardwarenahen Systemprogrammierer – ist es jedoch letztlich egal, in welchem Chip sich welches Register befindet (Hauptsache, die Adresse ist bekannt). Die Arbeitsweise des Video-Controllers soll das in Abbildung 2.9 aufgeführte Prinzipschaltbild verdeutlichen.

So werden Bits zu Pixel

Die Schieberegister im Video-Controller haben Wortbreite und werden zyklisch mit den Daten aus dem Bildschirmspeicher geladen. CPU-Zugriffe und Zugriffe des Videocontrollers (Shifters) sind dabei zeitlich ineinander verzahnt. Die Information wird dann bitweise im Pixel-Takt an die Register-Select-Logik "hinausgeschoben". Diese bestimmt aufgrund der Ausgangsbitkombination der Schieberegister und der eingestellten Grafikbetriebsart im Shift-Mode-Register das für das Pixel maßgebende Farbregister.

Die Grafikbetriebsart wird mit dem Shift-Mode-Register nach dem in Abbildung 2.10 dargestellten Schema eingestellt. Ist von der Register-Select-Logik das Farbregister ausgewählt worden, wird aus diesem die Information für Farbe und Farbintensität an das Output-Register ausgegeben. Die Farbintensität, mit der jede der drei Primärfarben im Ausgangssignal vertreten ist, ist in acht Stufen einstellbar. Für jede Farbe existieren drei digitale Ausgangsanschlüsse, also sind $2^3 = 8$ Kombinationen = Farbintensitäten möglich. So läßt sich durch entsprechende Bitkombination in jedem Farbregister einer von $2^{(3+3+3)} = 2^9 = 512$ Farbtönen anwählen.

Die Information aus dem für den Bildpunkt angewählten Farbregister gelangt also ins Output-Register und wird an dessen Ausgängen mit einem aus jeweils drei Widerständen gebildeten Digital/Analog-Wandler in eine analoge Spannung umgesetzt.

Dazu wird noch das Blanking-Signal gemischt, und dann geht es über Impedanzwandler ab an die Monitorbuchse des ST.

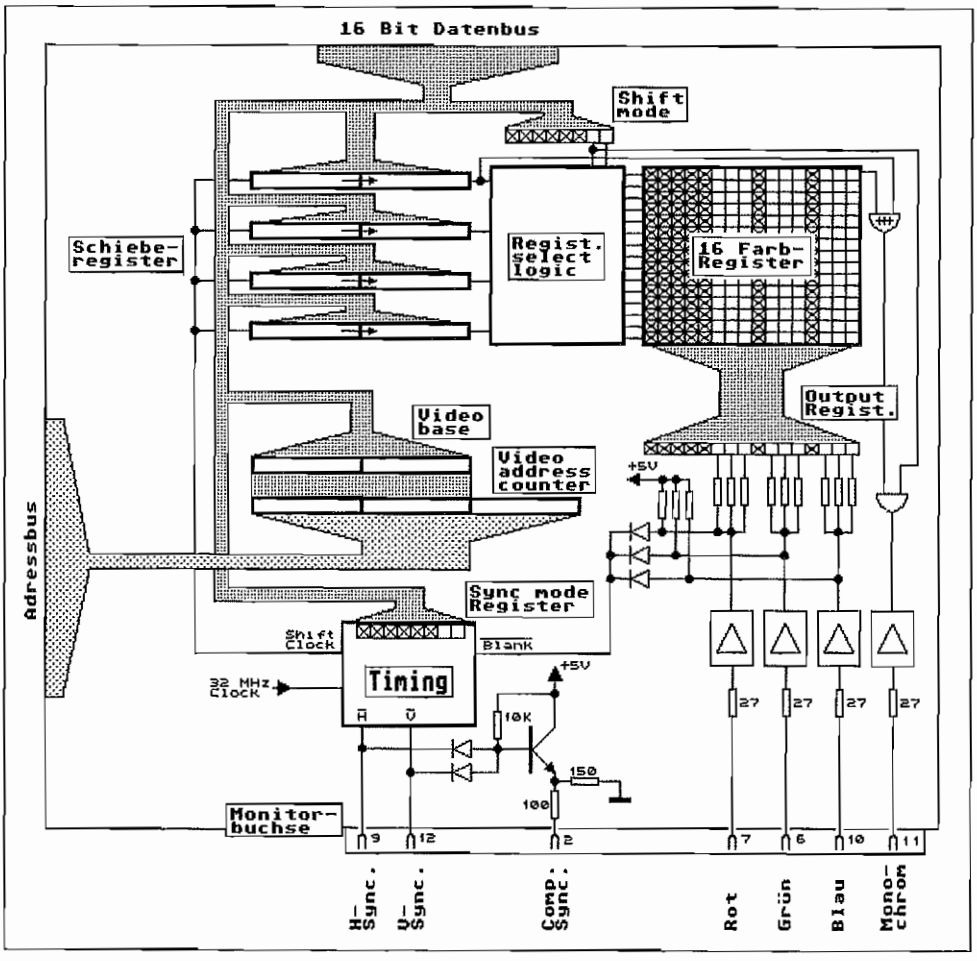


Abb. 2.9: Prinzipschaltbild des ST-Video-Controllers

In der Monochrom-Betriebsart kann lediglich über das niedrigstwertige Bit des Farbregisters 0 eingestellt werden, ob der Bildpunkt normal oder invertiert ausgegeben werden soll, was eine "normale" oder "invertierte" Bildschirmdarstellung zur Folge hat.

Für einen reibungslosen Ablauf dieser Vorgänge sorgt die Timing-Logik, welche aus dem 32-MHz-Takt des ST-Systems alle für den Video-Controller erforderlichen Timing-Signale generiert. Der 32-MHz-Mastertakt gelangt vom Oszillator in den Shifter und wird dort zur Pixel-

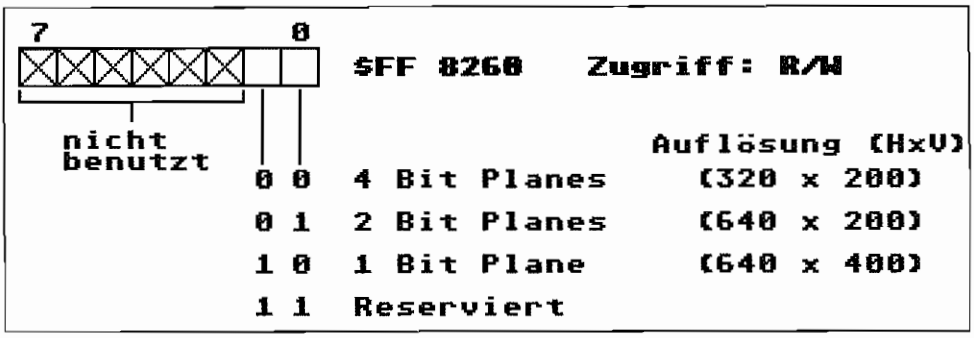


Abb. 2.10: Das Shift-Mode-Register

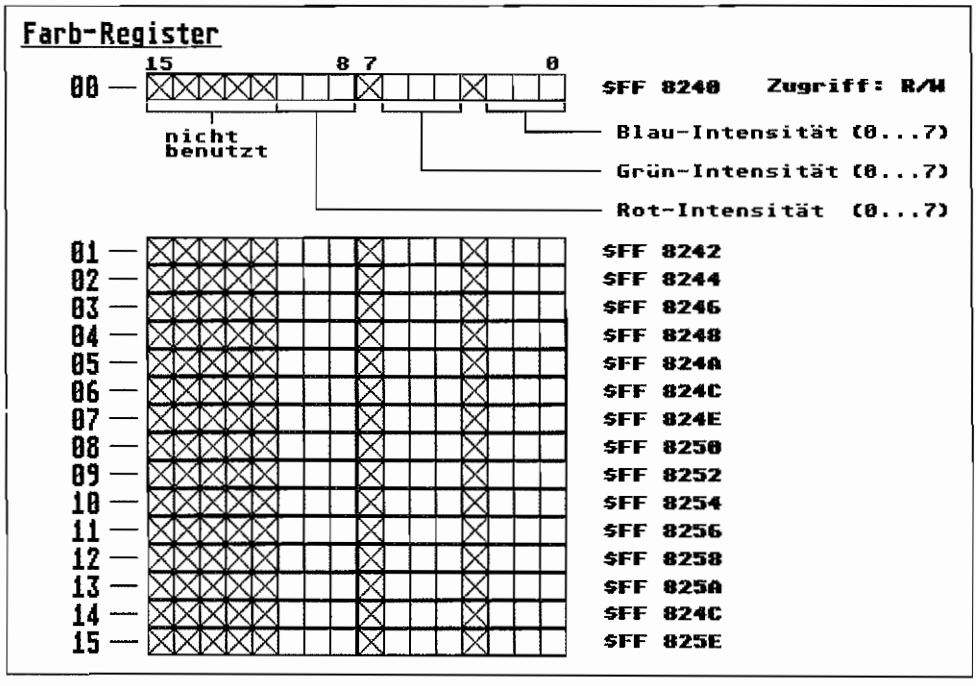


Abb. 2.11: Die Farbregister

ausgabe benutzt. Ein aus den 32 MHz im Shifter abgeleiteter 16-MHz-Takt wird zur MMU geführt, die daraus Horizontal- und Vertikal-Synchronimpulse erzeugt und außerdem den 8-MHz-Takt für u. a. die CPU erzeugt.

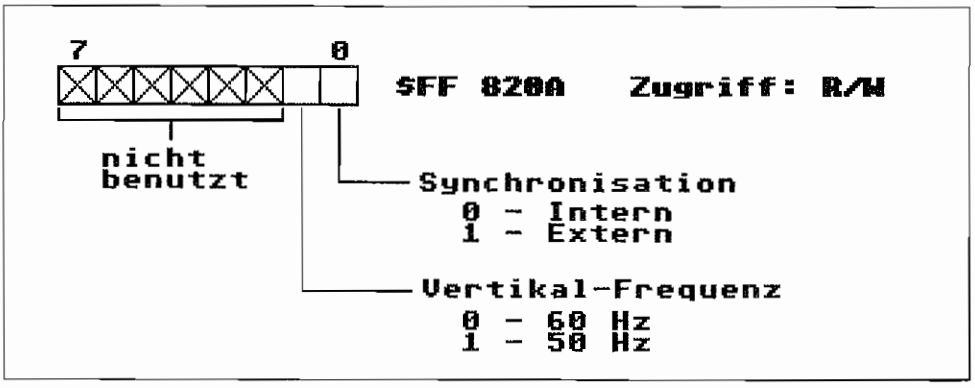


Abb. 2.12: Die Bitbelegung des Sync-Mode-Registers

Mit dem Sync-Mode-Register läßt sich unter anderem einstellen, mit welcher Vertikal-Synchronisationsfrequenz gearbeitet wird. Das Bit 0 verdient dabei noch besondere Beachtung. Ist es gelöscht, werden die Impulse für die Horizontal- und Vertikalsynchronisation von der Timing-Logik des Video-Controllers selbst erzeugt. Bei gesetztem Bit 0 erwartet die Timing-Logik jedoch die H- und V-Synch.-Impulse (Low-aktiv) von einer externen Signalquelle. Eingespeist werden diese an den H- und V-Sync.-Anschlüssen der Monitorbuchse. Die H- und V-Sync.-Anschlüsse des ST-Video-Controllers sind dann als Eingänge geschaltet!

So ließe sich der ST also mit externen Videosignalquellen wie z. B. Videokamera oder Videorecorder synchronisieren. Leider ist damit noch keine pixelgenaue Synchronisation möglich, und es kommt zu unschönem "Zeilenreißen". Von einer Zeile zur nächsten ergeben sich Sprünge und somit ein verzerrtes Mischbild. Um diese Verzerrungen auszumerzen, muß man (im STE endlich realisiert) den gesamten ST mit einem externen Takt versorgen.

Der Video-Controller ist im Prinzip ein System mit "Eigenleben", das sowohl Zugriff auf den Adreß- als auch den Datenbus (nur der RAMs!) hat, ohne den Umweg über CPU-Register nehmen zu müssen. Einige Daten muß die CPU dem Video-Controller aber schon mitteilen, damit dieser z.B. weiß, wo die Bildschirmdaten im Speicher stehen. Den Anfang des Bildschirmspeichers erfährt der Video-Controller aus dem

Video-Base-Register

Adresse	Zugriff	Funktion	Label
$FF 8201	R/W	Video-Base-Register High-Byte	"dbaseh"
$FF 8203	R/W	Video-Base-Register Mid-Byte	"dbasel"

Das Video-Base-Register besteht eigentlich aus zwei einzelnen Registern zu je acht Bit, beginnt an einer ungeraden Adresse und kann deshalb nicht mit einem Word-Zugriff angesprochen werden. Der Anfang des Bildschirmspeichers kann im Hauptspeicher des ST auch immer nur an einer 256-Byte Schwelle (an einer Page-Grenze) liegen, da ja für das Low-Byte der vollständigen Adresse das Register fehlt. Ein Zugriff auf das Register ist nur im Supervisor-Modus möglich!

Das folgende Register dient dem Video-Controller als Adreßzähler für den Bildschirmspeicher. Die CPU kann den Inhalt des Video-Adress-Counters nur auslesen (Supervisor-Mode!).

Video-Adress-Counter

Adresse	Zugriff	Funktion	Label
$FF 8205	R	Video-Adress-Counter High-Byte	"vcounthi"
$FF 8207	R	Video-Adress-Counter Mid-Byte	"vcountmid"
$FF 8209	R	Video-Adress-Counter Low-Byte	"vcountlow"

Gesetzt wird der Video-Adress-Counter über das Video-Base-Register. Dieser Counter übernimmt zu Beginn eines jeden Bildes die Anfangsadresse des Bildschirmspeichers aus dem Video-Base-Register.

Erwünschte Unterbrechungen?

Die H- und V-Synchronisationssignale vom Video-Controller werden zusätzlich im ST noch zur Interruptsteuerung benutzt. Der Vertical-Blank-Interrupt wird ausgelöst, sobald die letzte Bildschirmzeile eines Bildes geschrieben ist. Im Monitor muß der Elektronenstrahl, der die Pixel auf der Phosphorschicht des Bildschirms zum Leuchten bringt, ausgeschaltet werden. So wird verhindert, daß der Betrachter sieht, wie der Elektronenstrahl vom Strahl-Ablenksystem wieder an den Anfang der ersten Bildschirmzeile des neuen Bildes positioniert wird (also von rechts unten nach links oben springt).

Während dieser Zeitspanne lassen sich einige Programmroutinen abwickeln, die zyklisch irgendwelche Aktionen durchführen. So wird z. B. im Vertical-Blank abgefragt, ob bei den Floppys der Motor noch läuft. Steht dieser, wird die entsprechende Floppy deselektiert. Vom Betriebssystem ist die Möglichkeit für den Programmierer vorgesehen, eigene VBlank-Interruptroutinen einzufügen (siehe auch Teil I, Kapitel 1, "Der Vertical Blank Handler").

Es ist möglich, auch beim Strahlrücklauf vom Ende einer Bildschirmzeile zum Anfang einer neuen Bildschirmzeile (Horizontal-Blanking = Horizontal-Strahlaustastung) einen Interrupt

auszulösen. Im normal arbeitenden ST-System ist dieser HBlank-Interrupt jedoch nicht vorgesehen. Ein laufender Prozeß würde nämlich in der Monochrom-Betriebsart sonst alle 28 µSek für mindestens sieben µSek unterbrochen. Solange dauert es nämlich, bis der Prozessor den Interrupt erkannt, quittiert und abgeschlossen hat. Dazu kommt natürlich noch die Zeit für die eigentliche Interrupt-Routine, die ausgeführt werden soll. Der HBlank-Interrupt würde somit schon mindestens 25% der Prozessorzeit beanspruchen; ein schöner Bremsklotz also.

Jede Menge Manipulationen möglich

Die Programmierung des ST-Video-Controllers wird dem Programmierer durch eine Reihe von XBIOS-Aufrufen erleichtert. Es sind dies die XBIOS-Funktionen #2 ("Physbase"), #3 ("Logbase"), #4 ("Getrez"), #5 ("Setscreen"), #6 ("Setpallette"), #7 ("Setcolor") und #37 ("Vsync"). Nähere Einzelheiten dazu im Teil I, Kapitel 1, in der "XBIOS-Referenz".

Die komfortablen Möglichkeiten des ST, Text und Grafiken gleichermaßen gut darzustellen, liegen darin begründet, daß es keinen speziellen Character- und Grafik-Modus gibt.

In herkömmlichen Systemen hat man üblicherweise einen sogenannten Charactergenerator (Zeichengenerator), der in Zusammenarbeit mit einem Video-Controller die Darstellung von Textzeichen auf dem Bildschirm ermöglicht. Die Zeichen können so nur in einem relativ groben Raster auf dem Bildschirm angeordnet werden (Zeichenmodus). Ein zusätzlicher Grafikmodus ermöglich das Setzen und Rücksetzen von einzelnen Bildpunkten.

Beim ST gibt es nur den Grafikmodus. Alle Zeichen und Buchstaben werden durch einzelnes Setzen und Rücksetzen von Pixeln gebildet. Das beansprucht natürlich einiges an Prozessorzeit und Speicherplatz, gestattet dafür aber das komfortable Mischen von Grafik und Text, da Textzeichen für den ST nichts anderes als Gruppen von Grafikpunkten darstellen. Außerdem lassen sich so Zeichen im Pixelraster beliebig auf dem Bildschirm plazieren (z. B. Textverarbeitung "Signum!"). Proportionalschrift ist somit auch möglich!

Der Blitter

Zur Entlastung der CPU und Beschleunigung der Graphikausgabe wurde von ATARI schon bei der Entwicklung der ST-Computerserie ein Hilfsbaustein vorgesehen, der wesentliche Aufgaben bei der Aktualisierung des Bildschirmspeichers übernehmen sollte. Zunächst wurden die Geräte der MEGA ST-Serie mit dem Bit-Block-Transfer-Processor (Blitter) ausgestattet. Der Blitter übernimmt dabei die Aufgaben, welche bisher von der BitBlt-Funktion der Line-A-Routine #$A007 ausgeführt wurden. Es handelt sich hierbei also um eine hardwaremäßige Realisierung des BitBlt-Algorithmus.

Bei einem Bit-Block-Transfer können Bitfelder-Daten von einer Ausgangsposition im Bildschirmspeicher an eine Zielposition übertragen werden.

Dabei ist es möglich, die Ursprungsdaten über eine wählbare logische Verknüpfung mit dem bereits im Zielspeicherbereich vorhandenen Bitmuster zu verküpfen. Es ist aber ebenfalls möglich, einen Zielbereich mit einem vorher definiertem Muster zu füllen und so beispielsweise Bildschirmausschnitte zu löschen oder vorzubesetzen.

Unter Zuhilfenahme der BitBlt-Funktion lassen sich so relativ einfach Funktionen wie Text-Scrolling, Textumsetzungen in Fettschrift, Italic oder Outline realisieren. Funktionen wie Window-Updating und -Verschiebung profitieren ebenfalls von den Möglichkeiten, welche die BitBlt-Funktion bietet.

Der Atari ST-Blitter

Der Grundalgorithmus der BitBlt-Funktion wurde erstmalig durch Newman & Sproull im Jahre 1979 bei der Beschreibung der RasterOp-Funktion, im Buch "Principles of interactive computer graphics" definiert. Diese Definition sah einen "Bit für Bit"-Block-Transfer vor. Es waren nur wenige logische Verknüpfungsmöglichkeiten zwischen Quell- und Zielbitfeldern vorgesehen.

Im Laufe der Zeit wurden die Grundfunktionen um weitere Verknüpfungsmöglichkeiten erweitert und die Parallelverarbeitung von Bits implementiert.

Außerdem wurde vorgesehen, die Ursprungsdaten mit einem frei definierbaren Grundmuster "vorzuverarbeiten" (sogen. Halftone Pattern) und erst das daraus entstehende Ergebnis mit den Zieldaten zu verknüpfen. Ein nach diesen Definitionen erstellter RasterOp-Chip (die "VL16160 RasterOp Graphics/Boolean Operation ALU" von VLSI-Technology) konnte jedoch immer nur Einzeldaten bearbeiten. Die Quell-, Halftone- und Zieldaten mußten laufend von der CPU in den RasterOp-Chip nachgeladen werden. DMA-Betrieb war nicht möglich.

Der ATARI ST-Blitter ist ein eigenständiger Prozessor, der alle Definitionen der BitBlt-Funktion erfüllt. Er verarbeitet die Daten wortweise (also maximal 16 Bits auf einmal) und braucht von der CPU nicht für jedes Datenwort mit den Quell-, Halftone- und Zieldaten nachgeladen zu werden.

Die erforderlichen Quell- und Zieldaten holt er sich als DMA-Baustein selbst aus dem RAM und legt sie nach Verarbeitung auch wieder dort ab. Außerdem verfügt er über ein eigenes schnelles Halftone-RAM. Zugriffe auf das RAM des ST muß der Blitter mit der CPU und der DMA-Einheit "absprechen".

Bit-Block Verarbeitung

Beim Bit-Block-Transfer werden Bitfeld-Daten aus einem Ursprungs-Speicherbereich mit den Daten eines Bitfelds im Ziel-Speicherbereich unter Verwendung eines Booleschen Operators verknüpft.

Die Verknüpfung wird dabei jeweils auf die zueinandergehörenden Bits des Quell- und Zielbitfelds angewendet.

Folgende logische Verknüpfungsmöglichkeiten ergeben sich dabei:

Verknüpfgs.-Nr.	Ergebnis der Verknüpfung im Zielbitfeld
0	Zielbit = 0
1	Zielbit = Quellbit AND Zielbit
2	Zielbit = Quellbit AND NOT Zielbit
3	Zielbit = Quellbit
4	Zielbit = NOT Quellbit AND Zielbit
5	Zielbit = Zielbit
6	Zielbit = Quellbit XOR Zielbit
7	Zielbit = Quellbit OR Zielbit
8	Zielbit = NOT Quellbit AND NOT Zielbit
9	Zielbit = NOT Quellbit XOR Zielbit
10	Zielbit = NOT Zielbit
11	Zielbit = Quellbit OR NOT Zielbit
12	Zielbit = NOT Quellbit
13	Zielbit = NOT Quellbit OR Zielbit
14	Zielbit = NOT Quellbit OR NOT Zielbit
15	Zielbit = 1

Bit-Block-Transfer mit dem Blitter

Der ST-Blitter verfügt über eine Anzahl von Registern, die an den in der Abbildung 2.13 aufgeführten Adressen im Speicherraum des ST zu finden sind (sofern der Blitter eingebaut ist!).

Blitter-Registerzugriffe müssen jedoch immer im Supervisor-Modus erfolgen, sonst "wirft" der ST Bomben!

Zur Erläuterung des Ablaufes eines Bit-Block-Transfers mit dem Blitter soll das in Abbildung 2.14 gezeigte Datenflußbild dienen.

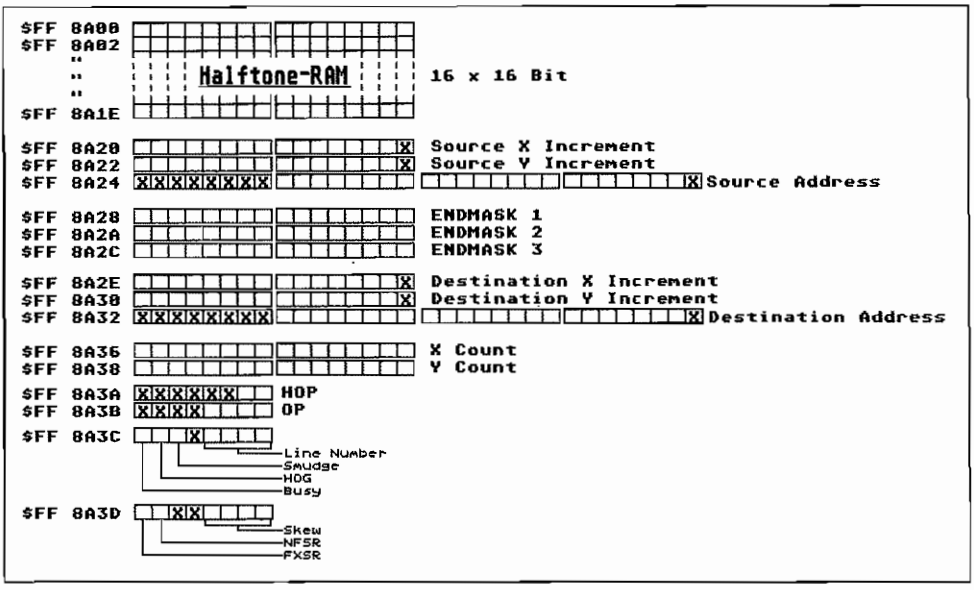


Abb. 2.13: Die Register des Blitter-Chips

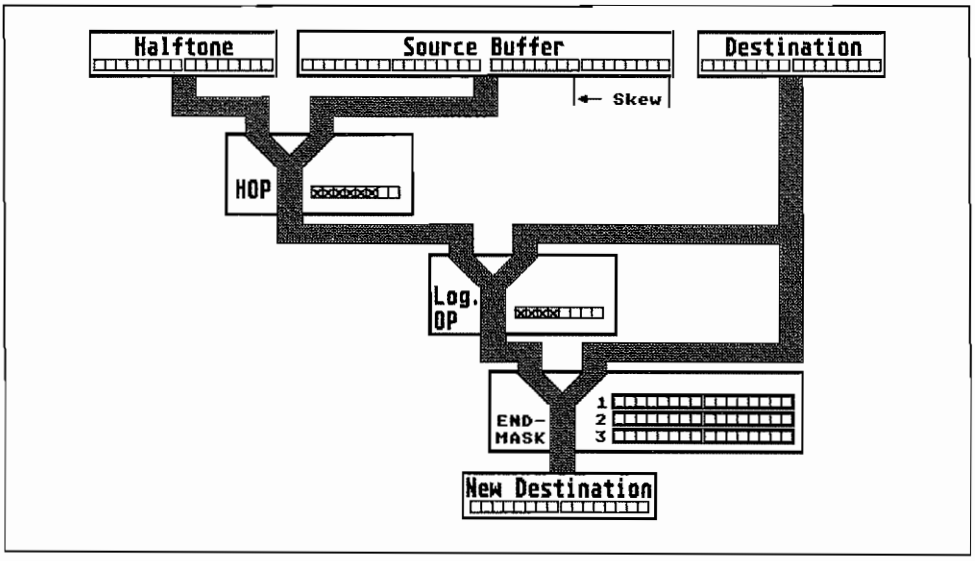


Abb. 2.14: Der Datenfluß beim Bit-Block-Transfer

Nun zur Arbeitsweise des Blitters. Betrachten wir zunächst eine einfache Blitter-Operation: Ein Bit-Block soll vollständig mit 0- oder 1-Bits gefüllt werden.

Zur Wahl der Verknüpfung muß das

Operation-Register

Adresse	Zugriff	Funktion	Größe	Label
$FF 8A3B	R/W	Verknüpfungsart festlegen	Byte	OP

im Blitter mit der entsprechenden Verknüpfungs-Nummer geladen werden (Beispiel: Füllen des Zielbitfeldes mit 0-Bits = OP-Nr. 0, Füllen mit 1-Bits = OP-Nr. 15). Daten aus dem Quellbitfeld und Halftone-RAM sind in diesem Fall nicht erforderlich (der Source-Buffer und das Halftone-RAM werden also nicht benötigt). Der für die Durchführung der logischen Verknüpfung zuständige Funktionsblock (im Datenflußbild mit Log. OP bezeichnet) erzeugt die nötigen 0- bzw. 1-Bits für die Fülloperation.

Bei jeder Blitter-Operation wird der Funktionsblock ENDMASK gebraucht. Hier muß nämlich festgelegt werden, welche Bits im Zieldatenwort geändert werden müssen und welche nicht. Wenn alle Bits eines Wortes im Zielbitfeld mit 0- oder 1-Bits gefüllt werden, schreibt der Blitter einfach ein Datenwort nach dem anderen, ohne jemals Daten aus dem Zielbitfeld lesen zu müssen. Sind dagegen jedoch nicht alle Bits eines Wortes im Zielbitfeld zu ändern, so geben die ENDMASK-Register die Bitpositionen an, die im Zieldatenwort geändert werden müssen.

ENDMASK1-Register

Adresse	Zugriff	Funktion	Größe	Label
$FF 8A28	R/W	Zu ändernde Zielbits (Zeilenanfang)	Word	Endmask1

Dieses Register legt fest, welche Bits im ersten Datenwort einer Zeile des Zielbitfeldes geändert werden sollen.

ENDMASK2-Register

Adresse	Zugriff	Funktion	Größe	Label
$FF 8A2A	R/W	Zu ändernde Zielbits	Word	Endmask2

Wird benutzt, wenn keines der anderen beiden ENDMASK-Register angewendet wird (das ist für alle Datenworte zwischen dem ersten und letzten Datenwort einer Zeile des Zielbitfeldes der Fall).

ENDMASK3-Register

Adresse	Zugriff	Funktion	Größe	Label
$FF 8A2C	R/W	Zu ändernde Zielbits (Zeilenende)	Word	Endmask2

Für das Schreiben des letzten Datenworts einer Zeile im Zielbitfeld wird dieses Register als Maske benutzt, um festzulegen, welche Bits geändert werden müssen.

Folglich ist also immer dann ein READ-MODIFY-WRITE-Zyklus für das Zieldatenwort erforderlich, wenn in dem zugehörigen ENDMASK-Register nicht alle Bits gesetzt sind (zur Veranschaulichung dient die Abbildung 2.15).

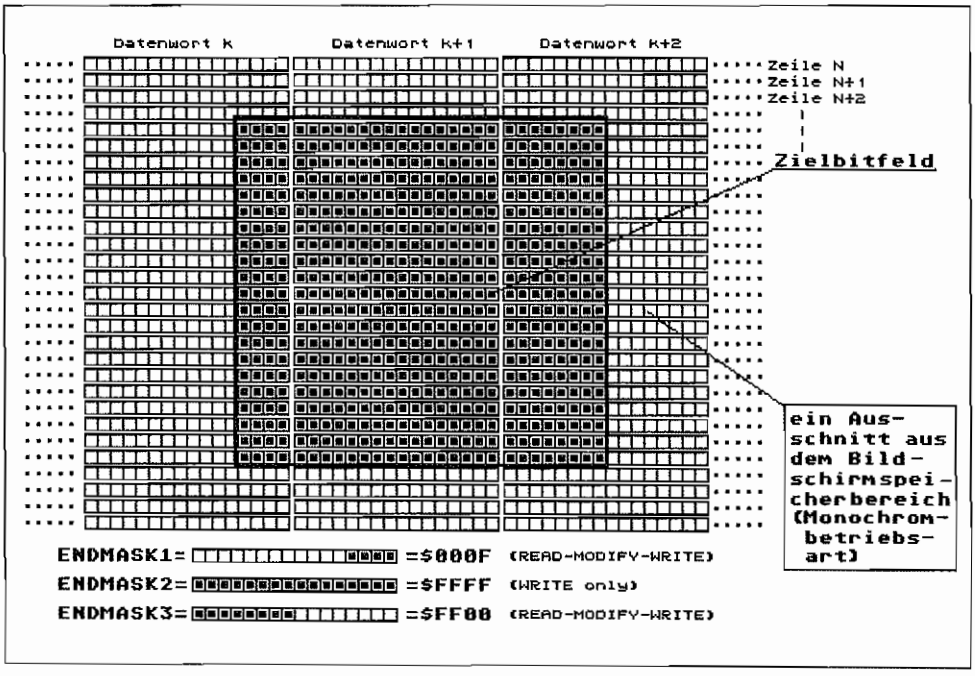


Abb. 2.15: Eine einfache Aufgabe für den Blitter: das Füllen eines Zielbitfeldes mit 1-Bits

Ist eine Zeile des Zielbitfelds übrigens "schmaler" als ein Datenwort (z. B. nur 12 Bit breit), so wird nur das ENDMASK1-Register benutzt.

Nachdem der Blitter ein Datenwort im Zielspeicherbereich bearbeitet hat, muß die nächste zu bearbeitende Adresse ermittelt werden. Dazu benötigt der Blitter die Angaben im Destination-X-Increment-, Destination-Y-Increment-, X-Count-, Y-Count- und Destination-Address-Register.

Mit diesen Registern wird sozusagen der zu bearbeitende Bitblock von seinen Abmessungen her festgelegt.

X-Count-Register

Adresse	Zugriff	Funktion	Größe	Label
$FF 8A36	R/W	Anzahl Words/Zeile im Zielbitfeld	Word	X_Count

Hiermit wird festgelegt, wie viele Datenworte zu einer Zeile des Zielbitfeldes gehören (in dem Beispiel in der Abbildung 2.15 steht also hier ein Wert von 3).

Der kleinste Wert ist 1 (mindestens ein Datenwort muß im Zielspeicherbereich schon zu bearbeiten sein, sonst braucht man ja keine Transferoperation durchzuführen!), und der größte Wert ist 65536 (wird durch einen Wert von $0000 dargestellt).

Der Blitter zählt jedesmal dann den Inhalt des Registers um 1 herunter, wenn er ein Datenwort geschrieben hat. Bei Erreichen von $0000 wird automatisch das X-Count-Register wieder auf den ursprünglich programmierten Wert gesetzt. (Ähnlichkeiten in der Arbeitsweise mit dem Timer-Data-Register des MFP sind zufällig und nicht beabsichtigt.) Beim Auslesen erhält man die Zahl der Datenworte, die in der augenblicklich in Bearbeitung befindlichen Zeile noch geschrieben werden müssen.

Y-Count-Register

Adresse	Zugriff	Funktion	Größe	Label
$FF 8A38	R/W	Zeilenzahl im Zielbitfeld	Word	Y_Count

Gibt an, aus wie vielen Zeilen das Zielbitfeld besteht. Wertebereich wie beim X-Count-Register (mindestens eine Zeile, maximal 65536 Zeilen sind möglich. Im Beispiel in Abbildung 2.15 also ein Wert von 21).

Nachdem eine komplette Zeile des Zielbitfeldes geschrieben ist, wird der Inhalt des Registers jeweils um 1 vermindert. Bei $0000 ist der Transfer dann beendet! Ein Lesezugriff liefert die Zahl der Zeilen, die noch geschrieben werden müssen.

Destination-X-Increment-Register

Adresse	Zugriff	Funktion	Größe	Label
$FF 8A2E	R/W	Distanz zwischen zwei Words im Zielbitfeld	Word	Dst_Xinc

Wird als vorzeichenbehaftete Zahl interpretiert (höchstwertiges Bit wird für Vorzeichen benutzt). Das niedrigstwertige Bit wird nicht benutzt, weil der Blitter immer mit Wordzugriffen arbeitet. Und die sind nur auf gerade Adressen erlaubt.

Anzugeben ist in diesem Register der Abstand in Bytes zwischen zwei Datenwörtern einer Zeile des Zielbitfeldes (Achtung bei Bitplanes! In Low-Resolution steht hier also 8 oder –8, bei Mid-Resolution dann 4 bzw. –4 und im Monochrom-Modus entspr. 2 oder –2). Ein negativer Wert weist darauf hin, daß die Zeile von rechts nach links bearbeitet wird.

Nachdem der Blitter ein Zieldatenwort geschrieben hat, wird der Wert aus dem Destination-X-Increment-Register unter Berücksichtigung seines Vorzeichens zu dem Inhalt im Destination-Address-Register addiert (jedoch nur, wenn das X-Count-Register einen Wert <>1 hat, also eine Zeile im Zielbitfeld aus mehr als einem Datenwort besteht).

So kann der Blitter in einstellbaren Schrittweiten den Zielspeicherbereich bearbeiten und nicht nur unmittelbar nebeneinanderliegende Datenwörter ansprechen (wichtig bei Bit-Block-Transfers auf den Bitplanes in der niedrigen und mittleren Bildschirmauflösung, bei denen ja die zu einem Plane gehörigen Datenwörter nicht unmittelbar nebeneinander, sondern vier bzw. acht Bytes auseinanderliegen!).

Destination-Y-Increment-Register

Adresse	Zugriff	Funktion	Größe	Label
$FF 8A30	R/W	Distanz zwischen zwei Zeilen im Zielbitfeld	Word	Dst_Yinc

Daten werden auch hier vorzeichenbehaftet betrachtet. Das Register enthält den Byte-Abstand von der *letzten* Zieladresse einer Zeile zur *ersten* Zieladresse in der nächsten Zeile des Zielbitfeldes. Dieser Wert wird nach dem Schreiben des letzten Datenworts einer Zeile

vorzeichenrichtig zum Inhalt des Destination-Address-Registers addiert. Somit zeigt das Destination-Address-Register dann wieder auf das nächste zu bearbeitende Zieldatenwort.

Sollte das X-Count-Register von vornherein nur mit einem Wert von 1 geladen worden sein (weil eine Zeile im Zielbitfeld max. ein Datenwort breit ist), arbeitet der Blitter bei der Ermittlung der nächsten Zieldatenadresse ausschließlich mit dem Destination-Y-Increment-Register. Eine Zeile im Zielbitfeld besteht dann ja nur aus einem Datenwort, und nach Bearbeitung dieses einen Wortes ist schon die nächste Zeile an der Reihe.

Destination-Adress-Register

Adresse	Zugriff	Funktion	Größe	Label
$FF 8A32	R/W	Zeiger auf (nächstes) Zieldatenwort	Long	Dst_Addr

Zu Beginn eines Bit-Block-Transfers ist hier die Anfangsadresse des ersten Wortes des Zielbitfeldes einzutragen. Während des Transfers wird das Register ständig mit Hilfe der Werte im Destination-X-Increment- und Destination-Y-Increment-Register auf den aktuellen Stand gebracht. Man würde deshalb bei einem Lesezugriff während eines Transfers die Adresse des nächsten zu bearbeitenden Zieldatenwortes erhalten.

Für etwas komplexere Operationen werden noch einige weitere Register gebraucht. Unter anderem werden diese erforderlich, wenn z. B. ein Zielbitfeld mit Daten eines Musters AND-verknüpft werden soll. Als Ergebnis bleiben also im Zielbitfeld nur jene Bits gesetzt, die auch im Muster gesetzt sind. Hier kommt nun das Halftone-RAM (Größe: 16 Datenwörter) ins Spiel.

Zunächst wird das Muster, mit dem das Zielbitfeld verknüpft werden soll, als 16x16 Bitfeld ins Halftone-RAM eingeschrieben. Dann wird das OP-Register mit dem Wert für die AND-Verknüpfung (OP-Nr.1) geladen. Anschließend muß dem Blitter noch mitgeteilt werden, daß die Verknüpfung von Daten im Zielbitfeld nur mit dem Muster aus dem Halftone-RAM durchgeführt werden soll.

Das geschieht über das

Halftone-Operation-Register

Adresse	Zugriff	Funktion	Größe	Label
$FF 8A3A	R/W	Halftone-Verknüpfungsvorschrift	Byte	HOP

Folgende Möglichkeiten sind einstellbar:

HOP	Verknüpfungsergebnis
0	Alle Bits = 1
1	Nur Halftone-Daten
2	Nur Source-Daten
3	Source AND Halftone

Das HOP-Register muß bei obigem Beispiel also mit dem Wert von 1 geladen werden, weil ja nur die Musterdaten aus dem Halftone-RAM benötigt werden (Achtung beim Registerzugriff! Das HOP-Register ist nur 1 Byte breit.)

Außerdem kommt jetzt noch ein weiterer Parameter ins Spiel:

Line-Number-Register

Adresse	Zugriff	Funktion	Größe	Label
$FF 8A3C	R/W	Zeiger auf Halftone-Verknüpfungsword	Byte	Line_Num

Bei einer Halftone-Operation gibt die Line Number an (gebildet aus den niedrigstwertigen vier Bits in diesem Register), welches der 16 Wörter im Halftone-RAM als Verknüpfungsdatenwort für die gerade zu bearbeitende Zeile benutzt wird.

Abhängig vom Destination-Y-Increment-Register wird am Ende einer Zeile die Line Number um 1 erhöht (bei positivem Wert in Destination-Y-Increment-Register) oder um 1 erniedrigt (bei negativem Wert im Destination-Y-Increment-Register), um so das nächste Verknüpfungsdatenwort im Halftone-RAM zu adressieren. Die Line Number wird "rund" gezählt, d. h., bei einem Wert von 15 wird bei weiterer Erhöhung der Zähler wieder bei 0 beginnen. Bei einer weiteren Dekrementierung bei einem Zählerstand von 0 wird die Line Number entsprechend auf 15 springen.

Die schon bekannten Register (Destination-Address-, Destination-X-Increment, Destination-Y-Increment-, X-Count- und Y-Count-Register) werden entsprechend mit Werten, die sich aus der Lage und den "Abmessungen" des zu füllenden Zielbitfeldes ergeben, geladen.

Im Prinzip funktioniert die ganze Angelegenheit genauso, wenn man statt des Musters aus dem Halftone-RAM ein Quellbitfeld zur Verknüpfung mit dem Zielbitfeld heranzieht. Nur das HOP-Register muß dann anders gesetzt werden (HOP-Nr.2 oder HOP-Nr.3).

Der Quellbitblock hat dann die gleichen "Abmessungen" wie der Zielbitblock (X-Count und Y-Count gelten sowohl für den Source- als auch den Destination-Bit-Block!). Unterscheiden dürfen sich jedoch die Werte für X-Increment und Y-Increment, weshalb auch für den Quellbitblock eigene Register zur Aufnahme dieser beiden Werte vorgesehen sind.

Source-X-Increment-Register

Adresse	Zugriff	Funktion	Größe	Label
$FF 8A20	R/W	Distanz zwischen zwei Words im Quellbitfeld	Word	Src_Xinc

Hier wird der Abstand (in Bytes) zwischen zwei Datenwörtern in einer Zeile des Quellbitblocks eingetragen (unter Berücksichtigung des Vorzeichens!). Nachdem jeweils ein Source-Wort gelesen wurde, wird der Wert dieses Registers zum Inhalt des Source-Address-Registers vorzeichenrichtig addiert (solange X-Count <>1 ist!).

Source-Y-Increment-Register

Adresse	Zugriff	Funktion	Größe	Label
$FF 8A22	R/W	Distanz zwischen zwei Zeilen im Quellbitfeld	Word	Src_Yinc

Abstand in Bytes zwischen dem letzten Wort einer Zeile des Quellbitfeldes und dem ersten Wort der nächsten Zeile (mit Vorzeichen!).

Wenn das letzte Wort einer Zeile des Quellbitblocks gelesen ist, wird der Wert im Source-Y-Increment-Register zum Inhalt des Source-Address-Registers vorzeichenrichtig addiert. Das Source-Address-Register zeigt danach wieder auf das nächste zu lesende Datenwort des Quellbitfeldes.

Source-Adress-Register

Adresse	Zugriff	Funktion	Größe	Label
$FF 8A24	R/W	Zeiger auf (nächstes) Zieldatenwort	Long	Src_Addr

Zu Beginn eines Bit-Block-Transfers wird hier die Anfangsadresse des ersten Wortes des Quellbitfeldes eingetragen.

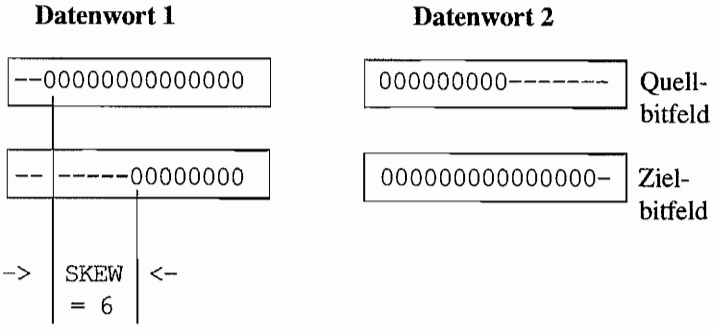
Während des Transfers wird das Source-Adress-Register ständig mit Hilfe der Werte im Source-X-Increment- und Source-Y-Increment-Register auf den aktuellen Stand gebracht. Bei einem Lesezugriff während des Transfers steht hier die Adresse des nächsten zu lesenden Quelldatenwortes. Werden Quellbitfeld-Daten in die Verarbeitung einbezogen, kommt der im Datenflußbild mit Source-Buffer bezeichnete Funktionsblock ins Spiel. Bei einer Bit-Block-Operation mit Sourcedaten spielt sich im Blitter dann folgendes ab (Zum besseren Verständnis sollte man die Abbildung 2.14 betrachten):

Die ersten beiden Sourceworte werden in den Source-Buffer eingelesen. Zur Verküpfung mit den Zielbits im Zieldatenwort muß eine Justierung (Bitverschiebung) der Sourcedaten im Source-Buffer erfolgen (das erste Bit des Quellbitfeldes soll ja mit dem ersten Bit des Zielbitfeldes verknüpft werden usw.). Den Grad der Justierung erfährt der Blitter aus dem SKEW-Register (dazu später mehr).

Ist das erste Quelldatenwort (nach entsprechender Justierung) mit dem ersten Zieldatenwort verknüpft, werden die 16 niederwertigen Bits im Source-Buffer in die 16 höherwertigen Bits des Source-Buffers übertragen. Das nächste zu lesende Sourcewort gelangt in die niederwertigsten 16 Bit des Registers. Es erfolgt wieder die Justierung im Source Buffer, dann die Verknüpfung mit dem Zieldatenwort usw.

Nun folgen noch die Erläuterungen für die verbleibenden Flags und Register. Unter der Adresse $FF 8A3D findet man die folgenden Flags und Werte:

SKEW (Label: "Skew") In die vier niederwertigsten Bits an Adresse $FF 8A3D muß der sogenannte Source-Skew eingetragen werden. Darunter versteht man die Zahl der nötigen Rechtsverschiebungen, die mit den Source-Daten im Source Buffer (siehe Datenflußschema) durchgeführt werden müssen, bevor diese Daten mit den Halftone- und Destination-Daten verknüpft werden dürfen. *Beispiel*: Dargestellt sind jeweils die ersten beiden Datenworte des Quell- und Zielbitfeldes.



Der Quellbitblock beginnt im Beispiel bei Bit 13 im Datenwort 1 des Quellbitfeldes, während der Zielbitblock erst bei Bit 7 im ersten Datenwort des Zielbitfeldes anfängt! Damit also jeweils die zusammengehörigen Bits (das erste Bit im Quellbitblock korrespondiert ja mit dem ersten Bit im Zielbitblock usw.) miteinander verknüpft werden können, müssen diese erst "übereinandergeschoben" werden.

Um wie viele Bitpositionen die Daten "geschoben" werden müssen, ist im Skew-Wert anzugeben.

FXSR (Force Extra Source Read) Ist das höchstwertige Bit im Blitter-Register an Adresse $FF 8A3D gesetzt, so wird zu Beginn einer jeden Zeile ein zusätzliches Datenwort eingelesen, um entsprechend "Luft" für Verschiebungen im Source-Buffer zu schaffen.

NFSR (No Final Source Read) Wenn Bit 6 im Blitter-Register an Adresse $FF 8A3D gesetzt ist, wird das letzte Quell-Datenwort einer Zeile nicht gelesen. Bei einigen Kombinationen von ENDMASK-Inhalten und SKEW-Werten kann auf das Lesen des letzten Datenwortes einer Zeile des Quellbitfeldes verzichtet werden.

Zu den FXSR- und NFSR-Bits hier noch einige Erläuterungen:

Ein Block mit einer Breite von beispielsweise 20 Bits benötigt mindestens zwei Datenworte/Zeile zu seiner Darstellung. Je nachdem, bei welchem Bit des ersten Datenwortes das Bitfeld beginnt, können für die Darstellung dieses 20 Bit breiten Bitfeldes aber auch drei Datenworte/Zeile nötig sein! Es ist also vorstellbar, daß im Quellbitfeld für die 20 Bits schon zwei Datenworte/Zeile ausreichend sind, während im Zielbitfeld drei Datenworte erforderlich sind. Der umgekehrte Fall ist natürlich genauso denkbar.

Daraus folgt die Notwendigkeit, evtl. eine unterschiedliche Zahl von Lese- und Schreibzugriffen in einer Zeile des Bit-Blocks durchführen zu müssen. Für diese Steuerung sind die beiden Bits FXSR und NFSR erforderlich.

Die folgenden Beispiele zeigen, abhängig von der Lage der Quell- und Zielbitfelder zueinander, die jeweils nötige Einstellung der FXSR- und NFSR-Bits: (Die Bearbeitung soll von rechts nach links erfolgen!)

S = Sourcebits	**D = Destin.-Bits**	**FXSR**	**NFSR**
----SSSSSSSSSSS	SSSSSSS--------	0	0
-------DDDDDDDDD	DDDDDDDDDD-----		

S = Sourcebits	D = Destin.-Bits	FXSR	NFSR
----SSSSSSSSSSSS	SSSSSSSS--------	0	1
------------DD	DDDDDDDDDDDDDDDD DD------		
---------------S	SSSSSSSSSSSSSSSS SSS-----	1	0
-------DDDDDDDDD	DDDDDDDDDD-----		
----SSSSSSSSSSSS	SSSSSSSS--------	1	0
--DDDDDDDDDDDDD	DDDDDD----------		

An Adresse $FF 8A3C (Label: "Line_Num", also dort, wo auch die Line Number zu finden ist) existieren noch die folgenden Steuerbits:

SMUDGE Dieses Bit befindet sich an der gleichen Adresse wie die Line Number für das Halftone-RAM ($FF 8A3C). Ist es gesetzt, bestimmt nicht die Line Number, welches Datenwort aus dem Halftone-RAM bei einer Halftone-Operation zur Verknüpfung benutzt wird, sondern die niederwertigsten vier Bits der verschobenen (ge"SKEW"ten) Sourcedaten im Source-Buffer. Das ergibt quasi zufällige Verknüpfungen mit den Daten im Halftone-RAM und kann für "Schmiereffekte" benutzt werden.

HOG Hier wird die Betriebsart des Blitters eingestellt. Bei gelöschtem Bit teilen sich CPU und Blitter Zugriffe auf den Bus zu gleichen Teilen auf. Jeder darf für 64 Zyklen an den Bus, während der andere angehalten wird!

- Ist das HOG-Bit gesetzt, wird die CPU so lange angehalten, bis der Bit-Block-Transfer beendet ist. Es darf aber durchaus die CPU mal den einen oder anderen Befehl ausführen, wenn der Blitter zwischendurch mal den Bus freigibt! Mit gesetztem HOG-Bit kann der Bit-Block-Transfer bis zu doppelt so schnell abgewickelt werden, als wenn sich CPU und Blitter den Bus im Verhältnis 1:1 teilen müssen. In beiden Betriebsmodi nimmt der Blitter aber Rücksicht auf den DMA-Baustein. Dieser hat nämlich Vorrang!

- ATARI betreibt den Blitter standardmäßig mit gelöschtem HOG-Bit (CPU und Blitter teilen sich Buszugriffe für jeweils 64 Buszyklen). Um jedoch nahezu die gleiche Verarbeitungsgeschwindigkeit wie bei gesetztem HOG-Bit zu erreichen (ca. 90% des HOG-Modus), macht die CPU während "ihrer" 64 Busszyklen nichts anderes, als den Blitter sofort wieder neu zu starten, so daß dieser unmittelbar (nach ca. sieben CPU-Buszyklen) die Buskontrolle zurückbekommt!

Programmbeispiel:

```
          :
          :
START:
        lea     Line_Num,A0     ; Pointer auf HOG-Register -> A0
        bset.b  #HOGBIT,(A0)    ; HOG-Bit löschen (Kein HOG-Modus!)

RESTART:
        bset.b  #BUSYBIT,(A0)   ; BUSY-Bit testen u. setzen (startet
        nop                     ; den BLITTER sofort wieder neu!)
        bne.s RESTART           ; Der "nop"-Befehl wird noch ausgeführt,
          :                     ; bevor der Blitter neu startet!
```

Abb. 2.16: Programmbeispiel

BUSY

Wenn alle anderen Register des Blitters gesetzt sind, wird durch Setzen des BUSY-Bits im Blitter der Transfer ausgelöst. Der auf das Setzen des BUSY-Bits folgende Befehl wird von der CPU in der Regel aber noch ausgeführt, bevor der Blitter das Regiment übernimmt!

Die Interrupt-Leitung zum MFP, Port I3 (GPU done), ist direkt mit diesem Bit gekoppelt. Wenn der Blitter fertig ist, wird das BUSY-Bit zurückgesetzt, und die GPU-done-Leitung geht zurück auf Low.

Wer's gerne knifflig mag,

kann den Blitter ja direkt programmieren. Die besprochenen Register und Steuerbits müssen jedoch alle vom Programmierer vor dem Bit-Block-Transfer selbst gesetzt werden! Die Beispiele für SKEW, NFSR, FXSR usw. waren ja noch relativ anschaulich. Wer Lust hat, kann sich ja mal überlegen, was alles zu beachten ist, wenn sich zu übertragende Bitblöcke überlappen (und dabei gibt's dann noch die Möglichkeiten, daß die Sourceadresse kleiner oder größer als die Zieladresse ist)!

Aber ATARI hat den Bit-Block-Transfer in neueren TOS-Versionen "transparent" für den Programmierer ausgelegt. Das bedeutet nichts anderes, als daß die BitBlt-Funktion die mit der Line-A Funktion $A007 aufgerufen wird, sowohl durch den Blitter als auch in Softwareemulation durch die CPU (wie gehabt) ausgeführt werden kann. Wenn der Blitter eingebaut ist, bietet

einem das TOS in der Desktop-Menüleiste unter dem Eintrag "Extras" die Möglichkeit, diesen ein- und auszuschalten. Die gleiche Funktion ist mit dem XBIOS-Aufruf #64 ("Blitmode") möglich. Sie kann auch benutzt werden, um abzufragen, ob der Blitter eingebaut ist. Näheres dazu in Teil I, Kapitel 1, in der "XBIOS-Referenz". Zur hardwaremäßigen Einbindung des Blitters sei auf die Abbildung 2.17 verwiesen.

Wie dort zu sehen, handelt es sich beim Blitter mit seinen 23 Adreß- und 16 Datenleitungen sowie dem kompletten Satz an Steuer- und Meldeanschlüssen um einen vollwertigen Mikro-

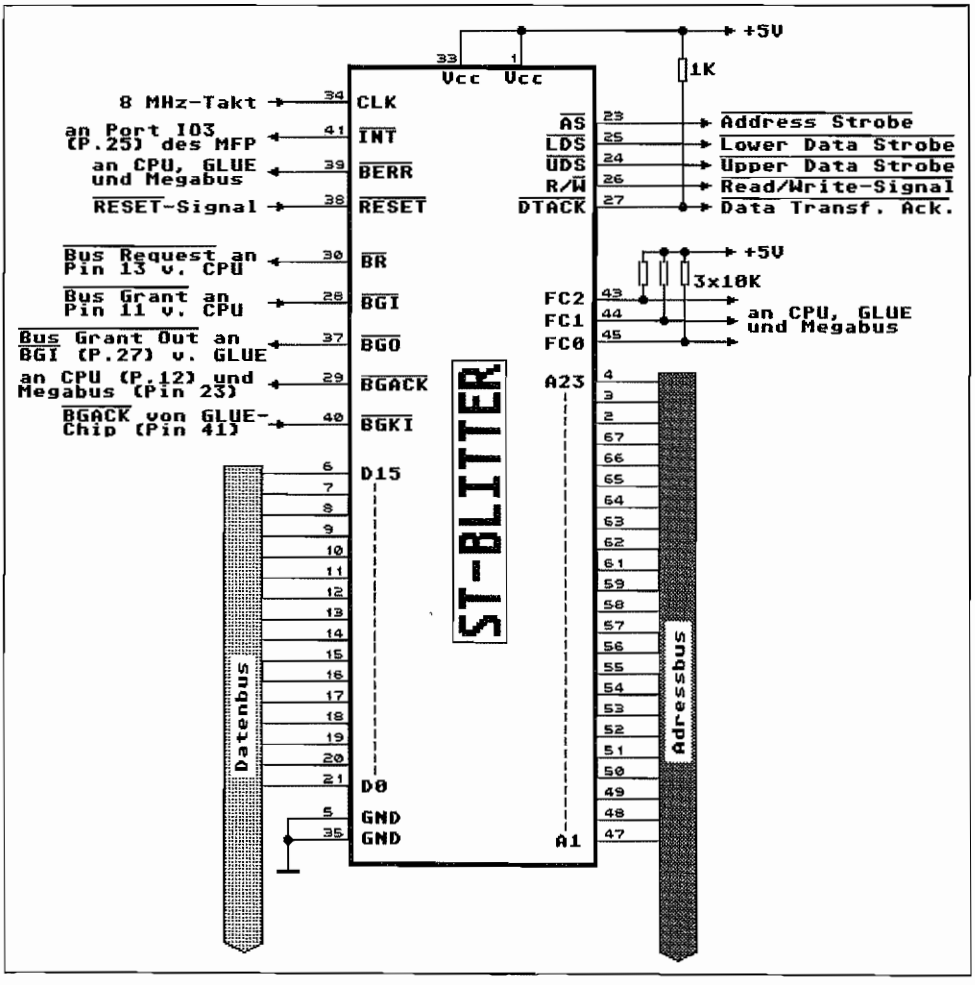


Abb. 2.17: Die Einbindung des BLITTERs in die ST-Hardware

prozessor, der sich in einem 68000er-System "zu Hause" fühlt. Als eigenständiges DMA-Device kann er deshalb in einem 68000er-System die Bussteuerung übernehmen.

Die Taktversorgung erfolgt ebenfalls mit dem gleichen 8-MHz-Takt, wie ihn die CPU des ST erhält.

Auf die einzelnen Steuer- und Meldeleitungen im einzelnen einzugehen,würde den Rahmen sprengen. Außerdem haben diese im Prinzip die gleiche Funktion wie bei einer 68000er-CPU (ein paar Informationen finden sich auch in Teil II, Kapitel 1, im Abschnitt über den Systembus des MEGA ST).

Die Anschlüsse BGI und BGO dienen der Verkettung des Blitters mit den anderen DMA-Einheiten des ST. Das Signal BG (Bus Grant = Bus Freigabesignal) gelangt nämlich von der CPU an BGI (Bus Grant In) des Blitters und von dessen BGO-Anschluß (Bus Grant Out) wieder hinaus an BGI des GLUE-Chips (der einen Großteil der DMA-Logik des ST enthält). Ebenfalls wird das BGACK-Signal vom GLUE erst durch BGKI - BGACK des Blitters "gefädelt", bevor es dann als BGACK-Signal an die CPU gelangt. Damit bekommt der Blitter immer als erster mit, wenn die DMA-Einheit des ST den Bus beansprucht.

Kapitel 3: Der Soundgenerator

Die ST-Computer können nicht nur per MIDI-Schnittstelle aktiv ins Musikgeschehen eingreifen, sondern sind auch in der Lage, eigene Laute von sich zu geben. So wird z. B. der Keyclick, die akustische Untermalung der Tastaturbedienung, von dem eingebauten Soundchip erzeugt.

Die Konstrukteure von Atari haben hierbei auf einen Standard-Chip zurückgegriffen, der zwar nicht durch überragende Soundeigenschaften hervortritt, aber wohl ein recht gutes Kosten-/Leistungsverhältnis aufweist. Zum Einsatz kommt ein Chip des Typs "YM 2149" von Yamaha oder der Typ "AY-3-8910" von General Instruments. Beide sind untereinander voll hard- und softwarekompatibel.

Die Tonerzeugung liegt voll unter Programmkontrolle, alle wesentlichen Parameter sind durch Software einstellbar. Der PSG (Programmable-Sound-Generator) wird mit einem aus dem Systemtakt gewonnenen 2 MHz Arbeitstakt versorgt. Daraus können mit Hilfe von drei programmierbaren Teilerstufen Ausgangsfrequenzen von 30 Hz bis 125 KHz erzeugt werden. Die Ausgangssignale haben Rechteckform.

Rauschen erwünscht

Ferner integriert in den PSG-Chip ist ein Rauschgenerator, dessen Ausgangssignal beliebig über eine Mixstufe mit den Ausgangssignalen der drei Tongeneratoren kombiniert werden kann. Das Rauschsignal ist ein Rechtecksignal, dessen Pulsbreite durch Pseudo-Zufallswerte bestimmt wird.

Die CPU wird bei der Tonerzeugung nur wenig benötigt, da der PSG ein gewisses Maß an Eigenintelligenz besitzt. Sind die für einen bestimmten Sound nötigen Parameter in den Registern des PSG eingestellt, so liegt die eigentliche "Arbeit" der Tonerzeugung beim Soundchip. Die CPU muß nur aktiv werden, wenn der Sound verändert werden soll.

Lautstärke unter Programmkontrolle

Zur Beeinflussung der Lautstärke der erzeugten Töne steht für jeden der drei Soundkanäle ein elektronischer Lautstärke-Steller zur Verfügung. Es besteht die Möglichkeit, die Ausgangsamplitude (Lautstärke) in 16 Stufen einzustellen. Diese Abstufung verläuft nicht linear, sondern logarithmisch und ist damit dem Hörempfinden des menschlichen Ohres besser angepaßt. Weiterhin bietet der PSG die Möglichkeit, bestimmte Lautstärkeverläufe eines Tones abzuru-

fen und damit den Amplitudenverlauf des Ausgangssignals zu steuern. Acht solcher Hüllkurven (Envelopes) sind bereits vorprogrammiert und können abgerufen werden.

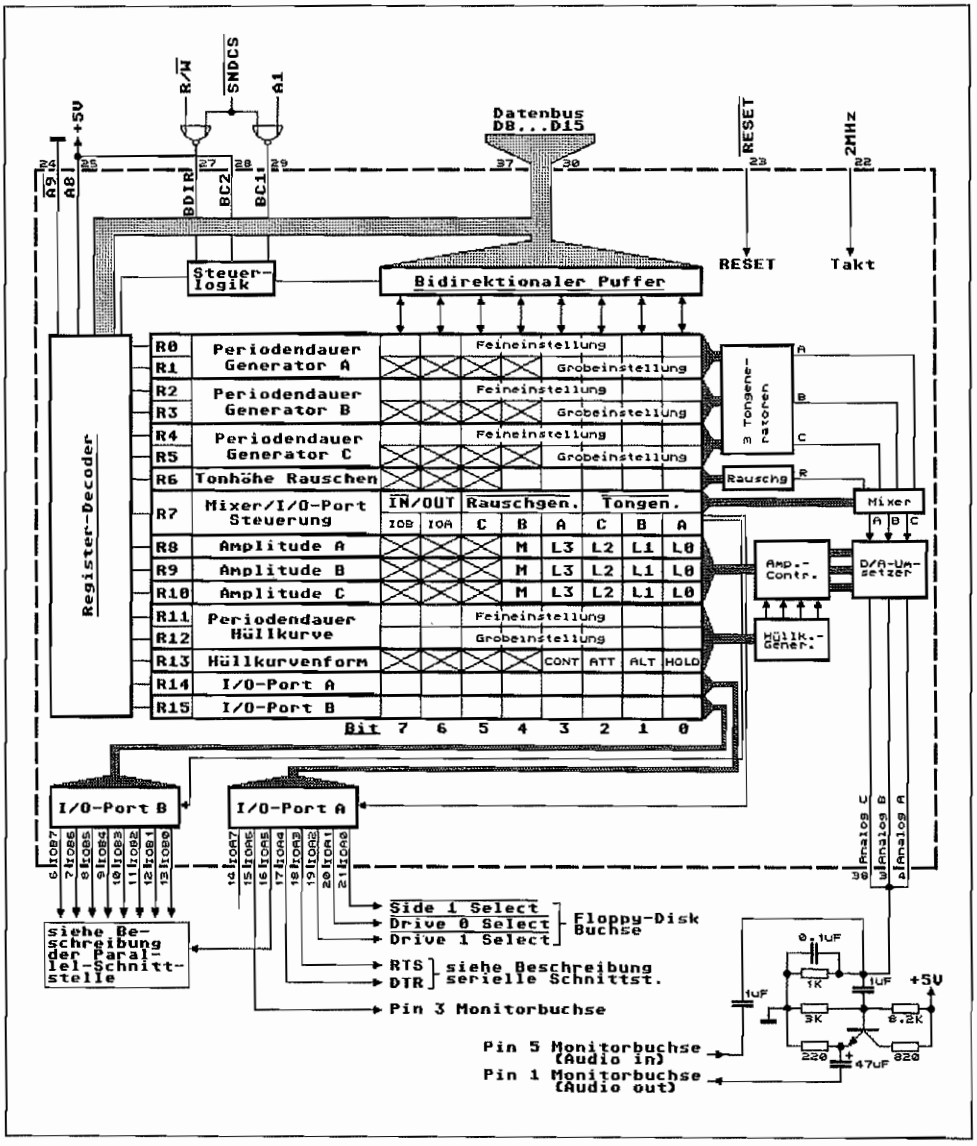


Abb. 3.1: Die Einbindung des Soundchips in die ST-Hardware

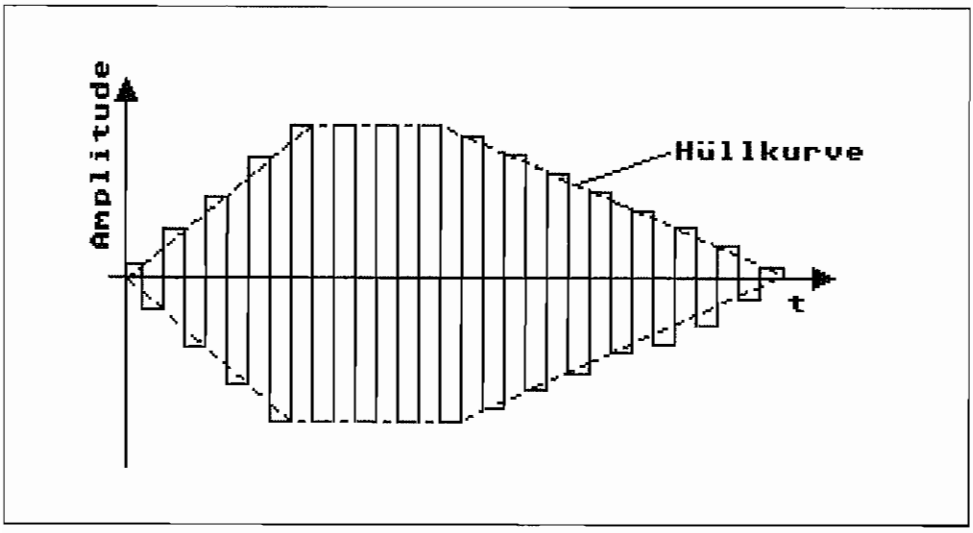


Abb. 3.2 : Die Hüllkurve um ein Rechtecksignal

Nicht nur Töne werden erzeugt

Zusätzlich zu seinen Tongeneratoren besitzt der PSG-Chip noch zwei I/O-Ports, die zur Bedienung der parallelen Schnittstelle, der seriellen Schnittstelle und für die Erzeugung einiger Steuersignale für die Floppy-Disk-Steuerung benutzt werden.

Zur gesamten Steuerung aller Funktionen des PSG besitzt dieser 16 Register mit jeweils 8 Bit Breite (also Bytezugriffe erforderlich!).

Diese 16 Register sind jedoch nicht direkt zugänglich, sprich an 16 verschiedenen Adressen im I/O-Speicherraum des ST zu finden, sondern werden folgendermaßen angesprochen:

- In die Speicherstelle $FF 8800 (Label: "giselect") wird die Nummer des Registers eingeschrieben, das angesprochen werden soll (Registernummer 0..15).

- Soll der Inhalt des so ausgewählten Registers verändert werden, so muß dessen neuer Inhalt an Speicherstelle $FF 8802 (Label: "giwrite") geschrieben werden.

- Den Inhalt eines selektierten Registers kann man an Adresse $FF 8800 (Label: "giread") auslesen.

Diese Art von Zugriff mag umständlich erscheinen, ist jedoch auf die Hardwaregegebenheiten des PSG-Chips zurückzuführen (Gemultiplexter Adreß- und Datenbus)! Ein Vorteil dieser Zugriffsart liegt darin, daß alle PSG-Register für eine Zugriffsart (Schreiben oder Lesen) auf der gleichen Speicherstelle zu finden sind. So entfallen Adreßberechnungen für Zugriffe auf verschiedene Register. Es wird so lange auf das gleiche PSG-Register zugegriffen, bis ein anderes ausgewählt wird.

	Bit	7	6	5	4	3	2	1	0
R0	Periodendauer Generator A			Feineinstellung					
R1		X	X	X	X	Grobeinstellung			
R2	Periodendauer Generator B			Feineinstellung					
R3		X	X	X	X	Grobeinstellung			
R4	Periodendauer Generator C			Feineinstellung					
R5		X	X	X	X	Grobeinstellung			
R6	Tonhöhe Rauschen	X	X	X					
R7	Mixer/I/O-Port Steuerung	IN/OUT		Rauschgen.			Tongen.		
		IOB	IOA	C	B	A	C	B	A
R8	Amplitude A	X	X	X	M	L3	L2	L1	L0
R9	Amplitude B	X	X	X	M	L3	L2	L1	L0
R10	Amplitude C	X	X	X	M	L3	L2	L1	L0
R11	Periodendauer Hüllkurve			Feineinstellung					
R12				Grobeinstellung					
R13	Hüllkurvenform	X	X	X	X	CONT	ATT	ALT	HOLD
R14	I/O-Port A								
R15	I/O-Port B								

Abb. 3.3: Die Register des Soundchips

Nun zur Bedeutung der einzelnen Register, wie in Abbildung 3.3 dargestellt:

R0..R5 Jeder der drei Tongeneratoren besitzt zur Frequenzeinstellung zwei 8-Bit-Register. Die oberen vier Bits des höherwertigen Registers werden jedoch nicht benutzt, so daß sich insgesamt ein 4 + 8 = 12 Bit-Wert für die Tonfrequenzsteuerung ergibt. Die Tonerzeugung geschieht dabei nach folgendem Prinzip:

Der Arbeitstakt des PSG (2 MHz) wird zunächst durch 16 geteilt. Daraus ergibt sich als Eingangsfrequenz für die drei Tonerzeugungsstufen eine Frequenz von 125 kHz. Dieser Takt gelangt in jeder der Tonerzeugungsstufen auf eine Teilerstufe, deren Teilerfaktor durch den 12-Bit-Wert im zugehörigen "Tongenerator-Control-Register" bestimmt wird. Ist also z. B. im "Periodendauer-Generator-C-Register" der Wert wie folgt

	Bit	7	6	5	4	3	2	1	0
R4	Periodendauer	1	1	1	0	1	0	0	0
R5	Generator C	X	X	X	X	0	0	1	1

eingetragen (entspricht einem Wert von $\$03E8_{hex.}$ (= $1000_{dez.}$)), so ergibt sich daraus eine Ausgangsfrequenz für den Tongenerator C von:

125 kHz / 1000 = 0.125 kHz = 125 Hz

Je kleiner also der Wert im Tongenerator-Control-Register ist, desto heller klingt der erzeugte Ton.

R6

Die für den Rauschgenerator maximal zur Verfügung stehende Frequenz ist ebenfalls 125 kHz. Diese Arbeitsfrequenz gelangt, analog zu der Tongenerator-Steuerung, auf eine mit dem "Tonhöhe-Rauschen-Register" zu programmierende Teilerstufe. Die daraus gewonnene Frequenz dient dem eigentlichen Rauschgenerator als Eingangsfrequenz. Auch hierbei gilt: Je kleiner der Wert im Register 6 (Tonhöhe Rauschen), desto heller klingt das Rauschen.

Bild 7

Durch entsprechendes Setzen bzw. Löschen einzelner Bits in diesem Register 7 wird für jeden der drei möglichen Soundkanäle eingestellt, ob ein "reiner" Ton erzeugt wird oder ob dieser mit Rauschen vom Rauschgenerator unterlegt wird.

Die Steuerung erfolgt Low-aktiv, d. h., bei gelöschtem Bit ist die entsprechende Funktion angewählt. Ist beispielsweise nur Bit 1 Low, dann wird nur das Signal des Tongenerators B weiter verwendet. Wird jetzt noch zusätzlich Bit 4 Low, so wird das Signal des Tongenerators B mit dem Signal des Rauschgenerators zusammengemischt. Außerdem wird durch Register 7 (Bit 6..7) festgelegt, ob die I/O-Ports des PSG als Eingänge oder Ausgänge

arbeiten. Es ist jedoch nicht möglich, für jeden Anschluß der PSG-Ports separat (also bitweise) die Datenrichtung festzulegen!

R8..R10 Die Register R8 .. R10 steuern die "Lautstärkesteller" des PSG. Dabei sind die Bits "L0..L3" die sogenannten "Level-Control-Bits" und bestimmen den Ausgangspegel der drei Soundkanäle.

Die Ausgangsamplitude läßt sich mit diesen vier Bits in 16 Stufen einstellen. Ist das "M-Bit" gesetzt, wird der Verlauf der Ausgangsamplitude des entsprechenden Kanals nicht durch die Level-Control-Bits, sondern durch den Hüllkurvengenerator gesteuert.

R11/R12 Der aus den 2 x 8 = 16 Bit der Register 11 und 12 des PSG gebildete Wert beeinflußt die Periodendauer der Hüllkurve. Die Form der Hüllkurve wird mit Register 13 ausgewählt.

Als Eingangsfrequenz steht hierbei die durch 256 geteilte Taktfrequenz des PSG (2 MHz/256 = 7,8125 kHz) zur Verfügung. Diese wird entsprechend dem 16 Bit-Wert in Register 11 und 12 weiter heruntergeteilt. Es lassen sich so Periodendauern von 128 µSek. (Reg. 11 + 12 = $0000) bis ca. acht Sek. (Reg. 11 + 12 = $FFFF) einstellen.

R13 Mit den vier niederwertigen Bits des Registers 13 wird die Hüllkurvenform eingestellt, die durch den Hüllkurvengenerator erzeugt wird. Da nur ein Envelope-Generator vorhanden ist, gilt der Verlauf der Hüllkurve für alle Soundkanäle, für die das M-Bit im Amplitudenregister gesetzt ist. Es läßt sich also leider nicht für jeden Soundkanal eine eigene Hüllkurve einstellen! Die oberen vier Bits des Registers haben keine Funktion.

R14/R15 Über Register 14 und 15 wird der Datentransport mit den beiden I/O-Ports des PSG abgewickelt. Port A ist beim ST als Ausgang geschaltet und kontrolliert einige Floppy-Disk-Steuerleitungen sowie Steuerleitungen der seriellen und parallelen Schnittstelle (siehe Abbildung 3.1).

Port B bedient die Datenleitungen der parallelen Schnittstelle.

Wie schon der Schaltungsauszug (Abbildung 3.1) zeigt, sind die drei Tonausgänge des PSG direkt miteinander verbunden.

Das Summensignal der drei Soundkanäle gelangt über einen Impedanzwandler an Pin 1 der Monitorbuchse und kann dort abgenommen werden.

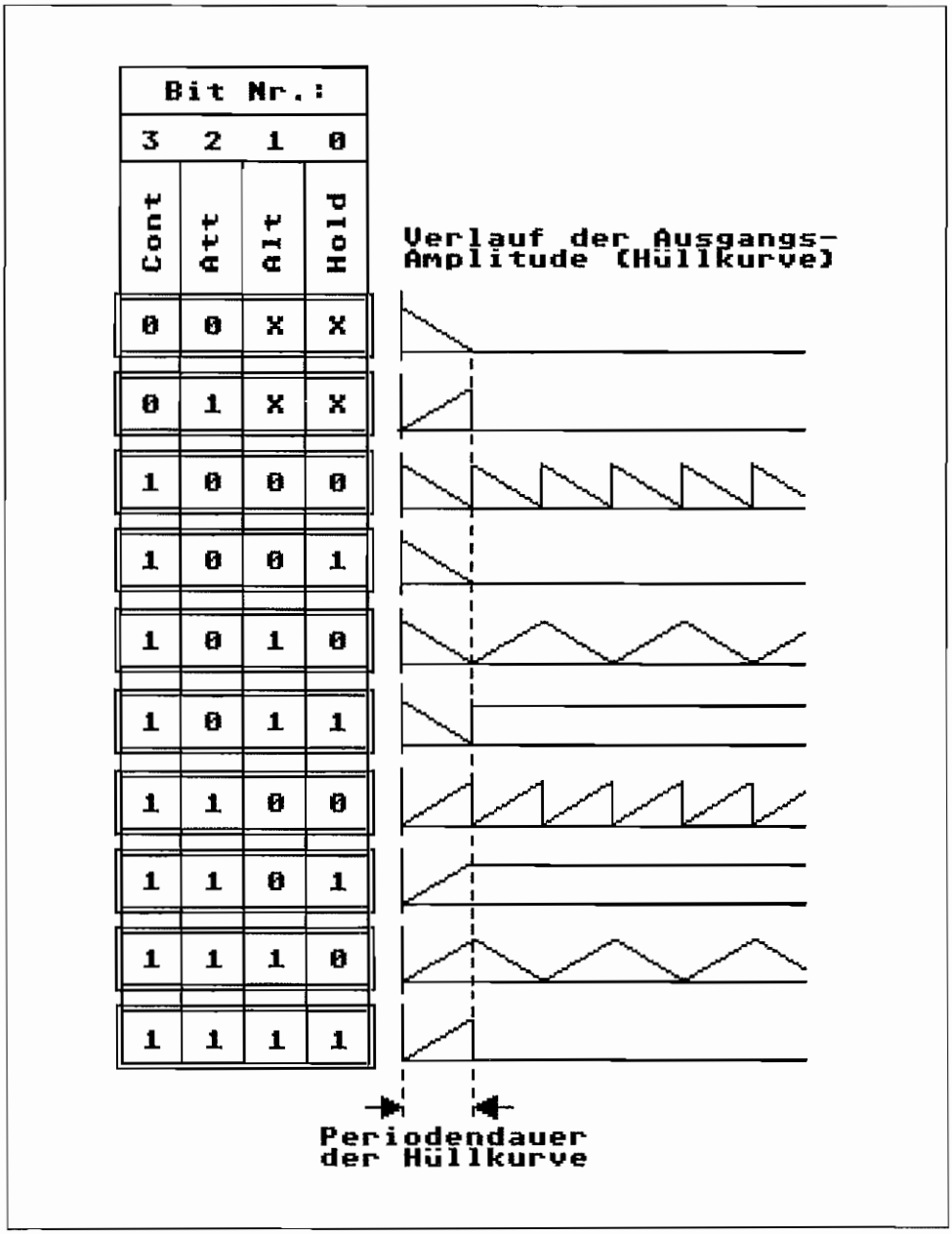


Abb. 3.4: Die Bedeutung der Steuerbits im Hüllkurvenform-Register (Reg. 13)

Es besteht die Möglichkeit, ein externes Tonsignal über Pin 5 der Monitorbuchse einzuspeisen. Dieses Signal wird dann zu den vom PSG erzeugten Signalen hinzugemischt. Eine weitere Beeinflussung des externen zugeführten Signals ist nicht möglich.

Von Port A des PSG sind die Ausgänge IOA6 und IOA7 im ST nicht weiter benutzt und stehen dem Endbenutzer für eigene Anwendungen zur Verfügung (viele Erweiterungen wie z. B. 16 MHz-Beschleunigerkarten benutzen diese freien Ports). Die Ausgänge können eine Standard-TTL-Last treiben.

- IOA6 ist auf Pin 3 der Monitorbuchse geführt und kann dort abgegriffen werden.

- IOA7 ist nur im ST am PSG direkt abzugreifen und nicht nach außen geführt.

Betriebssystemunterstützter Sound

Der PSG ist von den Systemprogrammierern bei der Programmierung des Betriebssystem berücksichtigt worden. Für das Löschen bzw. Setzen einzelner Bits in I/O-Port A des PSG sind die XBIOS-Funktionen #29 und #30 ("Offgibit" und "Ongibit") vorgesehen. Mit der XBIOS-Funktion #28 ("Giaccess") kann auf jedes Register des PSG zugegriffen werden.

Außerdem ist ein Pseudo-Sound-Prozessor mit eigenem Pseudo-Befehlssatz per XBIOS-Aufruf (siehe XBIOS-Funktion #32, "Dosound") aufrufbar. Dieser Aufruf erlaubt das relativ einfache Programmieren von Soundeffekten, welche quasi neben einem anderen Programm ablaufen können (im Timer C-Interrupt). Durch die Verwendung des Port A des PSG für die Floppy-Disk-Steuerung ist das unmittelbare Ansprechen der Floppy-Steuerleitungen des PSG aus z. B.. BASIC heraus, durch PEEK- und POKE-Befehle etwas schwierig.

Während des VBlank-Interrupts, also alle 20 mSek. bei Farbbetriebsart bzw. alle 14 mSek. in Monochrombetriebsart, werden nämlich vom Betriebssystem die angeschlossenen Disklaufwerke abgefragt, um festzustellen, ob der Motor noch läuft. Wenn dieser nicht mehr läuft, wird das entsprechende Laufwerk deselektiert. Um jedoch festzustellen, ob der Motor noch läuft, muß das entsprechende Disklaufwerk jeweils selektiert werden. Das erfolgt über den PSG, Port A, Leitung 1 oder 2.

Es ist also bei PEEK und POKE aus BASIC heraus ziemlich wahrscheinlich, daß zwischen einer vom Anwender erfolgten Registerauswahl im PSG und dem Beschreiben oder Lesen des selektierten Registers der VBlank-Interrupt des Betriebssystems dazwischenrutscht.

Diese Routine selektiert aber immer das Port A-Register, um dort mit den Drive-Select-Leitungen "herumzuspielen". Leider ist diese Routine des Betriebssystems nicht so program-

miert, daß sie den Zustand des PSG nach Laufwerksselektion wieder so herstellt, wie er vor Laufwerksauswahl vorgefunden wurde!

Deshalb ist es empfehlenswert, bei Registerzugriffen auf den PSG-Chip aus Hochsprachen wie z. B.. BASIC heraus die Betriebssystemroutinen zu verwenden.

Hier als Beispiel ein kleines OMIKRON.BASIC-Programm zum Auslesen der Register des ST-Soundchips unter Benutzung der XBIOS-Funktion #28, "Giaccess":

```
100 ' Programm zum Auslesen der 16 Register des ST-Soundchips
110 ' durch Benutzung der XBIOS-Funktion "Giaccess" (XBIOS #28)
120 '
130 CLS
140 PRINT
150 PRINT TAB (10);"  Registerinhalte des PSG-Chips"
160 PRINT TAB (10);"================================="
170 PRINT
180 PRINT "  PSG-Register      | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |"
190 PRINT "  -----------------+---+---+---+---+---+---+---+---+"
200 FOR Register%=0 TO 15
210     USING "####"
220     PRINT "    Register ";Register%;"    | ";
230     Print_Bits(Register%)
240 NEXT Register%
250 END
260 '
270 ' Registerinhalt in Tabellenform ausgeben
280 '
290 DEF PROC Print_Bits(Register%)
300     XBIOS (Wert%,28,0,Register%)
310     USING "*0 | # | # | # | # | # | # | # |"
320     PRINT RIGHT$( STR$( VAL( BIN$(Wert%))),31)
330 RETURN
```

Abb. 3.5: Auslesen der PSG-Registerinhalte

Kapitel 4: Der Multifunktionsbaustein MFP 68901

Der MFP (Multi Function Peripheral – Peripherie-Baustein mit Mehrfach-Funktion) ist ein "Familienmitglied" der MC 68000-Serie und arbeitet deshalb sehr eng mit der 68000er CPU zusammen. Im ATARI ST übernimmt der MFP eine Reihe von Aufgaben.

USART

Der USART (Universal Synchronous/Asynchronous Receiver/Transmitter = Universeller Synchron-/Asynchron Empfänger/Sender) bedient die serielle Schnittstelle des ST. Näheres über diese Funktionseinheit des MFP ist bei der Beschreibung der seriellen Schnittstelle des ST zu finden.

8-Bit-Parallelport

Der MFP besitzt 8 I/O-Anschlüsse (den General Purpose Input/Output Interrupt Port = GPIP). Jeder Anschluß kann für sich als Ein- bzw. Ausgang geschaltet werden. Beim ST sind jedoch alle acht Anschlüsse als Eingänge programmiert und auch belegt. Außerdem kann jeder als Eingang geschaltete Anschluß des MFP auch als Interrupt-Eingang benutzt werden. Durch entsprechende Programmierung kann eingestellt werden, ob ein Interrupt bei steigender oder fallender Eingangs-Signalflanke ausgelöst werden soll.

Der Parallelport wird mit drei Registern zu je 8 Bit Breite gesteuert (siehe auch Übersichtsbild des MFP-GPIP, Abbildung 4.1). Diese Register haben folgende Funktionen:

Adresse	Zugriff	Funktion des Registers	Label
$FF FA01	R	GPIP-Data-Register (GPIP)	"GPIP"
		Das Auslesen dieses Register liefert die log. Pegel der entsprechenden Eingangsanschlüsse.	
		Bit 0 stellt den Pegel auf der BUSY-Leitung der parallelen Schnittstelle des ST dar. Bei nicht angeschlossenem Drucker ist dieses Bit gesetzt (Anschluß liegt über Pull-Up-Widerstand an +5V).	

Adresse	Zugriff	Funktion des Registers	Label
$FF FA01	R	GPIP-Data-Register (GPIP)	"GPIP"

Die *Bits 1, 2 und 6* (= Inputs I1, I2 und I6) werden im ST für das Handshaking mit der seriellen Schnittstelle benutzt.

Der *Eingang I3* wird nur in STs mit Blitter benutzt. Er ist für das "GPU-Done"-Signal vom Blitter-Chip vorgesehen, der hier mit einem Low meldet, wenn eine BitBlt-Operation abgeschlossen wurde.

Mit *Bit 4* melden die beiden ACIAs (Asynchronous-Communication-Interface-Adapter) unter anderem den Zeichenempfang von der Tastatur oder der MIDI-Schnittstelle. Tritt ein Interrupt von den ACIAs auf, wird die Leitung I4 auf Low gezogen.

Welcher ACIA mit seiner IRQ-Leitung "geläutet" hat, wird vom Betriebssystem durch Abfrage der beiden ACIA-Statusregister ermittelt.

Port I5 ist wichtig für die Überwachung des Floppy-Disk-Controllers. Durch Abfrage des Bit 5 wird getestet, ob der Floppy-Controller einen Befehl beendet hat.

Außerdem melden hierüber evtl. am ACSI-Bus angeschlossene TARGETs den Abschluß eines Datenaustauschs über diesen schnellen DMA-Bus.

Ein Low an diesem Eingang signalisiert also einen Interrupt durch den FDC oder den ACSI-Bus.

Eingang I7 "lauscht" an der Monochrom-Detect-Leitung der Monitor-Buchse. Ist der Monochrom-Monitor angeschlossen, liegt Eingang I7 auf Low-Potential.

Wird z. B. im Monochrom-Betrieb der Stecker des Monitors abgezogen und ein Farbmonitor angeschlossen, so wird dies vom Betriebssystem im Vertikal-Blank-Interrupt festgestellt und die neue Betriebsart eingestellt (über eine Neuinitialisierung des Systems).

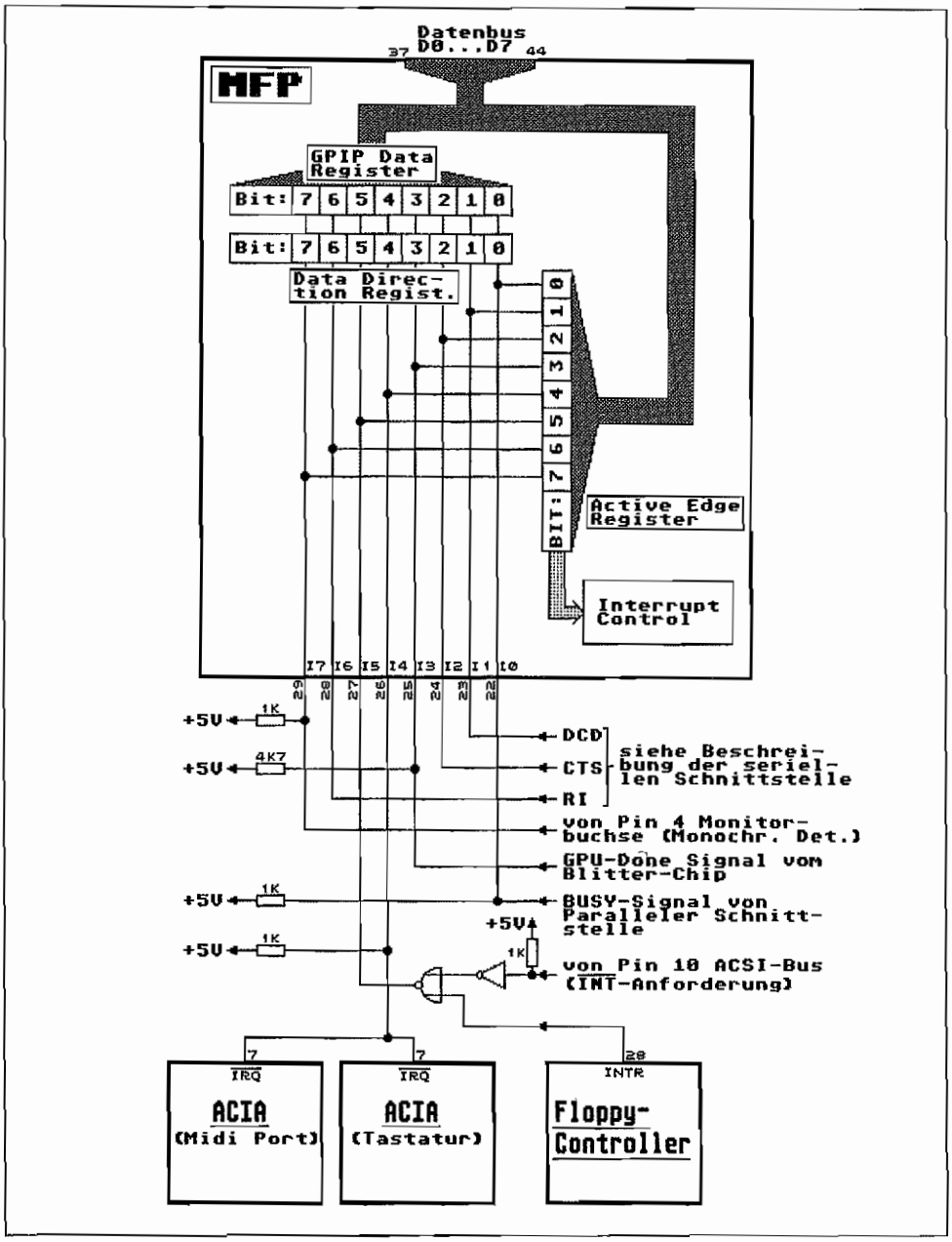


Abb. 4.1: Der General-Purpose-I/O-Port (GPIP) des MFP (MC68901) im ATARI ST

Adresse	Zugriff	Funktion des Registers	Label
$FF FA03	R/W	Active-Edge-Register (AER)	"AER"
		Hiermit läßt sich einstellen, wann durch einen der acht Eingänge ein Interrupt ausgelöst wird. Bit 0 ist hierbei dem Anschluß I0 zugeordnet, Bit 1 dem Anschluß I1 usw. Bei gelöschtem Bit erfolgt ein Interrupt bei einem High -> Low-Übergang, bei gesetztem Bit wird bei Low-> High-Übergang unterbrochen. Die zugehörigen Interrupts werden mit den Namen der Ports bezeichnet. Interrupt I1 wird also durch Port I1 ausgelöst, Interrupt I2 durch Port I2 usw. Zu der Interruptbehandlung des MFP jedoch später mehr.	
$FF FA05	R/W	Data-Direction-Register (DDR)	"DDR"
		Im DDR wird festgelegt, welche der acht I/O-Anschlüsse als Eingang und welche als Ausgang arbeiten. Im Gegensatz zum Soundchip (PSG) kann hier für *jeden* einzelnen Portanschluß die Datenrichtung festgelegt werden. Ein gesetztes Bit bedeutet hierbei, daß der zugehörige Portanschluß als Ausgang arbeitet. Bei einem gelöschten Bit ist also der entsprechende Port als Eingang programmiert. Beim ST sind alle Ports als Eingang geschaltet. Alle Bits sind also gelöscht.	

Timer

Der MFP besitzt vier Timer (Zeitgeber) zu je acht Bit. Gesteuert werden die Timer von einem eigenen Oszillator. Man hat die Möglichkeit, einen Quarz zwischen die Anschlüsse XTAL0 und XTAL1 zu "setzen" oder über XTAL0 einen Takt mit TTL-Pegel einzuspeisen.

Die Timer können damit völlig unabhängig von dem am CLK-Eingang des MFP anliegenden Takt oder sonstigen im System verwendeten Takten arbeiten. (Der CLK-Takt dient lediglich der Steuerung des internen Timing im MFP und muß nicht unbedingt der Systemtakt sein oder zu diesem in Phase liegen). Der Ausgang des Timer-Takt-Oszillators speist alle vier Vorteiler der Timer des MFP mit einer Frequenz von 2,4576 MHz.

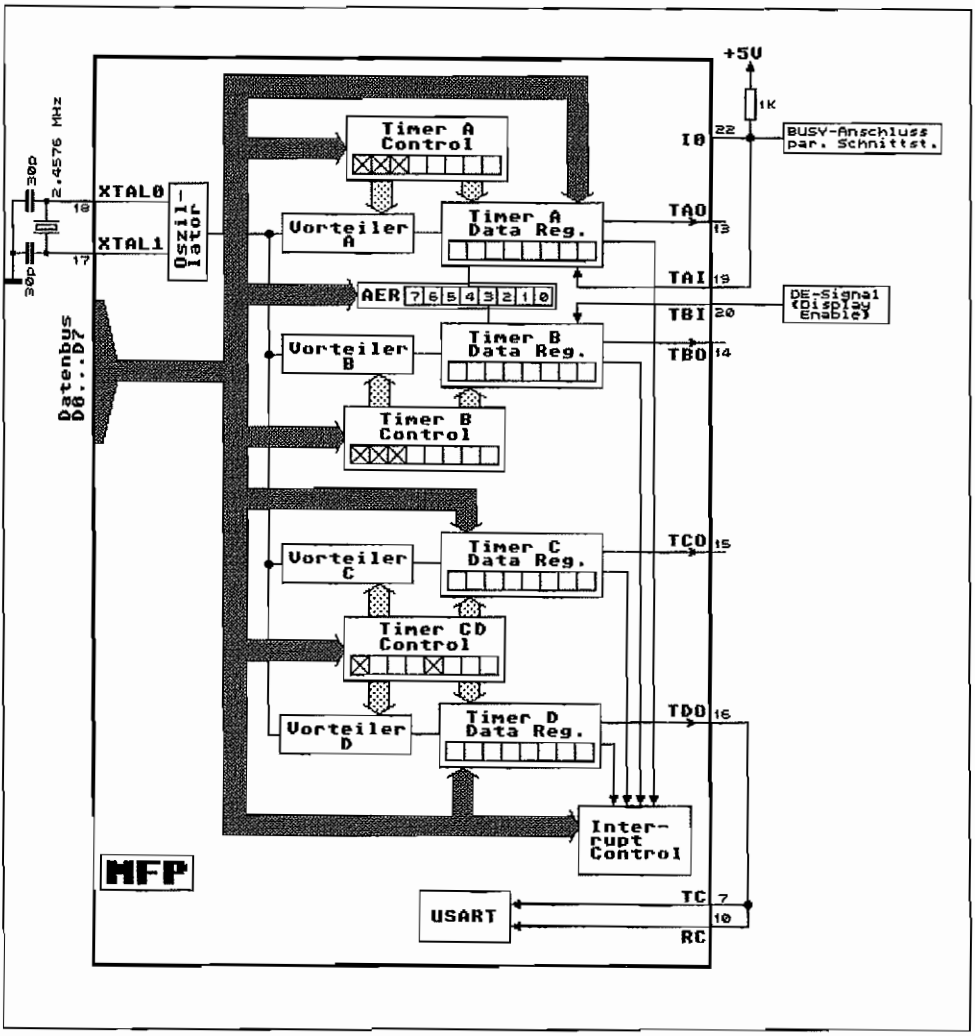


Abb. 4.2: Die Timer des MFP (MC68901) im ATARI ST

Für jeden der vier Timer ist ein eigener Interruptkanal vorgesehen. Damit kann z.B. nach einer mit dem Timer voreingestellten Zeitdauer (= Zahl von Taktzyklen) ein Interrupt ausgelöst werden.

Die Timer des MFP können in verschiedenen Betriebsarten arbeiten. Die Funktionsweisen werden nun kurz vorgestellt.

Reaktion mit Verzögerung – Der Delay Mode

Alle vier Timer können als Verzögerungs-Timer arbeiten. In diesem Delay Mode wechseln die Timer-Ausgänge ihren log. Zustand, wenn eine bestimmte, programmierbare Zahl von Taktzyklen nach Starten des Timers abgelaufen ist. Außerdem wird dann ein Interrupt für diesen Timer ausgelöst (sofern dieser aktiviert ist). Im Delay Mode arbeiten die Timer nach folgendem Prinzip:

- Der MFP-Timer-Oszillator liefert den 2,4576 MHz-Arbeitstakt an die programmierbaren Vorteiler. Die auf die Vorteilerstufen folgenden Abwärtszähler erhalten immer dann einen Impuls von der Vorteilerstufe, wenn diese die eingestellte Anzahl von Taktzyklen "gesehen" hat. Bei einem Vorteilerverhältnis von 1:10 gelangt also nach je zehn Taktzyklen ein Ausgangsimpuls zum Abwärtszähler.

- Jeder Impuls vom Vorteiler erniedrigt den Zählerstand im Abwärtszähler um 1. Hat der Abwärtszähler einen Zählerstand von 1 erreicht, wird durch den nächsten Impuls vom Vorteiler der log. Zustand des Timer-Ausgangs geändert. Gleichzeitig wird der Abwärtszähler wieder auf einen Anfangszählerstand gesetzt (dieser Anfangswert wird im Timer-Data-Register eingestellt) und ein Interrupt für den entsprechenden Timer ausgelöst (sofern dieser zugelassen ist).

Dazu ein Beispiel:

Wenn der Vorteiler mit einem Teilerfaktor von 1/64 eingestellt wird, erhält der Abwärtszähler alle 64 Taktzyklen einen Impuls. Bei einer Taktfrequenz von 2,4576 MHz geschieht das somit alle 64/2 457 600 = 26,0417 µSek.

Ist der Abwärtszähler über das Timer-Data-Register mit einem Wert von 64 geladen worden, so wechselt der Timerausgang seinen Zustand nach 64 Impulsen vom Vorteiler (= 64 x 26,0417 µSek. = 1,6667 mSek.), und der Abwärtszähler wird wieder mit dem Wert aus dem Timer-Data-Register (im Beispiel = 64) nachgeladen. Nach weiteren 64 Impulsen vom Vorteiler (also nach 1,6667 mSek.) wechselt der Timerausgang wieder auf den Anfangszustand zurück, so daß eine volle Periode am Timerausgang 2 x 1,6667 mSek. = 3,333 mSek. dauert. Die erzeugte Ausgangsfrequenz beträgt dann 1 / 3,333 mSek. = 300 Hz.

Grenzwerte im Timer-Data-Register

Gibt man dem Timer-Data-Register einen Anfangszählerstand von 1, so wechselt der Timerausgang mit jedem Impuls vom Vorteiler seinen Zustand. Wird der Anfangszählerstand dagegen auf 0 eingestellt, so sind 256 Impulse vom Vorteiler erforderlich, bevor sich am Timerausgang etwas "bewegt".

Mit den Kombinationsmöglichkeiten, die der Vorteiler und der Abwärtszähler des MFP im Delay Mode bieten, sind beim ST so Verzögerungszeiten von 1,6276 µSek. bis 20,8333 mSek. zu realisieren. Durch die "krumme" Quarzfrequenz lassen sich insbesondere die gebräuchlichsten Taktfrequenzen für die serielle Datenübertragung einstellen!

Am Puls des Geschehens – Die Pulsbreitenmessung

Während der Delay Mode mit allen vier Timern möglich ist, können die Timer A und B noch ein wenig mehr. Wie aus dem Übersichtsbild ersichtlich ist (Abbildung 4.2), besitzen diese beiden Timer zusätzlich noch je einen externen Eingang TAI (Timer-A-Input) bzw. TBI (Timer-B-Input). Mit Hilfe dieses Eingangs läßt sich die Zeitdauer eines Ereignisses feststellen.

Für die Dauer des Ereignisses muß ein entsprechender "Ein"-Pegel an dem TAI- oder TBI-Anschluß anliegen. Beim ST ist jedoch lediglich der Timer-A-Eingang (TAI) ohne weiteres von außen zugänglich. Er ist, wie aus Abbildung 4.2 ersichtlich, mit dem BUSY-Anschluß (Pin 11) der parallelen Schnittstelle verbunden.

Der Timer verhält sich in der Betriebsart "Pulsbreitenmessung" ähnlich wie im Delay Mode (der programmierbare Vorteiler liefert auch hier Impulse an den nachfolgenden Abwärtszähler). Während Ein-Pegel anliegt, zählt der Abwährtszähler im Timer A (bzw. B) wie im Delay Mode im Takt der vom Vorteiler eingehenden Impulse herunter. Geht der TAI-Anschluß (bzw. TBI) auf Aus-Pegel, wird der Zählvorgang gestoppt.

Da man den Startwert des Abwärtszählers ja über das Timer-Data-Register zu Beginn der Pulsbreitenmessung eingestellt hat, kann man nun über die Zahl der Impulse, die der Zähler bis zum Stoppen heruntergezählt hat, die Zeit ermitteln, für die Anschluß TAI (bzw. TBI) auf Ein-Pegel lag.

Was ist für den Timer denn nun Ein-Pegel?

Was am Timer-Eingang TAI (bzw. TBI) als Ein-Pegel gelten soll, wird durch Bit 4 (für Timer A) bzw. Bit 3 (für Timer B) im Aktive-Edge-Register (Label "AER" an Adresse $FF FA03) des MFP eingestellt. Ist das entsprechende *Bit gesetzt*, so wertet der Timer-Eingang ein *High-Signal als Ein-Pegel*, und dementsprechend wird *bei gelöschtem Bit* ein *Low-Signal als Ein-Pegel* angesehen.

Mit den Bits des AER wird ja sonst festgelegt, mit welcher Signalflanke an den als Eingängen geschalteten I/O-Ports (I0...I7) des MFP ein Interrupt ausgelöst werden soll (siehe Beschreibung des 8-Bit-Parallelports, Active-Edge-Register).

Die zugehörigen Portanschlüsse I3 und I4 können in der Betriebsart Pulsbreitenmessung der Timer A und B als I/O-Ports weiterverwendet werden. Interrupts lassen sich darüber nicht mehr auslösen! Die Interruptkanäle I4 und I3 reagieren jetzt nur auf die Signalflanken am Timer-Eingang TAI bzw. TBI! Ein Interrupt wird jetzt ausgelöst, wenn ein Flankenwechsel an dem entsprechenden Timer-Eingang von Ein- auf Aus-Pegel stattfindet. Hierzu ein Beispiel:

- Timer A sei auf Betriebsart "Pulsbreitenmessung" programmiert.
- Der Vorteiler A ist auf 1/10 eigestellt.
- Timer-A-Data-Register und somit der Abwärtszähler des Timers A seien mit einem Startwert von 200 geladen.
- Bit 4 im Active-Edge-Register sei gelöscht (TAI erkennt also Ein-Pegel bei Low).
- Timer A wird gestartet durch Low an TAI (BUSY-Anschluß der parallelen Schnittstelle).
- Stop für Timer A bei High an TAI und Auslösung eines I4-Interrupts (wenn dieser zugelassen ist).
- Der Inhalt des Timers A wird aus dem Timer-A-Data-Register ausgelesen und lautet beispielsweise auf 100 (Stopwert).

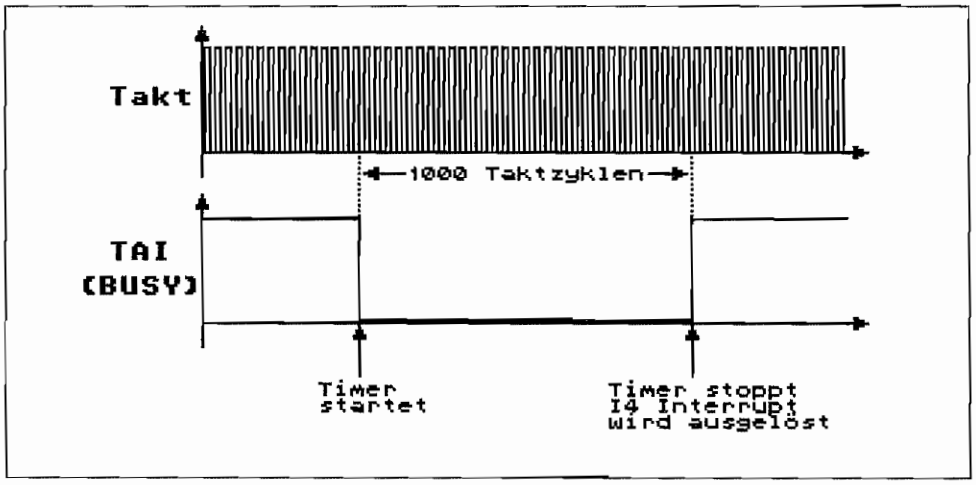


Abb. 4.3: Die Pulsbreitenmessung mit Timer A

Die Zeitdauer, für die TAI (der BUSY-Anschluß) auf Low war, beträgt also:

$$\frac{\text{(Startwert – Stopwert)}}{\text{Vorteiler x Taktfreq.}} = \frac{(200 - 100)}{0{,}1 \text{ x } 2{,}4576 \text{ MHz}} = 407 \text{ µSek.}$$

Man kann aber nun nicht nur Ereignisdauern erfassen, die im Wertebereich des 8-Bit-Abwärtszählers liegen. Sollte nämlich während der Pulsbreitenmessung der Abwärtszähler "durch" 1 zählen, so wird, genau wie im Delay-Modus, der Timerausgang TAO (oder TBO) seinen Ausgangspegel ändern, der Abwärtszähler mit dem Wert aus dem Timer-Data-Register neu geladen und ein Timer-Interrupt auf dem entsprechenden Interruptkanal ausgelöst (sofern dieser aktiviert ist). Mit diesem Timer-Interrupt läßt sich nun eine Interruptroutine ansteuern, die mitzählt, wie oft der Abwärtszähler während des Ein-Pegels an TAI (oder TBI) (= Dauer des Ereignisses) komplett heruntergezählt wurde.

Die Gesamtzahl der Zählimpulse (und damit die Ereignisdauer), für die der Timer "Ein" geschaltet war, ist damit wie folgt zu berechnen:

(Zahl der Timer-Interrupts x Anfangswert des Timer-Data-Registers)
plus (Differenz aus Anfangs- und Endwert im Timer-Data-Register)

Betrieb als Ereigniszähler

Die Timer A und B kennen noch eine weitere Betriebsart, den Event-Count-Mode. Dabei wird mit den Timern jedoch keine Zeit mehr gemessen, sondern die Zahl von aufgetretenen Ereignissen gezählt.

Hierfür werden, wie bei der Pulsbreitenmessung, die Timer-Eingänge TAI bzw. TBI benötigt. Die Interrupts I4 bzw. I3 des MFP reagieren auch hier wieder nicht auf die Signale an den Parallel-Ports I4 bzw. I3 des MFP, sondern, wie bei der Pulsbreitenmessung, auf Flankenwechsel an den TAI- bzw. TBI-Anschlüssen. Beim Betrieb als Ereigniszähler sind die Vorteiler A bzw. B jedoch nicht aktiviert (es sollen ja auch keine Zeitimpulse gezählt werden!).

Der Abwärtszähler im Timer erhält also seinen Zählimpuls nicht mehr vom Vorteiler, sondern direkt durch einen Pegelwechsel am TAI- bzw. TBI-Anschluß! Jeder Zählimpuls erniedrigt den Abwärtszähler im Timer um 1. Hat der Abwärtszähler einen Zählerstand von 1 erreicht, so wird durch den nächsten Impuls an TAI bzw. TBI der Timerausgang TAO bzw. TBO seinen Pegel ändern und ein entsprechender Timer A- bzw. Timer B-Interrupt ausgelöst, wenn dieser freigegeben ist. Außerdem wird der Abwärtszähler des Timers wieder neu mit dem Startwert aus dem Timer A- bzw. Timer B-Data Register geladen.

Ob der Zählimpuls für den Abwärtszähler bei einem Low/High- oder High/Low-Übergang am Timereingang erzeugt wird, läßt sich auch hier wieder durch Setzen bzw. Löschen des Bit 4 bzw. Bit 3 im Aktiv-Edge-Register (Label "AER" an Adr. $FF FA03) festlegen.

Ein gesetztes Bit 4 bzw. Bit 3 bedeutet, daß ein Zählimpuls erzeugt wird, wenn an TAI bzw. TBI ein Pegelwechsel von Low->High erfolgt. Bei gelöschtem Bit wird ein Zählimpuls bei High/Low-Wechsel des Timereingangs erzeugt. Außerdem wird mit jedem erzeugten Zählimpuls ein Interrupt für den entsprechenden Interruptkanal I3 bzw. I4 erzeugt (sofern zugelassen). Es ist aber sinnvoll, den Interrupt für den entsprechenden TAI- bzw. TBI-Anschluß zu sperren. Das Zählen der Ereignisse (Low/High- bzw High/Low-Pegelwechsel) am Timereingang könnte ja sonst gleich durch die Interrupt-Routine erfolgen und nicht durch den Abwärtszähler im Timer!

Timer B als Bildschirmzeilenzähler

Im ST liegt am TBI-Anschluß des MFP das Signal Display-Enable an. Dieses Signal geht so lange auf High, wie eine Bildschirmzeile geschrieben wird. So läßt sich mittels Ereigniszähler z. B. nach Ablauf einer programmierten Zahl von Bildschirmzeilen ein Interrupt auslösen, um dann eine Display-Manipulation durchzuführen.

Die Timer-Register

Die Programmierung der vier Timer erfolgt über vier 8 Bit breite Timer-Data-Register und drei 8 Bit breite Timer-Control-Register. Timer A und B besitzen je ein eigenes Control-Register. Für die Steuerung von Timer C und D reicht ein gemeinsames 8-Bit Control Register aus, da diese Timer ja nur eine Betriebsart, den Delay Mode, kennen.

Adresse	Zugriff	Funktion des Registers	Label
$FF FA1F	R/W	Timer-A-Data-Register	"TADR"
		Beim Auslesen erhält man den augenblicklichen Zählerstand des Timers A. Wird ein Wert in das Register geschrieben, so gelangt dieser gleichzeitig in den Abwärtszähler. Dies gilt jedoch nur für den Fall, daß der Timer gestoppt ist!	
		"Läuft" dagegen gerade ein Zählvorgang, so gelangt der neu eingeschriebene Wert erst in den Abwärtszähler des Timers, wenn dieser durch 1 zählt.	

Adresse	Zugriff	Funktion des Registers	Label
$FF FA21	R/W	Timer-B-Data-Register	"TBDR"
		Funktion wie bei Timer A für Timer B.	
$FF FA23	R/W	Timer-C-Data-Register	"TCDR"
		Funktion wie bei Timer A für Timer C.	
$FF FA25	R/W	Timer-D-Data-Register	"TDDR"
		Funktion wie bei Timer A für Timer D.	
$FF FA19	R/W	Timer-A-Control-Register "TACR"	

Bit 0..3 dieses Registers bestimmen Betriebsart des Timers A und Teilerverhältnis von Vorteiler A. Bits 5..7: keine Funktion.

Bit-Nr. 3 2 1 0	Funktion	
0 0 0 0	Timer gestoppt (*)	
0 0 0 1	Delay Mode, Vorteiler auf	1:4
0 0 1 0	Delay Mode, Vorteiler auf	1:10
0 0 1 1	Delay Mode, Vorteiler auf	1:16
0 1 0 0	Delay Mode, Vorteiler auf	1:50
0 1 0 1	Delay Mode, Vorteiler auf	1:64
0 1 1 0	Delay Mode, Vorteiler auf	1:100
0 1 1 1	Delay Mode, Vorteiler auf	1:200
1 0 0 0	Ereigniszählung	
1 0 0 1	Pulsbreitenmessung, Vorteiler auf	1:4
1 0 1 0	Pulsbreitenmessung, Vorteiler auf	1:10
1 0 1 1	Pulsbreitenmessung, Vorteiler auf	1:16
1 1 0 0	Pulsbreitenmessung, Vorteiler auf	1:50
1 1 0 1	Pulsbreitenmessung, Vorteiler auf	1:64
1 1 1 0	Pulsbreitenmessung, Vorteiler auf	1:100
1 1 1 1	Pulsbreitenmessung, Vorteiler auf	1:200

(*)= *Zählvorgang wird unterbrochen. Der Inhalt des Timer-A-Abwärtszählers bleibt erhalten. Wird jetzt ein neuer Wert ins Timer-A-Data-Register geschrieben, so gelangt dieser auch sofort in den Abwärtszähler!*

Adresse	Zugriff	Funktion des Registers	Label
		Ist durch Einschreiben in das Register das Bit 4 gesetzt, geht der Timer-A-Ausgang auf Low-Pegel und bleibt für die Dauer der Schreiboperation Low. Danach kann der Ausgang abhängig vom Zählvorgang (Abwärtszählen durch 1) wieder seinen Zustand wechseln.	
$FF FA1B	R/W	Timer-B-Control-Register	"TBCR"
		Die Funktion und Bitbelegung ist die gleiche wie für das Timer-A-Control-Register.	
$FF FA1D	R/W	Timer-C/D-Control-Register	"TCDCR"
		Die Teilerverhältnisse für die Vorteiler C + D werden durch dieses Register kontrolliert. Die Bits 0..2 steuern Timer D, die Bits 4..6 den Timer C.	

Bit-Nr. 6 5 4 (C) oder 2 1 0 (D)	**Funktion**	
0 0 0	Timer gestoppt (*)	
0 0 1	Delay Mode, Vorteiler	1:4
0 1 0	Delay Mode, Vorteiler	1:10
0 1 1	Delay Mode, Vorteiler	1:16
1 0 0	Delay Mode, Vorteiler	1:50
1 0 1	Delay Mode, Vorteiler	1:64
1 1 0	Delay Mode, Vorteiler	1:100
1 1 1	Delay Mode, Vorteiler	1:200

(*)= *Zählvorgang wird unterbrochen. Der Inhalt des Timer-Abwärtszählers bleibt erhalten. Wird jetzt ein neuer Wert ins Timer-Data-Register geschrieben, so gelangt dieser auch sofort in den Abwärtszähler!*

Achtung bei der Timerprogrammierung

Bei der Timerprogrammierung sollte man sich vorher schon genau im klaren sein, was man erreichen will und wie der Timer sich verhalten soll. So muß man z. B. mit falschen Ergebnis-

sen rechnen, wenn das Vorteilerverhältnis geändert wird, während der Timer "läuft". Weiterhin können in der Betriebsart "Pulsbreitenmessung" und "Ereigniszählung" Interrupts auftreten, wenn man die Flankenbits im Active-Edge-Register ändert, während der Timer arbeitet.

Bei der Betriebsart "Pulsbreitenmessung" sollte man auch darauf achten, daß einem nicht der Timer schon wieder neu gestartet wird (durch Ein-Pegel am Timer-Eingang), bevor man den Wert im Abwärtszähler für eine neue Messung initialisiert hat. Im Betriebssystem des ST ist für das Setzen der Timerregister des MFP ein eigener XBIOS-Aufruf vorgesehen (siehe XBIOS-Funktion #31, "Xbtimer"). Damit wird die Timerprogrammierung wesentlich vereinfacht.

Bevor nun auf die Interrupt-Bearbeitung des MFP eingegangen wird, zunächst einige Erläuterungen zur Interruptbehandlung durch die 68000er CPU.

Interrupt-Steuerung

Die im ST verwendete 68000er-CPU besitzt "von Haus aus" die Möglichkeit, Unterbrechungen (Interrupts) sieben verschiedene Prioritäten (Interrupt-Level) zuzuordnen.

Interrupts des Level 7 besitzen höchste Priorität und sind auch nicht maskierbar, d. h., sie können wiederum durch einen Interrupt der Priorität 7 unterbrochen werden.

Die Interrupt-Level 1..6 sind maskierbar und können bei entsprechender Programmierung der Unterbrechungsmaske in der CPU (Bits 8..10 im Supervisor-Status-Register) nur durch einen höher priorisierten Interrupt unterbrochen werden.

Ein in Bearbeitung befindlicher Interrupt setzt die Unterbrechungsmaske dabei automatisch auf seinen Interrupt-Level, d. h., er kann nur durch einen Interrupt höherer Priorität unterbrochen werden.

Der Level 0 bedeutet, daß keine Interrupt-Anforderung gestellt wurde.

Interrupt-Prioritäten im ST

Die Interrupt-Priorität der unterbrechenden Einheit wird der CPU über drei Interrupt-Priority-Level-Anschlüsse IPL0..IPL2 (low-aktiv) "mitgeteilt". Im ST liegt die Leitung IPL0 jedoch fest auf High, so daß nur die Interrupt-Ebenen 2, 4 und 6 existieren (Beim MEGA-ST sind die Interrupts 3, 5 und 7 auch möglich. Sie können jedoch nur über Anschlüsse am Megabus ausgelöst werden. Siehe auch Teil II, Kapitel 1, "Die Zentraleinheit").

Die *Interrupt-Ebene 2* wird im ST durch den Horizontal-Blanking-Interrupt (HBLANK-Interrupt) belegt. Dieser Interrupt tritt immer dann auf, wenn eine Bildschirmzeile geschrieben wurde und der Elektronenstrahl im Monitor für den Strahlrücklauf zum Anfang einer neuen Zeile dunkelgetastet (ge"blanked") wird. Um jedoch zu vermeiden, daß ein laufendes Programm durch jeden Horizontal-Blankingimpuls unterbrochen und damit verlangsamt wird, ist die Unterbrechungsmaske der CPU im ST im Normalfall auf 3 gesetzt. Der HBLANK-Interrupt ist somit maskiert und wird nicht ausgeführt.

Die bei der Systeminitialisierung installierte HBLANK-Interrupt-Routine macht nichts anderes, als die Unterbrechungsmaske in der CPU auf 3 zu setzen, um weitere auftretende HBLANK-Interrupt-Anforderungen unwirksam zu machen.

Programmierer, die sich den HBLANK-Interrupt zunutze machen wollen, um z. B. die Farbregisterinhalte zwischen zwei Bildschirmzeilen zu ändern, sollten darauf achten, daß zwischen zwei HBLANK-Interrupts nur 64 µSek. (28 µSek. in Monochrom-Betriebsart) liegen. Die Interruptroutine hat also nicht viel Zeit, um ihre Aufgabe zu erledigen. Wenn dann noch der Systemtimer-Interrupt "dazwischenrutscht" (Timer C "interrupted" alle 5 msec.), kann es zu einer zitternden, unruhigen Bildschirmdarstellung kommen.

Man kann sich mit dem Systemtimer-Interrupt jedoch dadurch helfen, indem man für die Zeit, in der mit HBLANK-Interrupts gearbeitet wird, den Interruptvektor für den Systemtimer zu einer ganz kurzen Routine umleitet, die nichts anderes macht, als die zwischenzeitlich auftretenden Systemtimer-Interrupts zu zählen.

Sind die HBLANK-Interrupts dann wieder gesperrt, so blockiert man zunächst durch entsprechende Maskierung (Interrupt-Priority-Level auf 6) in der CPU alle weiteren Interrupts. Dann wird die Interrupt-Routine für den Systemtimer so oft aufgerufen, wie dieser Interrupt während der "kritischen" Zeit aufgetreten ist (die Anzahl der Aufrufe wurde ja in der kleinen Systemtimer-Interrupt-"Hilfsroutine" mitgezählt).

Die *Interrupt-Ebene 4* ist für den Vertical-Blank-Interrupt (VBLANK-Interrupt) reserviert. Dieser Interrupt tritt in der Monochrom-Betriebsart alle 14 mSek. und in der Color-Betriebsart alle 20 mSek. auf. Zu weiteren Einzelheiten bezüglich dessen Aufgabe siehe Teil I, Kapitel 1, "Der Vertical Blank Handler".

Selbstgemachte Interrupt-Vektornummern

Sowohl bei dem HBLANK- als auch dem VBLANK-Interrupt handelt es sich um einen sogenannten Autovektor-Interrupt. Bei so einem Interrupt erzeugt die CPU intern, sozusagen "automatisch", eine Interrupt-Vektornummer. Diese Autovektor-Nummer ergibt sich dabei

aus der Priorität des Interrupts. Daß ein Autovektor-Interrupt aufgetreten ist, erfährt die CPU durch ein Low über den VPA-Anschluß (Valid-Peripheral-Address) während der Phase der Unterbrechungsanerkennung. Aus dieser Autovektor-Nummer wird dann die Interrupt-Vektor-Adresse berechnet (Autovektor-Adresse=Autovektornummer * 4). Unter der Interrupt-Vektor-Adresse findet die CPU dann einen Zeiger auf die abzuarbeitende Interruptroutine, die dann aufgerufen wird.

Für Interrupts der Priorität 2 (HBLANK-Interrupt) wird eine Vektornummer von 26 erzeugt. Der Zeiger auf die HBLANK-Interrupt-Routine befindet sich also an Adresse $68. Der VBLANK-Interrupt (Priorität 4) erzeugt die Vektornummer 28. Die Interrupt-Vektor-Adresse lautet somit auf $70.

Interrupts durch den MFP-Baustein

Ein Interrupt, der durch den MFP ausgelöst werden kann, liegt auf Level 6, hat also im ST nahezu höchste Priorität. Im MFP-Baustein selbst werden nochmals 16 Interruptkanäle unterschieden, denen MFP-intern 16 verschiedene Prioritäten zugeordnet sind. Wie die folgende Tabelle zeigt, hat z. B. der Interrupt des I/O-Port 7 (I7, Monochrom-Monitor-Detect) im MFP höchste Priorität. Das BUSY-Signal (I0) der parallelen Schnittstelle würde,wäre der Interrupt im ST aktiviert, dagegen einen Interrupt niedrigster Priorität bei der MFP-Logik auslösen.

Kanal Nr.:	Interrupt durch	Bedeutung im 'ST'
15	I/O-Port 7 (I7)	Monochrom Monitor Detect
14	I/O-Port 6 (I6)	RI von ser. Schnittstelle
13	Timer A	nicht benutzt
12	RCV Buffer full	ser. Schnittstelle Empfängerpuffer voll
11	RCV Error	ser. Schnittstelle Empfangsfehler
10	XMIT Buff. empty	ser. Schnittstelle Sendepuffer leer
9	XMIT Error	ser. Schnittstelle Sendefehler
8	Timer B	Display Enable-Signal Zähler
7	I/O-Port 5 (I5)	FDC-Ready/ACSI-Bus Steuerung
6	I/O-Port 4 (I4)	IRQ von Tastatur/MIDI-ACIA's
5	Timer C	200 Hz Systemtakt
4	Timer D	ser. Schnittstelle Baudraten-Generator
3	I/O-Port 3 (I3)	(GPU-Done Signal vom Blitter-Chip)
2	I/O-Port 2 (I2)	CTS von ser. Schnittstelle
1	I/O-Port 1 (I1)	DCD von ser. Schnittstelle
0	I/O-Port 0 (I0)	BUSY von par. Schnittstelle

Priorität: ↑ hoch — ↓ niedrig

Abb. 4.4: So sind die Interrupt-Kanäle des MFP intern gewichtet

Non-Autovektor-Interrupts

Der MFP erzeugt bei der CPU im ST einen "Non-Autovektor-Interrupt", d. h., nicht die CPU bildet eine Vektornummer, aus der dann die Lage des Interruptvektors im Speicher bestimmt wird, sondern der MFP-Baustein sendet, nachdem seine Unterbrechungsanforderung von der CPU bestätigt wurde, eine "eigene" Vektornummer zur CPU. Die CPU "erfährt" von einem Non-Autovektor-Interrupt durch ein Low am DTACK-Anschluß (Data-Transfer-Acknowledge) während der Interrupt-Anerkennungsphase.

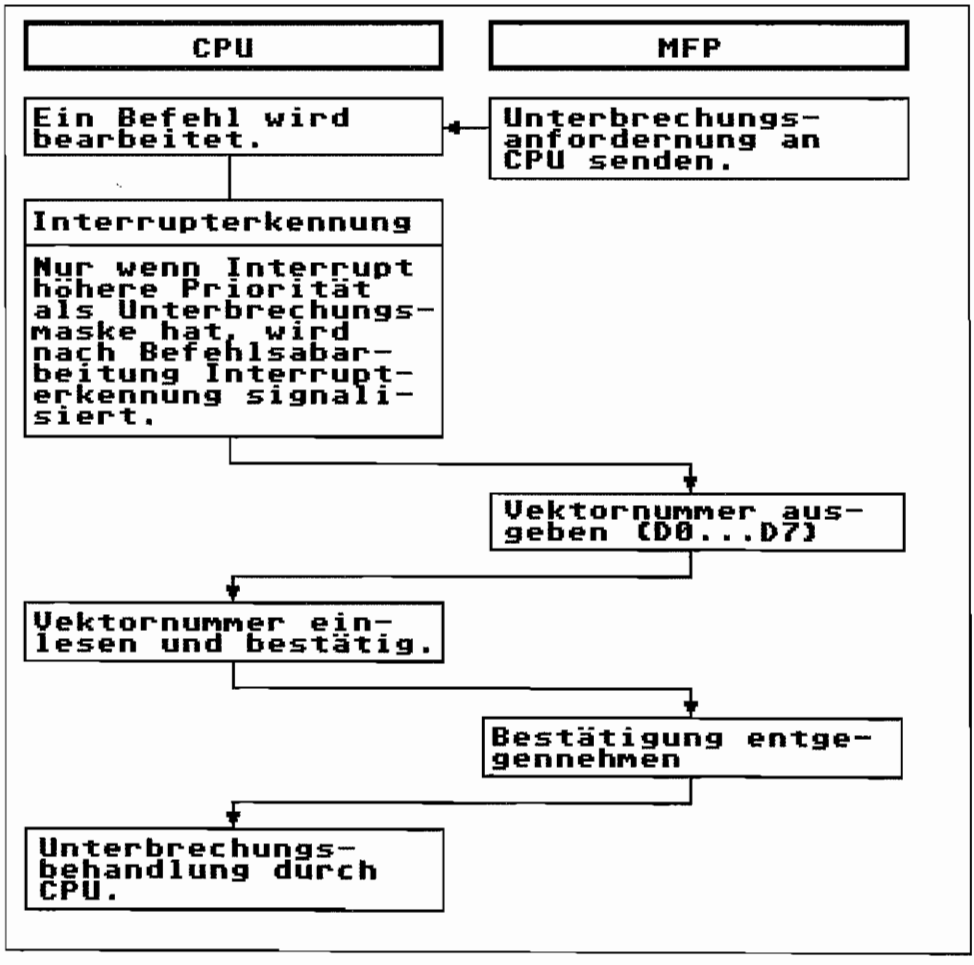


Abb. 4.5: Das Prinzip der Interrupt-Bearbeitung CPU<->MFP

Aus dieser vom MFP gelieferten Vektornummer berechnet die CPU dann die Speicheradresse, in der ein Zeiger auf den Anfang der Interrupt-Routine steht. Beim ST liegen diese 16 Zeiger (Unterbrechungsvektoren) für MFP-Interrupts ab Adresse $100 im Supervisorspeicherbereich. Es sind jedoch vom Betriebssystem nicht alle 16 Interruptkanäle des MFP aktiviert.

Beispielweise sind alle I/O-Port 0..3 (I0..I3)-Interrupts des MFP gesperrt, weil diese I/O-Anschlüsse nicht per Interrupt bedient werden, sondern evtl. durch einfaches "Polling". Das bedeutet nichts anderes, als daß vom Betriebssystem in einer Schleife oder nach Ablauf einer bestimmten Verzögerung der Zustand dieser als Eingang eingestellten Ports abgefragt wird.

So erfolgt z. B. die Zeichenausgabe an den Drucker über die parallele Schnittstelle nicht interruptgesteuert, sondern per Polling. Jedesmal wenn ein Zeichen an den Drucker gesendet werden soll, wird "nachgeschaut", ob BUSY (I/O-Port 0) noch auf High liegt. Ist der Drucker noch BUSY (BUSY auf High), so wird eine Schleife durchlaufen, in der fortlaufend der Zustand der BUSY-Leitung abgefragt wird (Polling). Erst wenn BUSY auf Low geht, gelangt das auszugebende Zeichen an die parallele Schnittstelle.

Es wird jedoch vom Betriebssystem eine "Time out"-Überwachung mittels Systemtimer (Timer C des MFP) durchgeführt, damit spätestens nach ca. 30 Sek. erfolglosen "Wartens auf BUSY" mit einer Fehlermeldung ins aufrufende Programm zurückgekehrt werden kann.

Bei einem nicht angeschlossenen Drucker oder wenn dieser "Offline" geschaltet ist, würde man sonst ewig (na ja, bis zum RESET) warten müssen.

Adresse	MFP-Int. Nummer:	Status	zeigt auf Interrupt-Routine für:	Interrupt durch
$100	0	gesperrt	BUSY-Anschluß der par. Schnittst.	I/O-Port 0 (I0)
$104	1	gesperrt	DCD-Signal serielle Schnittstelle	I/O-Port 1 (I1)
$108	2	gesperrt	CTS-Signal serielle Schnittstelle	I/O-Port 2 (I2)
$10C	3	gesperrt	(GPU-Done Signal vom Blitter Chip)	I/O-Port 3 (I3)
$110	4	gesperrt	Baudraten-Gen. ser. Schnittstelle	Timer D
$114	5	aktiv.	200 Hz System Timer	Timer C
$118	6	aktiv.	Tastatur/MIDI (ACIA) Bedienung	I/O-Port 4 (I4)
$11C	7	gesperrt	FDC-Ready/ACSI-Bus Steuerung	I/O-Port 5 (I5)
$120	8	gesperrt	Display Enable Signal (Hor.Sync.)	Timer B
$124	9	aktiv.	Sendefehler ser. Schnittstelle	XMIT-Error
$128	10	aktiv.	Sendepuffer leer (ser. Interface)	XMIT-Buf. empty
$12C	11	aktiv.	Empfangsfehler (ser. Interface)	RCV-Error
$130	12	aktiv.	Empfangspuffer voll (ser. Interf)	RCV-Buffer full
$134	13	gesperrt	nicht benutzt	Timer A
$138	14	gesperrt	RI-Signal serielle Schnittstelle	I/O-Port 6 (I6)
$13C	15	gesperrt	Monochrom Monitor Detect	I/O-Port 7 (I7)

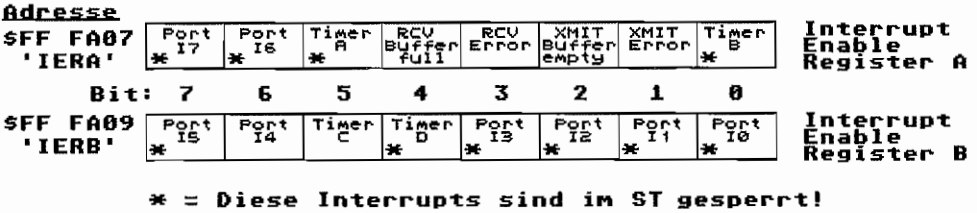
Abb. 4.6: Die Lage der MFP-Interrupt-Vektoren im ATARI ST-RAM

Einer der aktivierten Interrupts ist jener für den schon mehrfach erwähnten 200 Hz-Systemtimer, der eine Art "innere Uhr" im ST darstellt und unter anderem für Tastenwiederholung und als Takt für den "Pseudo-Soundprozessor" benutzt wird.

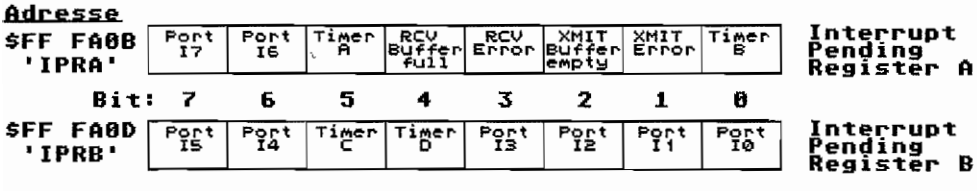
Aktiviert ist auch der Interrupt für die beiden ACIAs, welche zur Kommunikation mit der Tastatur und der MIDI-Schnittstelle dienen. Es muß ja gewährleistet sein, daß Informationen von der "Intelligenten Tastatur", wie Tasten- oder Mausbetätigungen, laufend ans Betriebssystem weitergegeben werden. Die Interrupts zur Bedienung des Sende- und Empfangsteils der seriellen Schnittstelle des MFP sind ebenfalls aktiviert.

Mit drei Steuerregistern die MFP-Interrupts im Griff

Zur Interrupt-Bearbeitung besitzt der MFP drei Steuerregister, die jeweils 2 * 8 Bit breit sind.



Alle 16 möglichen Interruptkanäle des MFP können einzeln ein- bzw. ausgeschaltet werden. Die MFP-Logik registriert nur Interrupts von Kanälen, die auch eingeschaltet sind, und gibt diese dann in Form von Unterbrechungsanforderungen an die CPU weiter. Ausgeschaltete Interruptkanäle werden nicht beachtet. Durch Setzen des entsprechenden Bits im IERA (Interrupt-Enable-Register A) oder IERB wird der zugehörige Interruptkanal eingeschaltet.



Jedem Interruptkanal des MFP ist ein Bit im Interrupt-Pending-Register A oder B (IPRA oder IPRB) zugeordnet. Empfängt die MFP-Logik einen Interrupt von einem aktivierten Interrupt-Kanal, so wird das zugehörige Bit gesetzt. Dieser Interrupt wird so lange als noch nicht anerkannt = "schwebend" (pending) geführt, bis die Unterbrechungsanforderung von der CPU bestätigt wird.

Im "Non-Autovektor-Interrupt"-Betrieb des MFP, wie es im ATARI ST der Fall ist, wird ein gesetztes Bit im IPRA oder IPRB erst dann gelöscht, wenn die CPU den Interrupt für den entsprechenden Kanal bestätigt hat und der MFP die zugehörige Vektornummer an die CPU sendet.

Die IPRA und IPRB können von der CPU jederzeit gelesen und beschrieben werden. Es ist auch möglich, Bits des IPRA oder IPRB per Software (durch Programmbefehl) zu löschen. Dazu muß für alle jene Bits, welche nicht gelöscht werden sollen, eine "1" in das Register geschrieben werden.

Die zu löschenden Bits werden als "0" geschrieben. Gelöschte Bits können jedoch nicht wieder durch Beschreiben mit einer "1" gesetzt werden! Das ist nur durch Auslösung eines Interrupts möglich. Der Wert im Interrupt-Pending-Register wird also mit dem einzuschreibenden Wert UNDiert und das Resultat im Register abgelegt.

Einzelne Bits im IPRA bzw. IPRB werden auch dann gelöscht, wenn die zugehörigen Interrupt-Kanäle im Interrupt-Enable-Register A bzw. B gesperrt werden.

Adresse									
$FF FA13 'IMRA'	Port I7	Port I6	Timer A	RCV Buffer full	RCV Error	XMIT Buffer empty	XMIT Error	Timer B	Interrupt Mask Register A
Bit:	7	6	5	4	3	2	1	0	
$FF FA15 'IMRB'	Port I5	Port I4	Timer C	Timer D	Port I3	Port I2	Port I1	Port I0	Interrupt Mask Register B

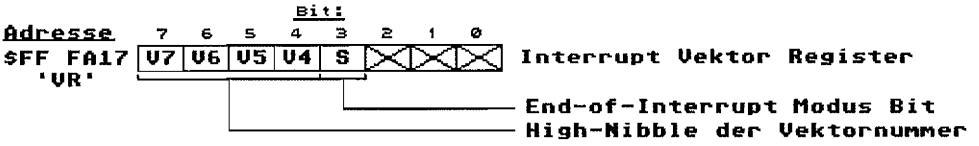
Mit Hilfe der Interrupt-Mask-Register A bzw. B (IMRA bzw. IMRB) ist es möglich, bestimmte Interruptkanäle des MFP zu maskieren (auszublenden).

Ist ein Interrupt-Kanal zwar aktiviert, aber maskiert, so wird eine evtl. Interruptsignalisierung zwar von der MFP-Logik registriert und das zugehörige "Pending"-Bit im IPRA bzw. IPRB gesetzt. Es wird jedoch keine Interrupt-Anforderung für diesen Kanal an die CPU gesendet.

Die Maskierung eines Interrupt-Kanals erfolgt durch Löschung des zugehörigen Bits. Durch Setzen eines Bits wird die Maskierung für den zugehörigen Kanal aufgehoben.

Signalisiert ein maskierter Kanal einen Interrupt zur MFP-Logik, so wird das entsprechende Bit des IPRA bzw. IPRB gesetzt (pending Interrupt). Nach Beendigung der Maskierung wird durch das gesetzte IPRA- bzw. IPRB-Bit die MFP-Logik nun bei der CPU einen Interrupt für diesen, nun nicht mehr maskierten, Kanal anfordern. Die Interrupt-Mask-Register können jederzeit beschrieben und ausgelesen werden.

Die im folgenden beschriebenen Register dienen weniger der Steuerung als der Signalisierung von MFP-Interrupts.



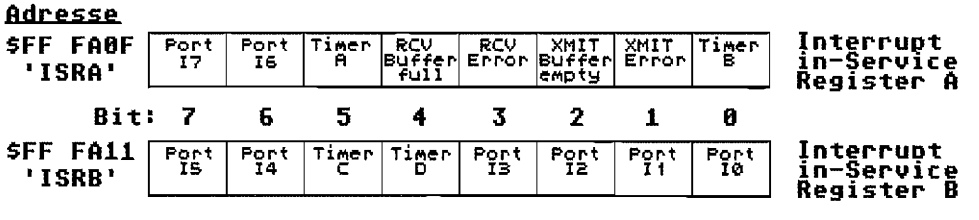
Wie bereits erläutert, sendet der MFP im "Non-Autovektor-Interrupt"-Modus nach Bestätigung der Interrupt-Anforderung durch die CPU eine dem MFP-Interrupt-Kanal entsprechende 8-Bit-Vektornummer an die CPU, damit diese daraus die Adresse des zugehörigen Interrupt-Vektors berechnen kann. Und so ist diese Vektornummer aufgebaut:

- Die unteren vier Bits (Low-Nibble) der 8-Bit-Vektornummer stellen die Nummer des Interrupt-Kanals (0...15) im MFP dar, der den Interrupt ausgelöst hat.

- Die oberen vier Bits (High-Nibble) der Vektornummer werden von der MFP- Logik aus dem Interrupt-Vektor-Register entnommen. Es handelt sich dabei um eine Kopie der oberen vier Bits dieses Registers. Im ST ist hierbei ein Wert von "0100" für die Bits 7..4 initialisiert.

Der MFP im ST kann somit mit seinen 16 Interrupt-Kanälen die Vektornummern "0100 0000" (dez. 64) bis "0100 1111" (dez. 79) erzeugen.

Das S-Bit stellt den End-of-Interrupt-Modus (EOI-Modus) ein. Bei gelöschtem Bit arbeitet der MFP im Automatic-EOI-Modus. Der Software-EOI-Modus ist bei gesetztem S-Bit aktiviert. Näheres zum EOI-Modus folgt nun bei der Beschreibung des Interrupt-in-Service-Registers.



Dieses Register ist nur von Bedeutung, wenn das S-Bit im Interrupt-Vektor-Register gelöscht ist. Der MFP arbeitet dann im Software-EOI-Modus. Im *Software-EOI-Modus* muß die CPU der MFP-Logik durch einen Befehl das Ende der Interrupt-Bearbeitung für den zur Zeit bearbeiteten Interrupt mitteilen. Dieser Modus wird im ST benutzt.

Die Interrupt-Bearbeitung läuft folgendermaßen ab (es soll noch keine Interrupt-Anforderung vorliegen):

- Ein aktivierter Kanal meldet der MFP-Logik einen Interrupt.
- Das zugehörige Bit im Interrupt-Pending-Register wird gesetzt. Der MFP sendet eine Interrupt-Anforderung an die CPU.
- Die CPU bestätigt die Anforderung, und der MFP sendet der CPU die entsprechende Vektornummer. Gleichzeitig wird das zugehörige Interrupt-Pending-Bit gelöscht.

  Dafür wird im Interrupt-in-Service-Register A bzw. B das zugehörige Bit gesetzt. Der Interrupt wird bearbeitet (der Interrupt ist "in-Service").
- Die laufende Interrupt-Bearbeitung für diesen Kanal kann nur durch einen Interrupt eines höherwertigen Kanals unterbrochen werden.
- Interruptsignalisierungen von niedriger priorisierten Kanälen werden zwar von der MFP-Logik registriert (Setzen des entspr. Bits in IPRA bzw. IPRB), führen aber nicht zu einer Unterbrechungsanforderung an die CPU.

  Weil das Interrupt-Pending-Bit für den gerade in Bearbeitung befindlichen Interrupt ja wieder gelöscht wurde (aufgrund der Interrupt-Bestätigung durch die CPU), kann schon wieder eine neue Interruptsignalisierung für diesen Kanal von der MFP-Logik registriert werden (Setzen des Interrupt-Pending-Bits). Aber auch hier wird der laufende Interrupt nicht unterbrochen.
- Nach Beendigung des laufenden Interrupts muß durch die CPU das zugehörige Bit im Interrupt-in-Service Register A bzw. B gelöscht werden. Ein Bit wird gelöscht, indem alle Bits des Registers, bis auf das zu löschende Bit, mit "1" beschrieben werden. Da Bits im Interrupt-Pending-Register per Software nur gelöscht und nicht gesetzt werden können, werden Registerinhalte durch das Beschreiben mit "1" nicht geändert.
- Erst nach Löschung des Interrupt-in-Service-Bits für den gerade beendeten Interrupt kann die MFP-Logik wieder Interrupt-Anforderungen für bereits signalisierte (Bits im IPRA od. IPRB gesetzt) Interrupts an die CPU stellen.

Im *Automatic-EOI-Modus* bleibt das Interrupt-in-Service-Register ohne Funktion. Auch hier wird nach Bestätigung der Interrupt-Anforderung durch die CPU das entsprechende Interrupt-Pending-Bit gelöscht. Nachfolgend auftretende Interrupts irgendeines Kanals des MFP (und damit jeglicher Priorität) führen zu einer Interrupt-Anforderung an die CPU, egal ob die laufende Interrupt-Bearbeitung beendet ist oder nicht.

Wie schon für die Timer des MFP sind im Betriebssystem des ST Funktionen für die Behandlung der MFP-Interrupts vorgesehen. Siehe hierzu im Teil I, Kapitel 1, "XBIOS-Referenz", die XBIOS-Funktionen #13 ("Mfpint"), #26 ("Jdisint") und #27 ("Jenabint").

So ein kleiner Chip – und doch so viele Möglichkeiten

Wie man sieht, ist der MFP-Chip im ST mit einer ganzen Reihe von Aufgaben betraut.

Einige wenige Anschlüsse und Funktionen sind, wie bereits vorher erwähnt, noch nicht benutzt. Hier bieten sich dem hardwarenahen Programmierer und dem engagierten Hobby-Elektroniker noch einige Möglichkeiten, eigene Vorstellungen zu verwirklichen.

Kapitel 5: Die serielle Schnittstelle

Wie der Name schon sagt, werden mit Hilfe der seriellen Schnittstelle Daten Bit für Bit nacheinander (seriell) über einen Übertragungsweg ausgetauscht. Ein gesetztes Bit wird dabei durch einen log. H-Pegel und ein gelöschtes Bit somit durch einen log. L-Pegel dargestellt. Für jedes Bit wird also auf die Übertragungsleitung für eine bestimmte Zeiteinheit ein dem Zustand des Bits entsprechender log. Pegel angelegt.

Die serielle Schnittstelle des ST lehnt sich dabei an den RS-232C-Standard an. Die Spannungspegel entsprechen den Vereinbarungen:

Log. L =	"Ein"-Zustand =	+3V..+15V
Log. H =	"Aus"-Zustand =	–3V..-15V

Mit dem im ST für die Steuerung der seriellen Schnittstelle zuständigen Chip ist sowohl synchrone als auch asynchrone Datenübertragung möglich. Leider kann aber das synchrone Gleichlaufverfahren nicht benutzt werden, da die dazu erforderlichen Taktleitungen nicht am Schnittstellenanschluß zur Verfügung stehen!

Nachfolgend eine kurze Erläuterung zu diesen beiden Gleichlaufverfahren.

Synchrone Betriebsart

Bei synchroner, serieller Übertragung arbeiten die beteiligten Datenübertragungseinheiten (Sende- und Empfangseinheit) ständig exakt im gleichen Takt (synchron). Dazu ist eine genaue Übereinstimmung zwischen Sender- und Empfängertakt erforderlich. An der Schnittstelle zum Datenübertragungsgerät sind deshalb Leitungen für den Sende- und Empfangstakt erforderlich.

Durch die zusätzlich vorhandenen Taktleitungen ist gewährleistet, daß bei einer Datenübertragungsverbindung beide Seiten bitsynchron arbeiten! Also sind beide Seiten auch synchron, wenn gar keine Nutzdaten übertragen werden. Damit die Gegenstelle "merkt", daß nun Nutzdaten übertragen werden und wo diese Zeichen beginnen, muß sie das an bestimmten Marken im Bitstrom erkennen können.

Dazu wird das sogenannte Synchronzeichen verwendet. Es dient dazu, die Empfangsstelle in die Zeichensynchronisation zu bringen. Sende- und Empfangseinheit müssen natürlich die "gleiche Sprache sprechen" (das gleiche Synchronzeichen benutzen).

Eine synchrone Datenübertragung beginnt immer mit der Aussendung eines Synchronzeichens. Die Empfangseinheit "sucht" zunächst im empfangenen seriellen Bitstrom nach dem typischen Bitmuster des vereinbarten Synchronzeichens. Dabei werden, je nach Vereinbarung, zum Start der Übertragung ein oder zwei Synchronzeichen gesendet. Erst wenn von der Empfangseinheit ein (oder zwei) Synchronzeichen erkannt wurde(n), "rastet" diese ein und beginnt die nachfolgenden seriellen Daten im so gefundenen Rhythmus zu decodieren.

Asynchrone Betriebsart

Die (in der ST/PC-Welt) geläufigere Art der seriellen Datenübertragung und deshalb vom TOS standardmäßig unterstützt, ist die asynchrone Betriebsart. Asynchron deshalb, weil nur für die Dauer eines Zeichens Gleichlauf zwischen Sender und Empfänger herrschen muß! Die sendende Einheit gibt nur Datenbits aus, wenn auch ein (Nutz-) Zeichen zur Übertragung vorliegt.

Da in diesem Fall Pausen auftreten können, während der Sender und Empfänger aus der Synchronisation laufen könnten (und das auch tun), muß eine Synchronisationsinformation mit jedem übertragenen Zeichen mitgeführt werden. Deshalb wird jedem Zeichen ein Startbit (log. L) vorangestellt, damit der Empfänger informiert wird, daß nun wieder ein Zeichen übertragen werden soll. Die Empfangseinheit beginnt mit dem Empfang des Startbits, den seriellen Bitstrom im Takt der eingestellten Übertragungsrate abzutasten.

Am Ende eines Zeichens werden dann noch ein oder zwei Stopbits (log. H) angehängt, um den Empfänger über das Ende der Zeichenübertragung zu informieren. Wenn der Empfänger also das Startbit und die vereinbarte Zahl von Stopbits korrekt erkannt hat, kann die Empfangseinheit davon ausgehen, daß auch das eigentliche Zeichen korrekt empfangen wurde.

Der Taktgleichlauf zwischen Sender und Empfänger ist also nur für einen relativ kurzen Zeitraum (nämlich für die Übertragungsdauer eines Zeichens + Start- und Stopbits) erforderlich und deshalb einigermaßen genau zu realisieren. Bei jedem neuen Zeichen hat der Empfänger mit dem Empfang des Startbits die Gelegenheit, sich neu zu synchronisieren.

Serielle Datenübertragung mit dem ST

Wie ja bereits zu Eingang des Kapitels erwähnt, besitzt der ST eine serielle Schnittstelle in Anlehnung an den RS-232C-Standard. Der ST ist dabei als Datenendeinrichtung (DEE) zu betrachten. Über eine TXD-Leitung (Transmitted Data) werden die Daten seriell ausgegeben; am RXD-Anschluß (Received Data) erwartet der ST die Empfangsdaten. Für den reibungslosen Ablauf der Datenübertragung stellt die serielle Schnittstelle des ST fünf Handshake-Signale zur Verfügung.

Anschluß Bezeichnung	ser. Port	Bedeutung
RTS	Pin 4	Request to send (Sendeteil einschalten) Der Anschluß geht auf "Ein", wenn das Datenendgerät (der ST) etwas senden will. Mit "Aus"-Pegel wird signalisiert, wenn nicht gesendet werden soll.
CTS	Pin 5	Clear to send (Sendebereitschaft) Rückmeldung der DÜE (Daten-Übertragungs-Einheit = Modem), daß deren Sendeteil eingeschaltet ist und die zu sendenden Daten über die TXD-Leitung vom ST ausgegeben werden können (bei "Ein"-Signal dürfen Daten vom ST gesendet werden, sonst nicht!)
DCD	Pin 8	Data carrier detect (Empfangssignalpegel liegt vor) Die DÜE meldet mit "Ein" an den ST, daß sie Signale mit ausreichendem Pegel von der Gegenstation empfängt.
DTR	Pin 20	Data terminal ready (DEE betriebsbereit) Der ST signalisiert der DÜE mit "Ein"-Pegel an diesem Anschluß Betriebsbereitschaft.
RI	Pin 22	Ring indicator (Ankommender Ruf) Die DÜE meldet dem ST einen "Anruf" von der Gegenstation. Das ist die "Telefonklingel", mit der dem ST mitgeteilt wird, daß die Gegenseite einen Datenaustausch durchführen will. Das Signal wird eigentlich nur bei Modems benutzt, die in einem als Mailbox arbeitenden Computer installiert sind.

Der ST besitzt zwar unter seinen Peripheriebausteinen zwei sogenannte ACIAs (Asynchronous-Communications-Interface-Adaptor = Interface für asynchrone Datenübertragung), jedoch werden diese nicht für die Bedienung der herausgeführten seriellen Schnittstelle benutzt. Sie versorgen die MIDI-Schnittstelle und die "intelligente" Tastatur.

Arbeitsteilung am RS232-Port

Wie auch bei der parallelen Schnittstelle teilen sich bei der "Betreuung" der seriellen Schnittstelle des ST der PSG- (Programmable-Sound-Generator) und der MFP-Chip (Multi-Function-Peripheral) die Arbeit.

Die Parallel/Seriell-Wandlung der zu sendenden bzw. empfangenden Daten nimmt der MFP vor, welcher neben anderen interessanten Schnittstellen und Funktionseinheiten (Timer), auch einen USART (Universal Synchronous/ Asynchronous Receiver/ Transmitter = Universelle Synchron/Asynchron Sende-/Empfangseinheit) besitzt.

Die Datenübertragungsrate (Zahl der übertragenen Bits/Sek.) wird dabei durch die Taktfrequenz an den Anschlüssen TC (Transmitter Clock = Sendetakt, Pin 7) und RC (Receiver Clock = Empfangstakt, Pin 10) bestimmt, welche beim ST direkt miteinander verbunden sind. Geliefert wird der Takt vom Anschluß TDO (Timer-D-Output, Pin 16) des MFP. Sender und Empfänger arbeiten also immer auf der gleichen Taktfrequenz (gleiche Datenübertragungsgeschwindigkeit)!

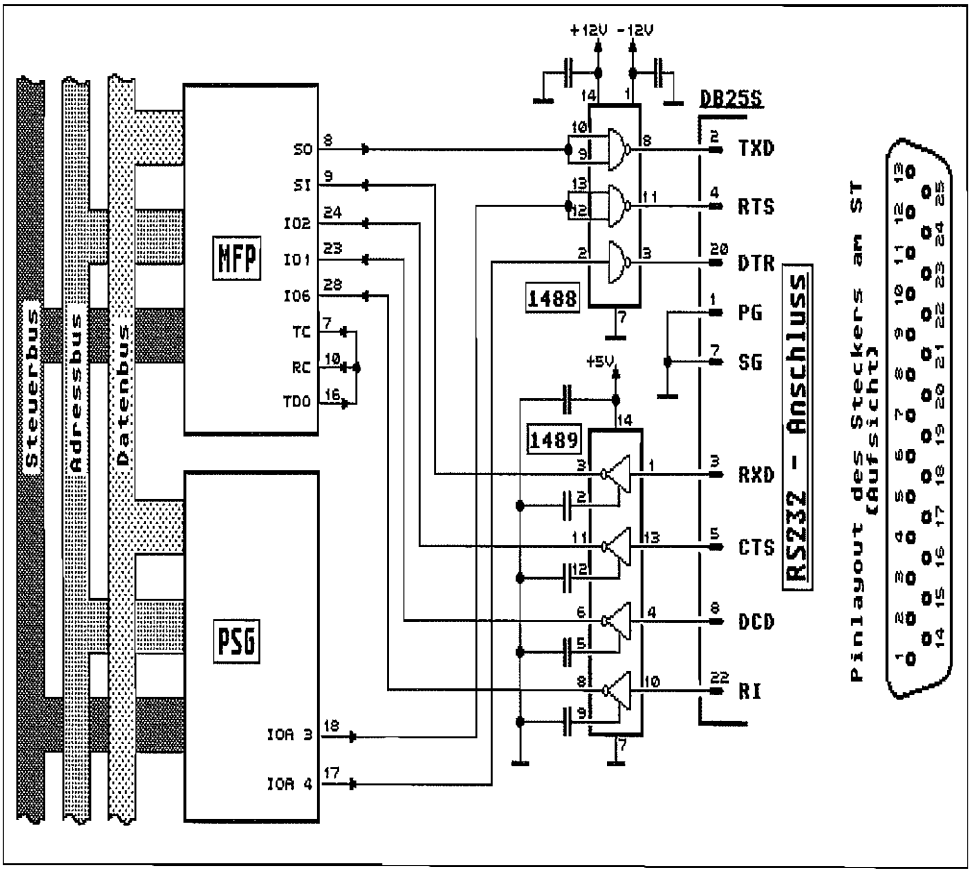


Abb. 5.1: So ist die serielle Schnittstelle des ST realisiert

Der Sender im USART des MFP besitzt ein Senderegister (Sendepuffer), in den das zu sendende Zeichen (maximal 8 Bit breit!) eingeschrieben wird. Von dort gelangt das Zeichen in ein Sendeschieberegister, das die Daten Bit für Bit im Takt der Übertragungsrate am SO-Ausgang des MFP (Pin 8) ausgibt.

Die Übergabe des Zeichens vom Sendepuffer ins Sendeschieberegister geschieht automatisch dann, wenn das vorherige Zeichen aus dem Sendeschieberegister komplett "hinausgeschoben" wurde. Der Status der Senderichtung kann aus dem Transmitter-Status-Register ausgelesen werden.

Genau umgekehrt ist der Ablauf im Empfangsteil des USART. Die seriellen Daten werden Bit für Bit über den SI-Anschluß des MFP (Pin 9) im Takt der vereinbarten Übertragungsrate in das Empfangsschieberegister "hineingeschoben". Wenn die für ein Zeichen vereinbarte Zahl von Bits empfangen ist, wird der Inhalt des Empfangsschieberegisters in das Empfangsregister (Empfangspuffer) übertragen und kann von dort ausgelesen werden. Der Status der Empfangsrichtung wird über ein Receiver-Status-Register signalisiert.

Damit die Datenübertragung interruptgesteuert durchgeführt werden kann (die CPU greift nur ein, wenn ein Zeichen komplett empfangen wurde und ausgelesen werden muß oder wenn ein neues Zeichen in den Sendepuffer gebracht werden muß oder wenn ein Fehler in der Übertragung aufgetreten ist), besitzt der USART des MFP für jede Datenrichtung zwei Interrupt-Kanäle.

Ein RCV-Buffer-full-Interrupt (Priorität 12 bei MFP-Interrupts) signalisiert die Bereitstellung eines Zeichens im Empfangspuffer. Der Empfangspuffer sollte nun ausgelesen werden. Geschieht das nicht rechtzeitig, und wurde ein weiteres Zeichen bereits empfangen (steht im Empfangsschieberegister bereit), so wird ein RCV-Error-Interrupt (Priorität 11 bei MFP-Interrupts) ausgelöst. Weitere Bedingungen für RCV-Error-Interrupts werden bei der Besprechung der USART-Register erläutert.

Ein XMIT-Buffer-empty-Interrupt (Priorität 10 bei MFP-Interrupts) wird ausgelöst, wenn das im Sendepuffer befindliche Zeichen in das Sendeschieberegister übertragen wurde und der Sendepuffer wieder gefüllt werden sollte. Wird das Senderegister nicht nachgeladen, bevor das Sendeschieberegister vollkommen geleert ist, so tritt ein XMIT-Error-Interrupt (Priorität 9 bei MFP-Interrupts) auf.

Die Register des MFP-USART

Für die Kontrolle des Sende- und Empfangsteils des MFP-USART existieren fünf Register mit jeweils 8 Bit Breite.

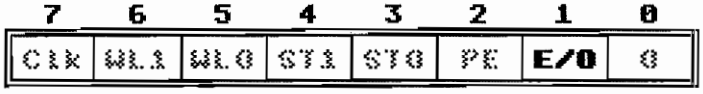
Register	Adresse	Zugriff	Bedeutung	Label
SCR	$FF FA27	R/W	Synchronous-Character-Register	"SCR"

Hier steht das Synchronzeichen, welches als Hilfszeichen für die Datenübertragung bei synchroner Betriebsart verwendet wird (zur Herstellung der Zeichensynchronisation mit der Gegenstelle).

UCR	$FF FA29	R/W	USART-Control-Register	"UCR"

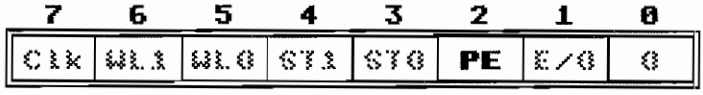
Durch Setzen bzw. Rücksetzen von einzelnen Bits werden verschiedene Betriebsmodi der seriellen Schnittstelle eingestellt.

Even-/Odd-Parity Bit



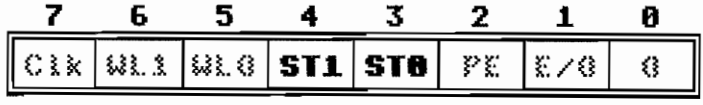
Bei gesetztem Bit arbeitet der USART mit gerader Parität (Even Parity = Das zusätzlich in einem Datenwort enthaltene Paritätsbit wird so gesetzt, daß sich eine gerade Anzahl von Eins-Bits im Datenwort ergibt), sonst mit ungerader Parität (Odd Parity = Zahl der Eins-Bits eines Datenwortes wird durch das Paritätsbit auf ungerade ergänzt).

Parity Enable Bit



Bei gesetztem Bit findet die Datenübertragung mit Paritätsbit statt. Es wird also am Ende des zu übertragenden Zeichens ein zusätzliches Bit angehängt, welches entsprechend der gewählten Art der Parität (Even/Odd) gesetzt wird. Siehe hierzu auch die Erläuterung des E/O-Bits.

Format Control



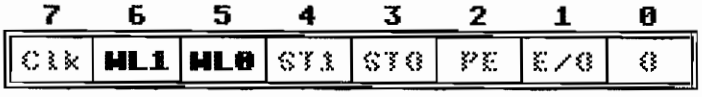
Einstellung der Betriebsart und bei asynchroner Betriebsart die Zahl der Start- /Stop-Bits.

Register	Adresse	Zugriff	Bedeutung	Label
UCR	$FF FA29	R/W	USART-Control-Register	"UCR"

ST1	ST0	Start Bits	Stop Bits	Format
0	0	–	–	Synchron
0	1	1	1	Asynchron
1	0	1	1,5	Asynchron
1	1	1	2	Asynchron

Der Betrieb mit 1,5 Stopbits ist nur möglich, wenn mit Vorteiler (Übertragungsrate ist 1/16 des Taktes am TC/RC-Anschluß) gearbeitet wird (Clk-Bit gesetzt)!

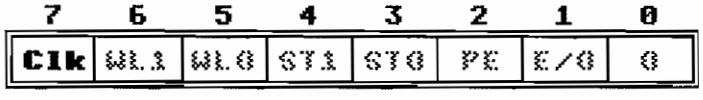
Word Length



Die Bitkombination legt fest, aus wie vielen Bits ein zu übertragendes Zeichen besteht.

WL1	WL0	Wortlänge
0	0	8 Bits
0	1	7 Bits
1	0	6 Bits
1	1	5 Bits

Clock Mode

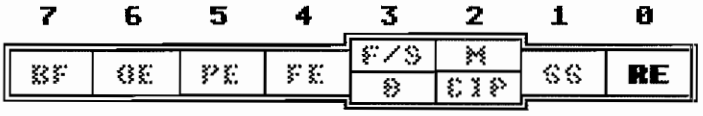


Hiermit läßt sich einstellen, in welcher Beziehung Datenübertragungsrate und USART-Takt stehen. Bei gesetztem Bit arbeitet der USART in der sogenannten "Vorteiler-Betriebsart". Die an den TC/RC-Anschlüssen liegende Taktfrequenz wird in einem Vorteiler des USART durch 16 dividiert und bildet erst dann die Taktfrequenz für die serielle Datenübertragung. Bei gelöschtem Bit stimmen Taktfrequenz und Übertragungsrate überein.

Register	Adresse	Zugriff	Bedeutung	Label
RSR	$FF FA2B	R/W	Receiver-Status-Register	"RSR"

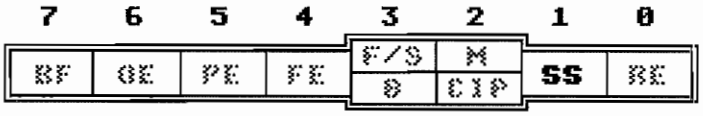
Der USART meldet durch Setzen/Rücksetzen einzelner Bits den Status eines Zeichenempfangs. Außerdem läßt sich durch Setzen/Löschen einiger Bits das Verhalten des Empfängers steuern.

Receiver Enable



Empfänger Ein/Aus. Bei rückgesetztem Bit ist der Empfänger im USART gesperrt. Außerdem werden alle Flags im RSR gelöscht.

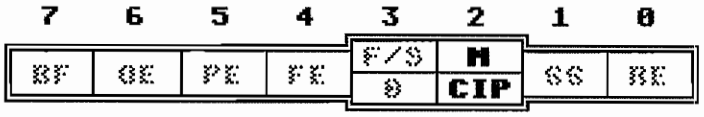
Synchronous Strip Enable



Nur bei Synchronbetrieb von Bedeutung. Bei gelöschtem Bit werden die empfangenen Synchronzeichen erst gar nicht in den Empfangspuffer übertragen.

Es gelangen somit nur alle "Nutz-Zeichen" in den Empfangspuffer!

Match/Character in Progress

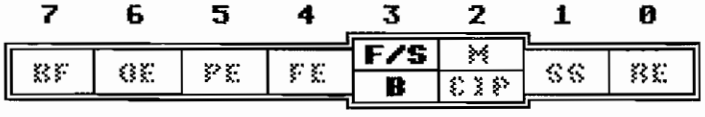


Synchronbetrieb: Match
Ein gesetztes Bit signalisiert den Empfang eines Synchron-Zeichens.

Register	Adresse	Zugriff	Bedeutung	Label
RSR	$FF FA2B	R/W	Receiver-Status-Register	"RSR"

Asynchronbetrieb: Character in Progress
Es wird aus dem gerade empfangenen Bitstrom ein Zeichen "zusammengebaut". Mit Erkennung des Startbits auf der Empfangsleitung wird das Bit gesetzt und mit Empfang des letzten Stopbits eines Zeichens wieder gelöscht. Also eine Art "BUSY"-Flag.

Found/Search bzw. Break



Synchronbetrieb: Found/Search
Durch Rücksetzen dieses Bits wird der Empfänger auf "Suchen" geschaltet. Der eintreffende Datenstrom wird auf Synchronzeichen untersucht. (Der Wortlängenzähler ist dabei ausgeschaltet, so daß die erste Bitfolge, die mit dem Zeichen im SCR (Synchronous-Character-Register) übereinstimmt, als Synchronzeichen erkannt wird). Wurde ein Synchronzeichen erkannt, so wird das Bit gesetzt, ein RCV-Error-Interrupt (siehe auch Teil II, Kapitel 4, "Interrupts durch den MFP-Baustein") ausgelöst und der Wortlängenzähler eingeschaltet.

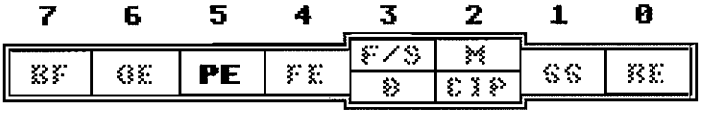
Asynchronbetrieb: Break Detect
Eine Pause in der Datenübertragung wird durch ein gesetztes Bit signalisiert. Es wird eine laufende Folge von "0-Bits" ohne Stopbit empfangen. Das Zeichen im Empfangspuffer ist ein "Break"-Zeichen (alles Nullen). Ein RCV-Error-Interrupt wird ausgelöst. Nach einem "Break" wird dieses Bit erst wieder gelöscht, wenn mindestens ein "1-Bit" empfangen und das RSR gelesen wurde (Quittierung).

Frame Error

Ein gesetztes Bit signalisiert einen "Rahmenfehler": dem im Asynchron-Betrieb empfangenen (Nicht Null!)-Zeichen folgt kein Stop-Bit. Das Bit wird gelöscht, wenn ein korrekt "umrahmtes" Zeichen empfangen wurde. (Hierfür wird kein RCV-Error-Interrupt ausgelöst!)

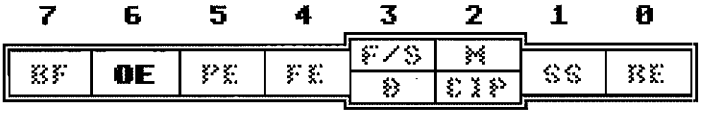
Register	Adresse	Zugriff	Bedeutung	Label
RSR	$FF FA2B	R/W	Receiver-Status-Register	"RSR"

Parity Error



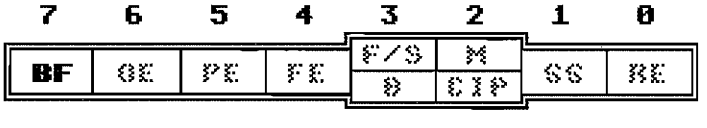
Ein gesetztes Bit zeigt an, daß ein im Empfangspuffer befindliches Zeichen einen Parity-Fehler aufweist. Dieser Fehler löst auch einen RCV-Error Interrupt aus. Das Bit wird gelöscht, sobald ein neues Zeichen ohne Parity-Fehler in den Empfangspuffer übertragen wird.

Overrun Error



Das Bit wird gesetzt, wenn ein gerade eingetroffenes Zeichen in den Empfangspuffer gebracht werden soll, dieser aber noch nicht ausgelesen wurde. Der Empfangspuffer wird jedoch *nicht* überschrieben. Weitere eintreffende Zeichen werden nicht beachtet, bis das Bit durch Lesen des RSR gelöscht wurde (Quittierung). Ein RCV-Error-Interrupt wird ausgelöst.

Buffer Full



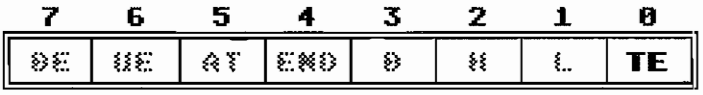
Das Bit wird gesetzt, sobald ein empfangenes Zeichen in den Empfangsbuffer übertragen wird. Gelöscht wird durch Auslesen des Empfangsbuffers (UDR = USART- Data-Register). Ein RCV-Buffer-Full-Interrupt wird ausgelöst.

TSR	$FF FA2D	R/W	Transmitter-Status-Register	"TSR"

Die Bits des TSR steuern den Sendeteil des USART und liefern Informationen über den Status des Sendeteils.

Register	Adresse	Zugriff	Bedeutung	Label
TSR	$FF FA2D	R/W	Transmitter-Status-Register	"TSR"

Transmitter **E**nable



Bei gelöschtem Bit ist der Sender außer Betrieb. Als Folge davon wäre das "END-Bit" gesetzt und das "UE-Bit" gelöscht. Ein Setzen des Bits gibt den Sender frei. Bis zur Aussendung des ersten Zeichens ist der Ausgang des Senders auf High- bzw. Low-Pegel oder Hochohmig (abhängig von der Bitkombination der "High and Low-Bits").

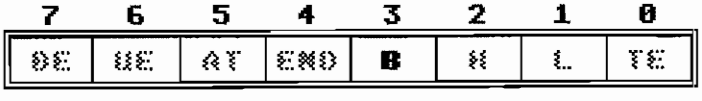
High and **L**ow

Die Bitkombination dieser beiden Bits legt den Zustand des Senderausgangs (Anschluß SO des MFP) fest, wenn der Sendeteil gesperrt ist oder gerade erst freigegeben wurde.

H	L	Senderausgang
0	0	Hochohmig
0	1	Low-Pegel
1	0	High-Pegel
1	1	Schleife

Bei "Schleife" wird der Senderausgang intern mit dem Empfängereingang verbunden. Ebenso werden Sende- und Empfangstakt miteinander gekoppelt. Die Anschlüsse RC (Receiver Clock) und SI (Serial Input) werden in diesem Fall nicht mehr benutzt.

Break

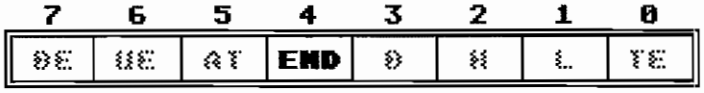


Im Asynchron-Betrieb wird bei gesetztem Bit ein "Break" gesendet, sobald das Senderegister leer ist.

Register	Adresse	Zugriff	Bedeutung	Label
TSR	$FF FA2D	R/W	Transmitter-Status-Register	"TSR"

Während "Break" gesendet wird, erfolgt an jeder Zeichengrenze ein "XMIT-Error-Interrupt" mit gesetztem "END-Bit". Sobald das "Break-Bit" gelöscht wird, erfolgt wieder normaler Sendebetrieb.

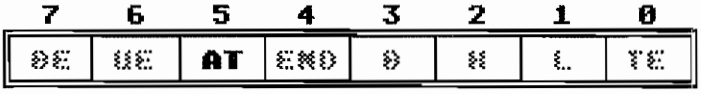
End of Transmission



Dieses Bit wird gesetzt, sobald der Sender gesperrt wird (TE-Bit). Ein XMIT-Error-Interrupt wird ausgelöst.

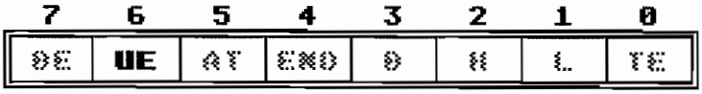
Befindet sich zum Zeitpunkt der Sperre des Sendeteils gerade noch ein Zeichen im Sendeschieberegister, so geht das "END-Bit" erst auf "1", wenn das Zeichen komplett ausgesendet wurde.

Auto Turnaround



Bei gesetztem Bit wird der Empfänger automatisch erst dann freigegeben, wenn der Sender gesperrt und das letzte zu sendende Zeichen komplett ausgesendet wurde (bei Halbduplex-Betrieb ganz sinnvoll).

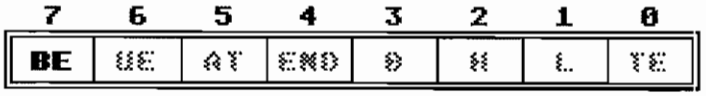
Underrun Error



Bit gesetzt: Ein Zeichen ist komplett gesendet worden, bevor ein neues Zeichen in den Sendepuffer eingeschrieben wurde. Ein XMIT-Error-Interrupt wird ausgelöst. Dieses Bit wird mit dem Lesen des TSR oder durch Sperren des USART-Sendeteils (Löschen des TE-Bits) wieder zurückgesetzt.

Register	Adresse	Zugriff	Bedeutung	Label
TSR	$FF FA2D	R/W	Transmitter-Status-Register	"TSR"

Buffer Empty



Sobald das Zeichen aus dem Sendepuffer ins Senderegister gelangt, wird das Bit gesetzt. Ein XMIT-Buffer-empty-Interrupt wird ausgelöst. Mit Einschreiben eines neuen Zeichens in den Sendepuffer wird das Bit zurückgesetzt.

UDR	$FF FA2F	R/W	USART-Data-Register	"UDR"

Bei einem *Schreibzugriff* gelangt das eingeschriebene Byte in den Sendepuffer. Ein *Lesezugriff* liefert das Byte aus dem Empfangspuffer. Aus der Vielzahl der Steuermöglichkeiten und Rückmeldungen des USART läßt sich ersehen, wie flexibel die serielle Schnittstelle des ST bedient werden kann.

Standardeinstellungen sind schon vorgesehen

Die Übertragungsrate der seriellen Schnittstelle des MFP wird zwar durch Timer D bestimmt, jedoch sind im Betriebssystem schon Voreinstellungen für die gebräuchlichsten Datenübertragungsraten vorgesehen. Diese lassen sich, zusammen mit anderen Einstellungen, durch die XBIOS-Funktion #15 ("Rsconf") abrufen.

Leider hat die XBIOS-Funktion "Rsconf" zwei Bugs (die von ATARI aus "Kompatibilitätsgründen" auch in neueren TOS-Versionen *nicht* behoben wurden):

- Die Übertragungsgeschwindigkeiten 50 und 75 Bit/s werden vom TOS falsch eingestellt (falsche Werte für das Timer-D-Data-Register)! Will man mit "Rsconf" auf 50 Bit/s einstellen, läuft die serielle Schnittstelle in Wirklichkeit mit 80 Bit/s. Bei einer Einstellung auf 75 Bit/s ergibt sich in Wirklichkeit eine Geschwindigkeit von 120 Bit/s.

- Durch einen falschen Adreßdistanzwert in der "Rsconf"-Routine werden nicht – wie gefordert – die beim Aufruf gültigen Einstellungen der USART-Register SCR, UCR, RSR, und TSR in CPU-Register D0 zurückgeliefert (SCR-Wert im höchstwertigen Byte, TSR-Wert im niedrigstwertigen Byte des Langworts), sondern die Werte für UCR, RSR, TSR und UDR!

Mit "Rsconf" lassen sich außerdem zwei verschiedene Handshaking-Methoden anwählen. Es ist sowohl Hardware-Handshake mittels RTS/CTS-Signalen möglich als auch Software-Handshake mit XON/XOFF-Betrieb.

Im Betriebssystem des ST ist unter anderem auch für den Datenaustausch über die serielle Schnittstelle ein Pufferspeicher für die zu übertragenden Daten vorgesehen.

Zu sendende Daten gelangen zunächst in einen Puffer und werden dann (interruptgesteuert) zeichenweise an die serielle Schnittstelle "weitergereicht". Ähnlich ergeht es empfangenen Zeichen, welche per Interruptroutine aus dem USART-Data-Register (UDR) in einen Empfangspuffer transferiert werden.

Protokolle sind manchmal wichtig

Wird mit Hard- oder Software-Handshake gearbeitet, so werden die "Füllstände" der Pufferspeicher mit einer sogenannten "High- bzw. Low-water mark" verglichen.

Werden die empfangenen Zeichen schneller in den Empfangspufferspeicher eingeschrieben, als sie durch z. B. ein Programm von dort ausgelesen werden können, wird der "Füllstand" irgendwann die "High-water mark" erreichen.

Tritt dieser Fall ein, wird dem Sender der Gegenseite über EIN-Signal (+12V) am RTS-Anschluß des ST (bei Hardware-Handshake) bzw. durch Senden eines "Ctrl-S"-Zeichens = XOFF-Zeichen (bei Software-Handshake) signalisiert, die weitere Aussendung von Zeichen zu unterbrechen.

Sinkt der "Füllstand" des Empfangspufferspeichers durch Auslesen von Zeichen dann wieder unter die "Low-water mark", so wird die Gegenseite per AUS-Signal (–12V) an RTS (bei Hardware-Handshake) bzw. Senden eines XON-Zeichens = "Ctrl-Q" (bei Software-Handshake) darüber informiert, daß der ST wieder bereit ist, Daten aufzunehmen.

Frühere TOS-Versionen (bis zum Blitter-TOS 1.02) erwarteten bei RTS/CTS-Handshake nach der Ausgabe *jedes* Zeichens ein Quittungssignal (kurzes "Ein") über die CTS-Leitung. Außerdem wurde vom ST die RTS-Leitung nach dem Empfang eines *jeden* Zeichens für die Dauer der Verarbeitung auf "Aus" und erst anschließend wieder auf "Ein" gesetzt!

In der TOS-Version 1.04 funktioniert der RTS/CTS-Handshake dann endlich wie gedacht, wenn er sich nur einschalten ließe! Die dazu nötige XBIOS-Routine #15 (Rsconf) hat einen Bug, der das Einschalten des RTS/CTS-Handshakes nicht erlaubt. Der Fehler läßt sich aber mit einem von ATARI gelieferten Autoordner-Programm (TOS14FIX.PRG) beheben.

Bei neueren TOS-Versionen (im MEGA STE und im TT!) hat ATARI *diesen* Bug nicht mit übernommen. Der RTS/CTS-Handshake läßt sich jetzt zwar mittels Rsconf-Aufruf setzen. Leider plaziert die Rsconf-Routine des TOS die Handshake-Parameter an eine Stelle in der IOREC-Struktur, die überhaupt nicht von den entsprechenden Interruptroutinen bei der Feststellung der gewünschten Handshake-Betriebsart ausgewertet wird (Rsconf trägt die Handshake-Art mittels *Word*zugriff in die IOREC-Struktur ab Byte 32 ein, getestet wird die Handshake-Art aber in den Interrupt-Routinen mit *Byte*zugriffen auf Byte 33)! Auch hier kann man sich dann nur wieder mit entsprechenden Patch-Programmen helfen.

Kapitel 6: Die parallele Druckerschnittstelle

Die ST-Computer besitzen eine Schnittstelle für den Anschluß eines Druckers mit Centronics-Interface.

Die Centronics-Schnittstelle erlaubt gegenüber einer seriellen Schnittstelle relativ hohe Datenübertragungsraten, weil jeweils ein komplettes Byte über acht parallele Datenleitungen an den Drucker ausgegeben wird. Für die Darstellung der Informationszustände (Bit-High oder -Low) auf den Leitungen werden TTL-Pegel verwendet.

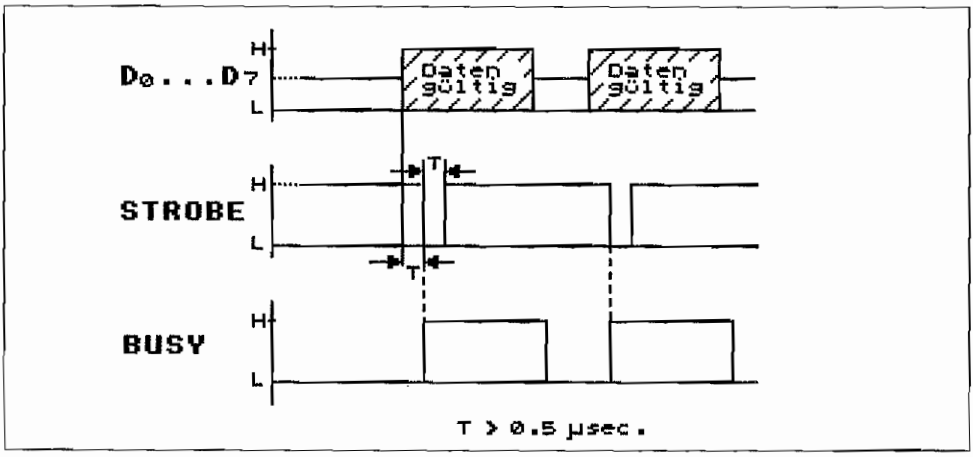


Abb. 6.1: Timing-Diagramm der Centronics-Schnittstelle

Eine STROBE-Leitung signalisiert dabei dem Drucker durch einen kurzen Low-Impuls, wann die auf dem 8-Bit-Bus liegenden Daten gültig sind. Die Daten werden bei der steigenden Flanke des STROBE-Impulses vom Drucker übernommen.

Da der Computer in der Regel mit der Datenausgabe schneller ist, als der Drucker die Zeichen aufs Papier bringen kann, ist ein weiteres Handshaking-Signal zur störungsfreien Datenübertragung erforderlich. Über die BUSY-Leitung zeigt der Drucker dem Computer mit einem High-Pegel an, wenn er noch beschäftigt (busy) ist und noch keine neuen Daten aufnehmen kann. Üblicherweise stellen Drucker mit Centronics-Schnittstelle noch mindestens ein sogenanntes ACKNLG-Signal (Acknowledge = Empfangsbestätigung) zur Verfügung, welches

durch Low-Impuls die korrekte Datenübernahme bestätigt. Dieses Signal wird jedoch von den ST-Computern nicht ausgewertet.

Drucken mit Musik?

Auch wenn für die parallele Schnittstelle der Soundchip benutzt wird, der ST macht während des Druckvorgangs keine Musik, um eventuell das nervtötende Geräusch eines sägenden Matrixdruckers zu übertönen. Vielmehr wird der PSG-Chip (Programmable-Sound-Generator) des ST für die Datenausgabe auf den Drucker mitbenutzt.

Wie der kurze Ausschnitt aus der ST-Schaltung zeigt, kommen die Interfaceleitungen für die Parallel-Schnittstelle von zwei verschiedenen Chips des ST-Computersystems.

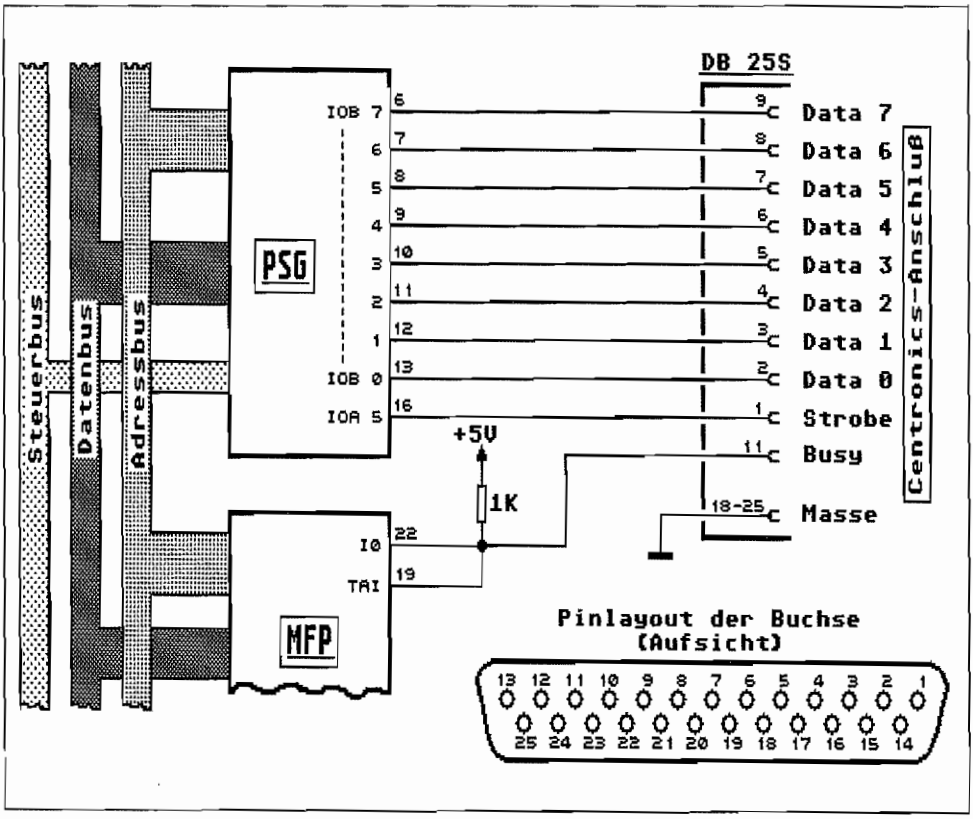


Abb. 6.2: So ist die Centronics-Schnittstelle des ST aufgebaut

Der Port B des PSG-Chips liefert die Signale für die acht Datenleitungen. An Port A, Anschluß IOA5, wird das STROBE-Signal generiert. Die Statusabfrage der BUSY-Leitung erfolgt über den MFP-Baustein (Multi-Function Peripheral) an dessen Eingang I0.

Für den Anschluß eines Druckers kann ein handelsübliches Druckerkabel mit DB 25S-Stecker auf der Seite zum Computer und 36pol. AMP-Stecker zum Drucker verwendet werden (wird auch für Anschluß von Druckern mit Centronics-Anschluß an PC-kompatible Computer benutzt).

Die Zeichenausgabe an die Parallelschnittstelle wird natürlich voll vom Betriebssystem unterstützt (siehe auch BIOS-Funktionen #1, #2 und #3 ("Bconstat", "Bconin", "Bconout")). Die Datentransferrate liegt bei ca. 4000 Bytes/Sek. (Wär das toll, wenn der Drucker da mitkäme), wobei die Ausgänge jeweils eine Standard-TTL-Last treiben können.

Beim Anschluß von älteren Druckern mit Centronics-Interface sollte man darauf achten, daß die Druckereingänge eine nicht zu hohe Belastung für die PSG-Ausgänge darstellen. Ursache sind in der Regel zu niederohmige Pull-up-Widerstände im Drucker, die offene Eingänge auf High-Pegel ziehen sollen. Nachmessen kann man die Stromaufnahme der Eingänge, indem man bei eingeschaltetem Drucker einen Dateneingang des Centronics-Anschlusses über ein Milliampere-Meter oder Vielfachmeßgerät im mA-Bereich mit Masse verbindet. Der sich dann einstellende Strom ist auch jener Strom, den der PSG an seinen Ausgängen liefern können muß. Mit mehr als 3 mA sollte man den PSG nicht belasten.

Keine Einbahnstraße!

Nun sind aber die Ports des PSG sowohl als Aus- wie auch als Eingänge verwendbar. Mit der BIOS-Funktion #2 ("Bconin") mit dem Parallelport als Eingabeeinheit (Char. Device #0) können über die parallele Schnittstelle Daten eingelesen werden!

Man kann dabei den ST als "Drucker" betrachten, der mit Daten von z. B. einem anderen Computer beschickt wird. Über die STROBE-Leitung des Parallelports meldet der empfangende ST, ob er empfangsbereit (High-Pegel) oder beschäftigt (Low-Pegel) ist. Über den BUSY-Anschluß erhält der empfangende ST die Signalisierung, daß die anliegenden Daten gültig sind (bei Low-Pegel).

Vorsicht ist geboten!

Man sollte bei der Benutzung des Parallelports als Eingang beachten, daß beim "Power-Up" des ST der Port B zunächst als Ausgang konfiguriert wird! Die Datenleitungen liefern so im

Ruhezustand ein Low. Eine an den ST sendende Einheit sollte also erst Daten (sprich Logikpegel) auf die Datenleitungen des Parallelports legen, wenn der empfangende ST den Port B auch auf "Eingang" geschaltet hat. Das ist am besten daran erkennbar, daß der STROBE-Anschluß des empfangenden ST auf High geht.

Sonst könnte es geschehen, daß zwei verschiedene Pegel aufeinandertreffen (das Low vom empfangenden ST und ein eventuelles High vom sendenden Computer!). Diese Tatsache könnte der Soundchip übelnehmen und sich verabschieden!

Die bidirektionale parallele Schnittstelle des ST wird von einigen Zubehör-Anbietern ausgenutzt. So sind z. B. EPROM-Programmiereinrichtungen und Sound-Sampler erhältlich, die über die parallele Schnittstelle arbeiten.

Ereigniszählung über den Druckerport

Ein weiteres Detail, welches bei der Betrachtung des Schaltungsauszugs auffällt, ist die gleichzeitige Verbindung der BUSY-Leitung des Parallelports mit dem Eingang I0 des MFP und mit dem Anschluß TAI (Timer-A-Input)! Somit kann Timer A des MFP, welcher vom TOS nicht gebraucht wird, z. B. als Ereigniszähler oder zur Pulsweitenmessung benutzt werden.

Außerdem ist der Eingang I0 des MFP, an welchem die BUSY-Leitung der parallelen Schnittstelle aufläuft, ja ebenfalls als Interrupt-Eingang zu betreiben. Damit kann man einen Interrupt auslösen lassen, wenn BUSY vom Drucker wieder auf Low gesetzt wird. Damit wird ja signalisiert, daß ein weiteres Zeichen an den Drucker ausgegeben werden kann. Die zugehörige Interruptroutine kann nun ein weiteres Zeichen aus einem Pufferspeicher holen und ausgeben. In diesem Pufferspeicher werden sinnvollerweise alle Zeichen gesammelt, die von den Druckausgaberoutinen des Betriebssystems sonst direkt an den Drucker gesendet werden.

Druckausgaben von Programmen füllen also schnell diesen Pufferspeicher und blockieren nicht lange die Arbeit mit dem ST, weil dieser warten muß, bis der Drucker alle Zeichen aufs Papier genadelt hat. Durch Ausnutzen des BUSY-Interrupts läßt sich also ein effektiver Druckerspooler programmieren (so z. B. im STMagazin 4/88, "Drucken ohne Zeitverlust", beschrieben), weil wirklich die CPU nur dann mit Zeichenausgaben an den Drucker "belästigt" wird, wenn dieser bereit ist, ein neues Zeichen zu verarbeiten!

Man sieht also, daß mit der parallelen Schnittstelle des ST noch einiges mehr anzufangen ist, als sie als simplen Druckeranschluß zu gebrauchen. Besonders dem Hardware-Bastler bieten sich hier noch einige interessante Möglichkeiten.

Kapitel 7: Die ACIAs im ST

Im ST sind zwei Asynchrone-Communications-Interface-Adapter (ACIA) des Typs 6850 eingesetzt. Sie helfen der CPU beim Datenaustausch mit der MIDI-Schnittstelle und der "intelligenten Tastatur". Das "Innenleben" eines ACIAs soll die Abbildung 7.1 veranschaulichen.

Jeder ACIA stellt einen kompletten Kanal (Sende- und Empfangsteil) für die serielle Datenübertragung nach der RS-232-Spezifikation zur Verfügung.

Sender und Empfänger können dabei mit unterschiedlichen Taktfrequenzen gespeist werden (über die Anschlüsse TxCLK und RxCLK) und daher mit unterschiedlichen Datenübertragungsgeschwindigkeiten (in diesem Fall gleich der Baudrate, da Verwendung binärer Signale) arbeiten.

Beim ST werden die beiden ACIAs jedoch, sowohl für die Sender- als auch die Empfängerseite, mit der gleichen Taktfrequenz (500 kHz) gespeist.

Der ACIA für die MIDI-Schnittstelle arbeitet (lt. MIDI-Spezifikation) mit einer Datenübertragungsrate von 31250 Bit/s, der ACIA für die "intelligente Tastatur" mit 7812,5 Bit/s. Wie aus dem Blockschaltbild hervorgeht, verfügt jeder ACIA über vier Register mit einer Breite von 8 Bit.

Die empfangenen seriellen Daten (über Anschluß RxDATA) werden zunächst in einem *Empfangs-Schieberegister* gesammelt. Ist ein Zeichen aus dem empfangenen seriellen Bitstrom "zusammengebaut", wird es ins *Empfangsregister* übertragen und zur Signalisierung im *Statusregister* das Bit 0 (RDRF-Bit = "Receive data register full"-Bit) gesetzt. Wenn der Empfänger-Interrupt zugelassen ist, wird auch gleichzeitig ein Interrupt ausgelöst.

Die zu sendenden Informationen müssen dem ACIA über das *Senderegister* eingegeben werden. Es kann nur beschrieben werden. Ein zu sendendes Zeichen wird, nachdem es ins Senderegister eingeschrieben wurde, von dort ins *Sende-Schieberegister* übertragen und im Takt der Übertragungsgeschwindigkeit Bit für Bit "ins Freie" geschoben (über den Anschluß TxDATA).

Zur Signalisierung, daß das Senderegister geleert wurde, also ein neues Zeichen eingeschrieben werden kann, wird im Statusregister das Bit 1 (TDRE-Bit = "Transmitter data register empty"-Bit) gesetzt. Ist der "Sender-Interrupt" freigegeben, wird ein Interrupt ausgelöst, wenn das Senderegister wieder "nachgeladen" werden muß.

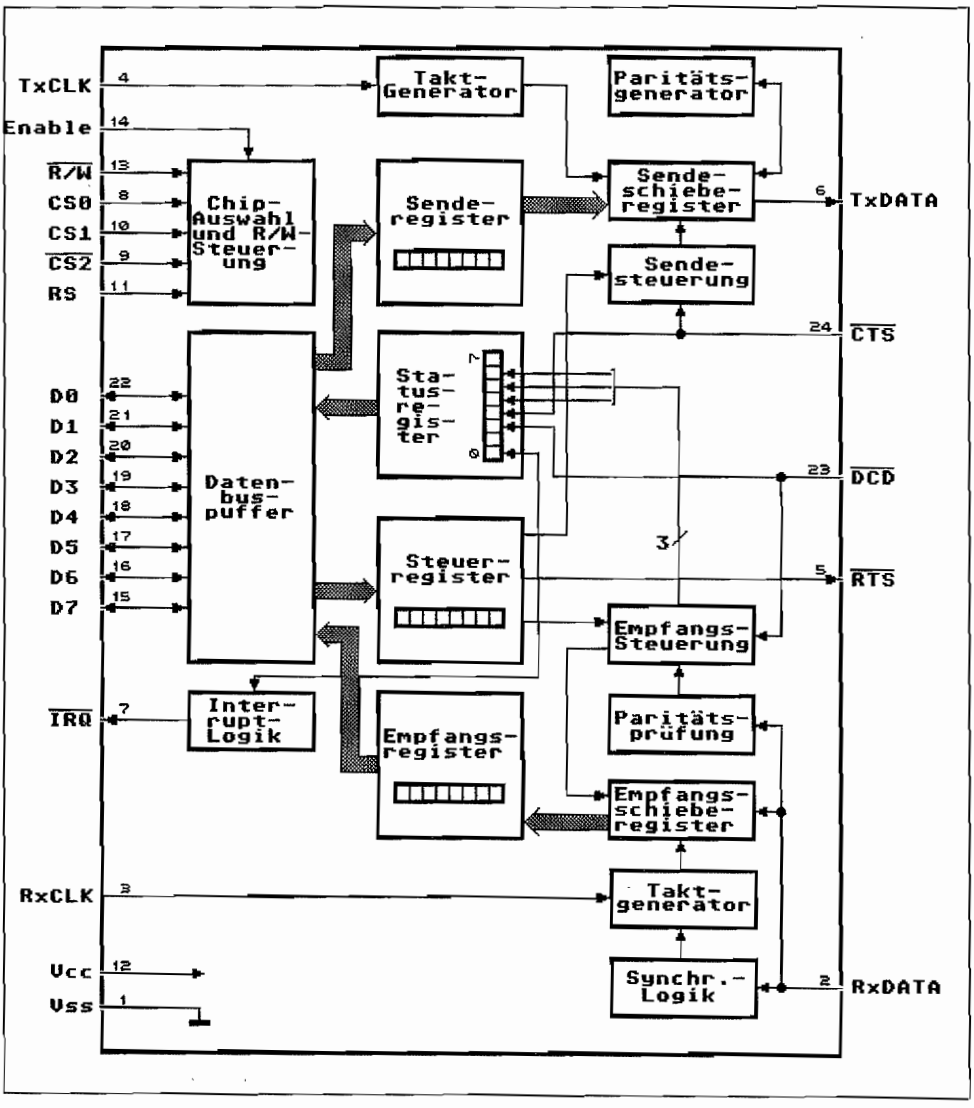


Abb. 7.1: Das Blockschaltbild des ACIA 6850

ACIA-Steuerregister

Nur Schreiben erlaubt! Bit für Bit lassen sich bestimmte Betriebsweisen des ACIA einstellen.

Bit 1	Bit 0	Funktion
0	0	Baudrate ist gleich Taktfrequenz
0	1	Baudrate ist 1/16 der Taktfrequenz (MIDI-Port)
1	0	Baudrate ist 1/64 der Taktfrequenz (Tastatur)
1	1	Rücksetzung des ACIA (Master-Reset)

Die Neuinitialisierung eines ACIAs muß softwaremäßig erfolgen. Das geschieht durch Setzen der beiden niederwertigen Bits im Steuerregister.

Die *Bits 2..4* des Steuerregisters bestimmen das Zeichenformat für die Datenübertragung.

Bit-Nr.:	4	3	2	Funktion
	0	0	0	7 Datenbits, gerade Parität, 2 Stop-Bits
	0	0	1	7 Datenbits, ungerade Parität, 2 Stop-Bits
	0	1	0	7 Datenbits, gerade Parität, 1 Stop-Bit
	0	1	1	7 Datenbits, ungerade Parität, 1 Stop-Bit
	1	0	0	8 Datenbits, keine Parität, 2 Stop-Bits
	1	0	1	8 Datenbits, keine Parität, 1 Stop-Bit
	1	1	0	8 Datenbits, gerade Parität, 1 Stop-Bit
	1	1	1	8 Datenbits, ungerade Parität, 1 Stop-Bit

Die *Bits 5 und 6* im Steuerregister dienen zur Steuerung des Sendeteils.

Bit 6	Bit 5	Funktion
0	0	RTS auf Low, TxINT gesperrt
0	1	RTS auf Low, TxINT freigegeben
1	0	RTS auf High, TxINT gesperrt
1	1	RTS auf Low, TxINT gesperrt, Break senden

"Break senden" bedeutet nichts anderes als die Sendeleitung bis auf weiteres auf Low-Pegel zu halten, also wird eine laufende Folge von 0-Bits gesendet.

Durch Setzen des *Bit 7 des Steuerregisters* wird der "Empfänger-Interrupt" (RxINT) eingeschaltet. So wird immer dann ein Interrupt ausgelöst, wenn das Empfangsregister voll ist und ein Zeichen ausgelesen werden soll.

Weiterhin führt ein Low/High-Wechsel am DCD-Anschluß (Data Carrier Detect) oder ein Empfangsfehler (Receiver Overrun Error) zum Interrupt.

ACIA-Statusregister

Dieses Register liefert Informationen über den Zustand einer Datenübertragung. Alle acht Bits signalisieren wieder verschiedene Betriebszustände oder Fehler.

Bit 0 = "Receive Data Register Full" (RDRF-Bit)

Wird gesetzt, sobald das Empfangsregister voll ist, also ein Zeichen ausgelesen werden muß. Dieses Bit wird gelöscht, wenn das empfangene Zeichen aus dem Empfangsregister ausgelesen oder ein Master-Reset durchgeführt wird.

Bit 1 = "Transmit Data Register Empty" (TDRE-Bit)

Geht auf Log. 1, wenn der Sender wieder mit dem nächsten zu sendenden Zeichen "nachgeladen" werden muß.

Bit 2 = "Data Carrier Detect" (DCD-Bit)

Wird gesetzt, wenn der DCD-Anschluß auf High geht (Modem signalisiert: "Es wird kein Daten-Trägersignal empfangen"). Ist der RxINT freigegeben, wird auch ein Interrupt ausgelöst! Das Bit wird erst gelöscht, wenn das Status- und dann das Empfangsregister "bedient", d. h. ausgelesen wurden. Bleibt danach der DCD-Anschluß weiterhin High, so wird zwar ein evtl. Interrupt gelöscht, das DCD-Bit bleibt aber weiter gesetzt, bis der DCD-Anschluß wieder auf Low geht. Im ST wird dieses Bit nicht verwendet, weil der DCD-Anschluß fest auf Low (Masse) liegt!

Bit 3 = "Clear To Send" (CTS-Bit)

Dieses Bit zeigt den Zustand des CTS-Anschlusses an. Ist der CTS-Anschluß High (Sender nicht bereit), bleibt auch das TDRE-Bit (Bit 1) unwirksam, da ja keine Daten gesendet werden können. Ein Master-Reset beeinflußt dieses Bit nicht! Im ST wird dieses Bit nicht verwendet, weil der CTS-Anschluß fest auf Low (Masse) liegt!

Bit 4 = "Framing Error" (FE-Bit)

Ein gesetztes Bit signalisiert einen Empfangsfehler. Es wurde ein "Nicht-Null"-Zeichen empfangen, dem aber "das Ende", d. h. ein Stopbit fehlt.

Bit 5 = "Receiver Overrun" (OVRN-Bit)

Wird gesetzt, wenn ein empfangenes Zeichen im Empfangsregister "vergessen" worden ist. Das nächste bereits empfangene Zeichen hat das noch im Empfangsregister vorhandene Zei-

chen "überrannt", d. h. überschrieben, bevor es ausgelesen wurde. Ist der RxINT zugelassen, erfolgt auch eine Interrupt-Signalisierung. Ein gesetztes OVRN-Bit wird durch Lesen des Empfangsregisters oder einen Master-Reset gelöscht.

Bit 6 = "Parity Error" (PE-Bit)

Wenn Datenübertragung mit Paritätsprüfung durchgeführt wird, signalisiert ein gesetztes PE-Bit, daß die Zahl der 1-Bits des empfangenen Zeichens nicht mit der gewählten Parität (gerade oder ungerade Anzahl von 1-Bits) übereinstimmt.

Bit 7 = "Interrupt Request" (IRQ-Bit)

Geht bei einem Interrupt auf log. 1 (der IRQ-*Anschluß* wird bei einem Interrupt *Low*!).

Einbindung in die ST-Hardware

Aufgrund der besonderen Registeranordnung (Zwei "Nur Lesen"- und zwei "Nur Schreiben"-Register) benötigt jeder ACIA nur zwei Adressen im I/O-Bereich eines Computersystems (so auch im ST).

Im ST sind das die folgenden Adressen:

Adresse	Device	Register	Zugriff	Label
$FF FC00	Tastatur	Steuerregister	Write	"keyctl"
		Statusregister	Read	
$FF FC02	Tastatur	Senderegister	Write	"keybd"
		Empfangsregister	Read	
$FF FC04	MIDI-Port	Steuerregister	Write	"midictl"
		Statusregister	Read	
$FF FC06	MIDI-Port	Senderegister	Write	"midi"
		Empfangsregister	Read	

Die Register sind nur 8 Bit breit (also keine Wortzugriffe!). Auf welches Register*paar* zugegriffen wird, hängt vom Zustand des RS-Anschlusses (Register Select) des ACIA-Chips ab. Über die R/W-Leitung wird dann das entsprechende Register angewählt.

RS	R/W	ausgew. Register	Zugriffsart
0	0	Steuerregister	Schreiben
0	1	Statusregister	Lesen

RS	R/W	ausgew. Register	Zugriffsart
1	0	Senderegister	Schreiben
1	1	Empfangsregister	Lesen

Soviel zum Innenleben des ACIA. Jetzt zum Einsatz der ACIA-Chips im ST.

Die MIDI-Schnittstelle

MIDI ist die Abkürzung für Musical-Instruments-Digital-Interface, was auf deutsch wohl mit "Digitale Schnittstelle für Musikinstrumente" übersetzt werden kann.

Musikinstrumente und Effektgeräte, die ihre Klänge auf digitalem Wege erzeugen oder die Klangerzeugung digital steuern können (z. B. Synthesizer, elektronische Orgeln, Rhythmusgeräte, Mischpulte), besitzen meist eine MIDI-Schnittstelle. Über diese Schnittstelle lassen sich diese Geräte nun "fernbedienen".

Um also mehrere Synthesizer und Effektgeräte simultan spielen zu können, braucht man nur ein Mastergerät, welches über nur einen Ausgang (MIDI-out) bis zu 16 Einheiten (Slaves) bedienen kann.

Ist ein Computer mit seiner großen Speicherkapazität und Verarbeitungsgeschwindigkeit als Master angeschlossen, so kann dieser als Sequenzer eingesetzt werden, d. h., er versorgt programmgesteuert die angeschlossenen Geräte mit Einstellungen oder kompletten Melodien. Die Instrumente werden sozusagen vom Computer gespielt.

Weiterhin kann der Computer als digitale Bandmaschine arbeiten und z. B. über ein Keyboard gespielte Melodien aufzeichnen. Es wäre aber nun wenig sinnvoll, wenn per Tastendruck am Mastergerät ständig alle angeschlossenen Slaves (Synthesizer, Rhythmusmaschinen) reagieren würden.

Man kann deshalb auch mit nur einem Ausgang am Master, über eine durchgeschleifte Verbindung (bei den Slaves in MIDI-in rein und bei MIDI-Thru wieder raus zum nächsten Slave) an mehrere angeschlossene Einheiten verschiedene Informationen senden. Diese Informationen müssen nur entsprechend "adressiert" werden. Folgende Betriebsarten sind möglich:

Der *OMNI-Modus* ist die einfachste Betriebsart. Nach einem RESET befinden sich alle Geräte in diesem Modus. Alle Instrumente, die an einem Bus hängen, spielen parallel und mehrstimmig (polyphon). Es existiert also eigentlich nur ein Kanal, und das gezielte Ansprechen eines einzelnen Geräts ist nicht möglich.

Per *POLY-Modus* lassen sich verschiedene Geräte separat ansteuern. Jeder Slave bekommt eine Kanalnummer zugeordnet, die für ihn gültig ist. Das Gerät hat nur auf Kommandos zu reagieren, die auf "seinem" Kanal empfangen werden, also mit seiner Kanalnummer adressiert sind. So können dann mit beliebig vielen Instrumenten maximal 16 verschiedene Melodien (auf jedem Kanal eine andere) gespielt werden.

Der *MONO-Modus* ist die komplexeste Betriebsart. Man kann damit einzelnen Stimmen eines Synthesizers getrennte Kanäle zuordnen und so unabhängig voneinander beeinflussen. Jede Stimme kann nun wie ein einzelner einstimmiger (monophoner) Synthesizer angesprochen werden. Verfügt ein 16stimmiger Synthesizer über die Mono-Betriebsart, kann also auf diesem mit nur einem Sequenzer ein Orchester mit bis zu 16 monophonen Instrumenten gespielt werden!

Man unterscheidet bei der MIDI-Datenübertragung die beiden großen Gruppen Statusbytes und Datenbytes. Die Datenbytes enthalten die Daten zu einem Kommando und folgen einem Statusbyte. Das Statusbyte "sagt", was gemacht werden soll, während die Datenbytes die nötigen Informationen beinhalten, mit denen die angesprochene Einheit eingestellt werden soll (z. B. Schalter- bzw. Reglerstellung). Einem Statusbyte folgen normalerweise zwei Datenbytes.

- MIDI-Statusbytes unterscheiden sich von den Datenbytes durch ein gesetztes achtes Datenbit. Sie lassen sich unterteilen in:

  - *Kanalkommandos*

    Hierzu gehören die sogenannten Voice-Commands, welche zur Steuerung und Programmierung von einzelnen Instrumenten-Stimmen dienen. Mit ihnen werden Töne ein- oder ausgeschaltet, Tonhöhen eingestellt oder Informationen über Tasten-Anschlagsdynamik ("attack velocity" bzw. Tasten-Loslaßdynamik ("release velocity") und über den Tastendruck ("after touch") gegeben. Außerdem können bis zu 32 Regler und 32 Schalter zur Klangeinstellung betätigt werden. Die Schalter-/ Reglernummer und die Schalter-/ Reglerstellung sind in den Datenbytes enthalten.

    Die zweite Gruppe der Kanalkommandos sind die Modus-Kommandos. Mit ihnen wird einem Instrument mitgeteilt, in welcher Betriebsart gearbeitet werden soll (Omni-, Poly- oder Mono-Mode). Weiterhin läßt sich damit z. B. die Tastatur eines Synthesizers abschalten, so daß er nur über MIDI gespielt werden kann.

  - *Systemkommandos*

    Diese gelten nicht für einen bestimmten Kanal. Man unterscheidet bei den Systemkommandos drei Gruppen:

Status-Byte	Daten-Byte 1	Daten-Byte 2	Bedeutung
80..8F	00..7F	00..7F	NOTE OFF (mit Kanalnr.) gefolgt von Notennr. und VELOCITY (00=Note Off, 01=pianissimo, 7F=fortissimo)
90..9F	00..7F	00..7F	NOTE ON (mit Kanalnr.) gefolgt von Notennr. und VELOCITY
A0..AF	00..7F	00..7F	POLYPRESSURE (mit Kanalnr.) gefolgt von Notennr. und PRESSURE VALUE
B0..BF	00..1F	00..7F	CONTROL CHANGE (mit Kanalnr.) gefolgt von CONTROLLER-Nr. (1..32, CTL.-Nr.1=00) und CONTROLLER-VALUE-MSB (1..128)
" "	20..3F	00..7F	CONTROL CHANGE (mit Kanalnr.) gefolgt von CONTROLLER-Nr. (1..32, CTL.-Nr.1=20) und CONTROLLER-VALUE-LSB (1..128)
" "	40..5F	00..7F	SWITCH CHANGE (mit Kanalnr.) gefolgt von SWITCH-Nr. (1..32, SW.-Nr.1=00) und SWITCH-POSITION (00=Off, 7F=On)
C0..CF	00..7F	--	PROGRAM CHANGE (mit Kanalnr.) gefolgt von PROGRAM-Nr.
D0..DF	00..7F	--	CHANNEL PRESSURE (mit Kanalnr.) gefolgt von PRESSURE-VALUE
E0..EF	00..7F	00..7F	PITCH WHEEL CHANGE (mit Kanalnr.) gefolgt von VALUE-LSB und VALUE-MSB

Abb. 7.2: Voice Commands

Common Commands haben Einfluß auf alle angeschlossenen Geräte und werden benutzt, um einen bestimmten Song oder eine Sequenz oder eine bestimmte Stimmung auszuwählen oder um ein System-Reset auszuführen.

Real Time Commands dienen zur Synchronisation der Instrumente. Mit ihnen werden Rhythmusmaschinen gestartet, gestoppt oder getaktet. Ihnen folgen keine Datenbytes. Sie können auch zwischen Datenbytes anderer Kommandos gesendet werden.

Exclusive Commands sind herstellerspezifische Informationen für Sonderfunktionen, die nicht von jedem Gerät beherrscht werden. Sie können beliebig viele Datenbytes hinter sich herziehen.

Status-Byte	Daten-Byte 1	Daten-Byte 2	Bedeutung
B0..BF	7A	00 / 7F	LOCAL KEYBOARD CONTROL (mit Kanalnr.) gefolgt von '7A' und SWITCH-VALUE (00 = LOC. KBD. off, 7F = LOC. KBD. on)
" "	7B	00	ALL NOTES OFF (mit Kanalnr.)
" "	7C	--	OMNI MODE OFF (mit Kanalnr.)
" "	7D	00	OMNI MODE ON (mit Kanalnr.)
" "	7E	00..0F	MONO MODE ON / POLY MODE OFF (mit Kanalnr.) gefolgt von '7E' und Anzahl der monophon gespielten VOICES (00 = alle VOICES des Instruments))
" "	7F	--	POLY MODE ON / MONO MODE OFF (mit Kanalnummer)

Abb. 7.3: Mode Commands

Die technische Realisierung der MIDI-Schnittstelle

Die MIDI-Schnittstelle arbeitet nach dem Stromschleifen-Prinzip (gearbeitet wird mit einer Standardstromstärke von ca. 5 mA), d. h., "Strom" und "kein Strom" sind die Aussagen für eine logische 0 oder 1. Dieses Prinzip ist bei größeren Leitungslängen wesentlich unempfindlicher gegenüber Störungen als die Datenübertragung mit Spannungspegeln.

Die Datenübertragungsrate liegt nach der MIDI-Spezifikation bei 31,25 kBit/s. Ein Zeichen ist 8 Bit breit, umrahmt von einem Start- und Stopbit. Ein Bit ist also 32 µSek. lang; die Übertragung eines Zeichens dauert somit 320 µSek. Optokoppler an den Eingängen (MIDI-in) sorgen für eine Potentialtrennung der angeschlossenen Einheiten und verhindern so Masseverschleifungen und Brummstörungen.

Der serielle MIDI-Bus hat für jede Datenrichtung eine eigene Leitung. An MIDI-in werden Daten empfangen, über MIDI-out ausgesendet. Beim ST werden die empfangenen Daten von MIDI-in nach MIDI-out durchgeschleift. So wird die Funktion MIDI-Thru realisiert.

Den Empfang von seriellen Daten melden sowohl der MIDI- als auch der Tastatur-ACIA über ein Low-Signal auf der IRQ-Leitung. Beide ACIA-IRQ-Anschlüsse sind miteinander verbunden (Open-Collector-Ausgänge) und laufen am Eingang I4 des MFP auf.

Status-Byte	Bedeutung
Common Informations	
F1	nicht definiert !
F2	SONG-POSITION (die beiden folgenden Datenbytes enthalten LSB und MSB der SONG POSITION. (1 Count = 6 MIDI-Clocks))
F3	SONG SELECT (das folgende Datenbyte enthält die SONG-NUMBER)
F4..F5	nicht definiert !
F5	TUNE REQUEST
FF	SYSTEM RESET
Real Time Informations	
F8	TIMING CLOCK (24 Clockimpulse / Viertel-Note)
F9	nicht definiert !
FA	START (setzt SONG POSITION-Pointer zurück!)
FB	CONTINUE
FC	STOP
FD	nicht definiert !
FE	ACTIVE SENSING (wird in Pausen alle 300 ms gesendet, um Empfänger bereit zu halten!)
System Exclusive Informations	
F0	SYSTEM EXCLUSIVE (das folgende Datenbyte enthält die Herstelleridentifikation; beliebig viele Datenbytes können folgen!)
F7	END OF SYSTEM EXCLUSIVE (zeigt Ende der SYSTEM EXCLUSIV-Daten an)

Abb. 7.4: System Commands

Welcher ACIA (Tastatur oder MIDI) "interrupted" hat, muß dann in der Interruptroutine für den Port I4 des MFP, durch Abfrage der beiden ACIA-Statusregister ermittelt werden.

Auch am MIDI-in Eingang des ST findet man natürlich den MIDI-typischen Optokoppler zur Potentialtrennung (siehe Abbildung 7.5). Das empfangene MIDI-Signal wird gleich wieder an die MIDI-out Buchse (Pin 3 + 1) durchgeschleift (MIDI-Thru-Funktion).

Für die Wandlung von log. Pegeln des ACIA-Ausgangs TxDATA auf die Stromsteuerung der MIDI-Schnittstelle sorgen die Inverter, gebildet mit TTL-Chips des Typs "LS04" und "LS05".

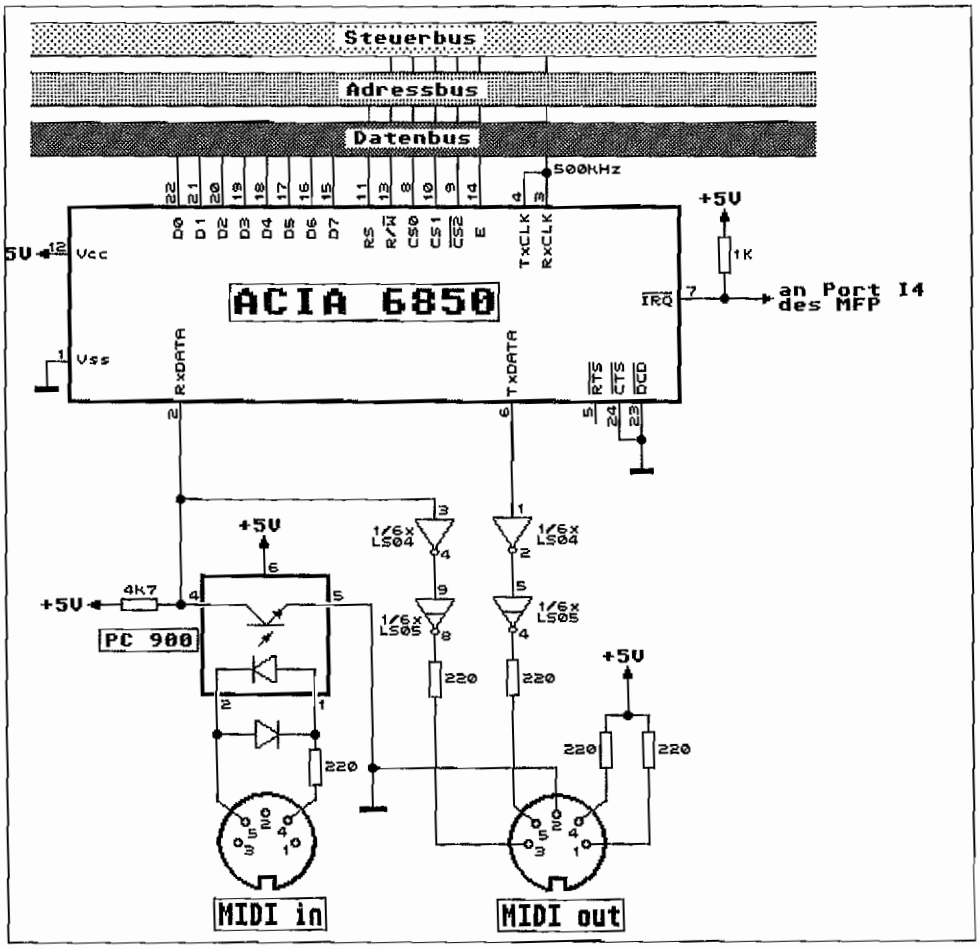


Abb. 7.5: Die MIDI-Schnittstelle des ST

Die "LS05"-Inverter besitzen dabei Open-Collector-Ausgangsstufen und schalten damit nur den Strom im Ausgangskreis ein bzw. aus. Da es bei MIDI keinen hardwaregesteuerten Handshake gibt, sind die entsprechenden Anschlüsse des ACIA (RTS, CTS, DCD) unwirksam geschaltet.

Softwaremäßig wird die Programmierung der MIDI-Schnittstelle vom TOS durch die XBIOS-Funktionen #12 ("Midiws") und #14 ("Iorec") unterstützt. "Iorec" arbeitet jedoch, im Gegensatz zur Datenübertragung über die RS232-Schnittstelle, bei MIDI-Betrieb nur mit einem Empfangspuffer.

Die Tastatur

Die üppig ausgelegte Tastatur des ST (95 Tasten) hilft dem Anwender dabei, sich dem Betriebssystem "bemerkbar" zu machen. Ein Tastendruck muß dabei in ein entsprechendes Zeichen umgeformt und dem TOS übergeben werden. Außerdem ist laufend zu überwachen, was der Anwender mit der Maus und/oder evtl. einem angeschlossenen Joystick anstellt. Um die CPU dabei möglichst wenig mit dieser eintönigen Tätigkeit (die größte Zeit vergeht dabei mit dem Warten auf einen Tastendruck oder eine Mausbewegung) zu belasten, wurde für diese Aufgabe ein eigener Mikrocontroller herangezogen.

Damit dieser sich während des Wartens auf irgendwelche Reaktionen nicht zu sehr langweilt, darf er sich auch noch gleichzeitig als Uhr betätigen. Mittels einer zusätzlichen Batterie oder eines Akkus kann man den IKBD-Chip (Intelligent-KeyBoarD-Chip) auch dann mit Energie versorgen, wenn der ST ausgeschaltet ist. So erhält man eine Uhr, die immer richtig geht (siehe auch Bauanleitung in 68000er 3/87)!

Ein eigener Computer für die Tastatur

Für diese Aufgabe wurde ein Single-Chip-Mikrocomputer der 6800-Familie eingesetzt, der "6301V1". Er enthält in einem normalen 40pol. DIL-Gehäuse eine 8-Bit-CMOS CPU, 4KByte ROM, 128 Bytes RAM, einen 16-Bit-Timer, eine serielle Schnittstelle und vier I/O-Ports mit insgesamt 29 Portleitungen! Abbildung 7.6 gibt einen Einblick in das Innenleben des IKBD-Chips.

Die Denkzentrale

Die CPU des IKBD-Chips enthält zwei 8-Bit-Akkumulatoren A und B, die auch zusammen als ein 16-Bit-Akku D arbeiten können. Weiterhin existiert noch ein 16-Bit-Index-Register

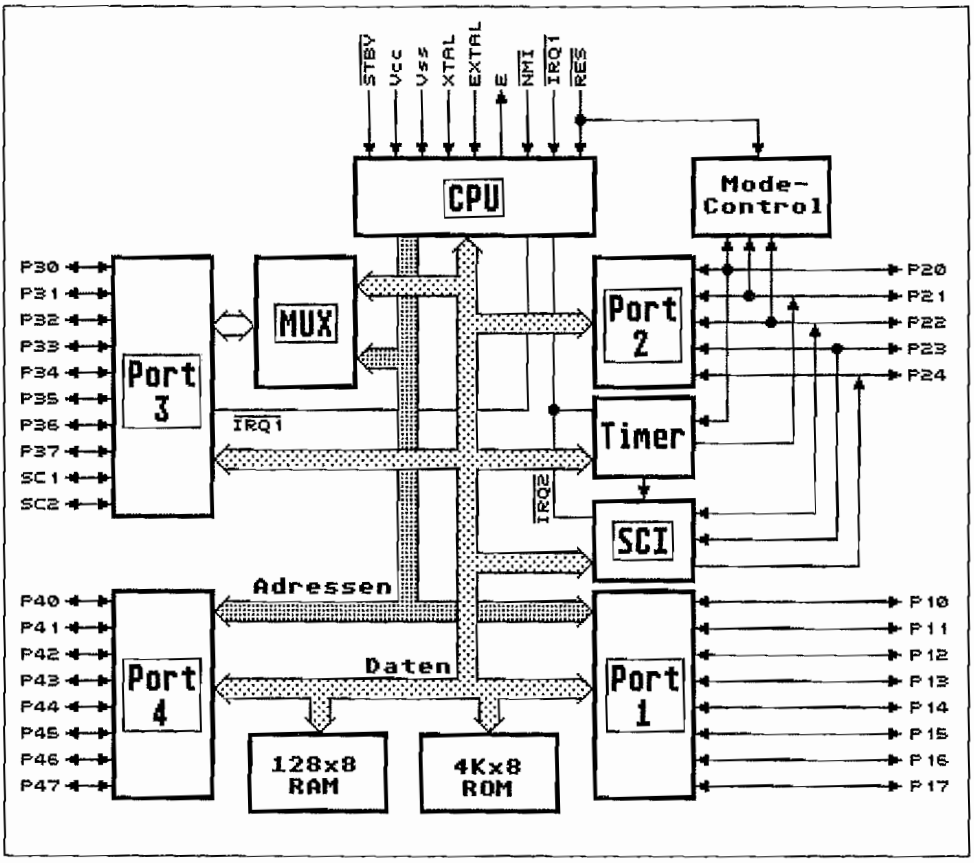
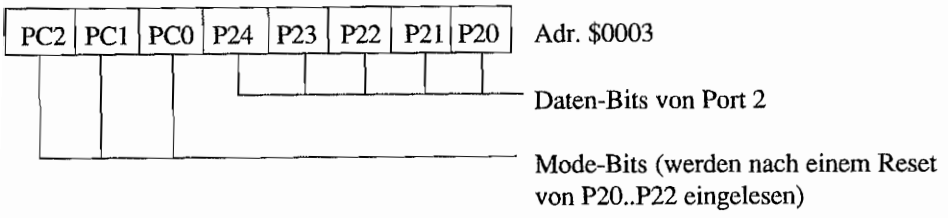


Abb. 7.6: Das Innenleben des Tastaturprozessors

(X) und je ein 16-Bit-Stack-Pointer (SP) und Programmzähler (PC). Das Condition-Code-Register (CCR) mit den Flags für Carry, Overflow, Zero usw. fehlt natürlich ebenfalls nicht. Der 6301 ist im Objektcode aufwärts-kompatibel zum 6801 und damit zum 6800.

Mehrere Betriebsarten zur Auswahl

Im Blockschaltbild findet sich, neben bekannten Funktionsblöcken, wie man sie von anderen Mikrocontrollern kennt, auch ein sogenannter Mode-Control-Block. Dieser Mode-Control-Block wird bei einem Reset benutzt, um festzustellen, in welcher Betriebsart der Mikrocontroller arbeiten soll.



Im ST arbeitet der 6301 im Single-Chip-Modus (Mode 7). Alle Ports werden dabei für I/O-Zwecke benutzt. Der Expanded-Multiplex-Mode (Mode 0, 2, 4 oder 6) wird benutzt, um den Adreßraum des 6301 mit externen ROMs/RAMs auf bis zu 64 KBytes zu erweitern. Port 3 wird dabei zu einem gemultiplexten Daten-/Adreßbus umfunktioniert.

Weitere acht Adreßleitungen können an Port 4 zur Verfügung gestellt werden.

Im Expanded-Non-Multiplexed-Mode (Mode 1 und 5) kann der 6301 ebenfalls bis zu 64 KByte adressieren. Port 3 wird dann für den Datenbus, Ports 1 und 4 für den Adreßbus benutzt. Außerdem kann der 6301 in diesem Modus Peripherie-Bausteine der 6800-Familie direkt (ohne Adressen-Zwischenspeicherung) ansprechen. Welcher Modus gewünscht ist, erfährt der 6301 bei einem RESET über die Portleitungen P20..P22 von "außen".

Softwaremäßig läßt sich der Modus dann nicht mehr ändern.

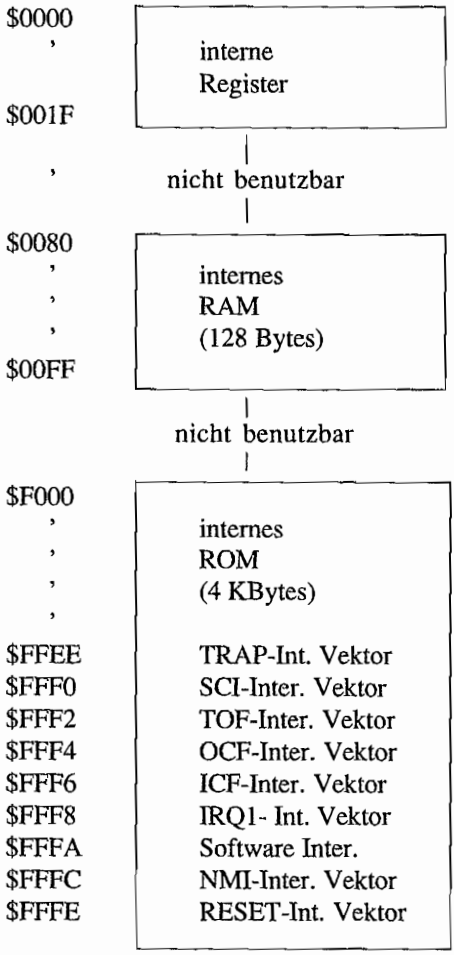
Wie man aus dem Schaltbild der ST-Tastatur (siehe Abbildung 7.7) ersehen kann, liegen die drei Anschlüsse P20...P22 beim ST über Pull-up-Widerstände auf High (drei "111" ergibt Mode 7!). Man kann beim ST jedoch einen anderen Modus einstellen, indem man bei einem RESET des ST eine oder beide Maustasten gedrückt hält. Damit "zieht" man den Eingang P21, P22 oder beide auf Low-Pegel und setzt im Mode-Register einen Modus von 1, 3 oder 5.

Das führt aber nur dazu, daß man mit der Tastatur und der Maus nicht mehr arbeiten kann, da der Tastaturprozessor ja von Atari als Single-Chip-Anwendung (Modus 7) konzipiert ist.

Weil Modus 5 und 7 nahezu identisch sind, darf man jedoch beim Einschalten seines STs ruhig die linke Maustaste gedrückt halten!

Das Gedächtnis des Tastaturchips

Die Funktionsblöcke RAM und ROM brauchen eigentlich keine weitere Erläuterung. Die 128 Bytes RAM werden bei der ST-Tastatur für die Zwischenspeicherung von Tastencodes und Mausaktionen sowie für einige Pointer, den Stack, Uhrzeitdaten usw. benutzt. Im ROM ist das "Betriebssystem" des IKBD-Chips untergebracht.



Memory Map des 6301 im Mode 7 (Single-Chip-Modus)

Wie spät ist es?

Der Timer-Block des Tastaturprozessors enthält einem freilaufenden 16-Bit-Zähler. Dieser Zähler wird mit 1/4 der Systemtaktfrequenz (das entspricht dem Takt des E-Anschlusses) hochgezählt. Im ST arbeitet der 6301 mit 4 MHz Systemtakt, so daß der 16-Bit-Zähler des Timers mit 1 MHz gefahren wird. Der Zähler kann jederzeit ausgelesen werden. Ein Beschreiben mit einem neuen Startwert ist ebenfalls möglich, dabei sind jedoch Doppel-Byte-Schreibzugriffe zu verwenden (16-Bit-Register!).

16 Bit-Free-Running-Counter (FRC)

Adr. $0009	Adr. $000A
XXXXXXXX	XXXXXXXX
High-Byte	Low-Byte

Es können IRQ2-Interrupts durch den Timer ausgelöst werden, sobald ein bestimmter Zählerstand erreicht ist oder sobald dieser von $FFFF -> $0000 zählt (Timer Overflow). Auch durch ein Triggerereignis von außen kann der augenblickliche Zählerstand "eingefangen" und so der Timer für Zeitmessungen benutzt werden.

16 Bit-Output-Compare-Register (OCR)

Adr. $000B	Adr. $000C
XXXXXXXX	XXXXXXXX
High-Byte	Low-Byte

Erreicht der FRC den im OCR eingestellten Wert, kann über P21 ein Ausgangsimpuls abgegeben und ein Interrupt ausgelöst werden.

16 Bit-Input-Capture-Register (ICR)

Adr. $000D	Adr. $000E
XXXXXXXX	XXXXXXXX
High-Byte	Low-Byte

Durch Flankenwechsel an P20 kann der augenblickliche FRC-Zählerstand im ICR "eingefangen" werden. Gleichzeitig kann ein Interrupt ausgelöst werden.

8 Bit-Timer-Control/Status-Register (TCSR)

ICF	OCF	TOF	EICI	EOCI	ETOI	IEDG	OLVL

Adr. $0008

ICF (Input-Capture-Flag). Gesetzt, sobald ein entsprechend dem IEDG-Bit eingestellter Flankenwechsel stattgefunden hat.

OCF (Output-Capture-Flag). High, sobald Wert im OCR mit FRC übereinstimmt.

TOF (Timer-Overflow-Flag). Gesetzt bei $FFFF -> $0000-Übergang des Free-Running-Counters (FRC).

EICI (Enable-Input-Capture-Interrupt). Bei gesetztem Bit kann eine Interrupt-Anforderung durch entsprechenden Flankenwechsel an P20 ausgelöst werden.

EOCI (Enable-Output-Capture-Interrupt). Freigabe der Interrupt-Anforderung bei Übereinstimmung von OCR und FCR.

ETOI (Enable-Timer-Overflow-Interrupt). Bei gesetztem Bit wird ein Interrupt bei Timer Overflow ($FFFF -> $0000-Übergang) ausgelöst.

IEDG (Input-Edge). Bei gesetztem Bit erfolgt "Input-Capture" ("Einfangen des augenblicklichen FRC-Wertes") bei ansteigender Flanke an P20. "Input-Capture" auf fallender Flanke erfolgt bei gelöschtem Bit. Der P20-Anschluß muß natürlich als Eingang programmiert sein.

OLVL (Output-Level). Tritt "Output-Capture" ein (Übereinstimmung zwischen OCR und FRC), wird der OLVL-Bit-Wert am Port P21 präsentiert (natürlich nur, wenn der P21-Port als Ausgang programmiert ist).

Daten im Gänsemarsch

Der SCI-Block enthält das sogenannte Serial-Communications-Element, die serielle Schnittstelle des 6301. Es handelt sich hierbei um eine Vollduplex-Schnittstelle, mit der dem ST die Informationen über betätigte Tasten, Mausbewegungen, Mausclicks und Uhrzeit übermittelt werden. Das Datenformat beträgt dabei acht Datenbits mit einem vorangestellten Startbit und zur Signalisierung des Zeichenendes ein Stopbit.

Es gibt auch hier, wie schon bei den ACIAs, ein Sende- und ein Empfangsregister. In einem Rate/Mode Control-Register wird die Datenübertragungsrate (im ST wird mit Systemtakt/512 = E/128 = 1 MHz/128 = 7812,5 Bit/s gearbeitet) eingestellt und ausgewählt, ob der erforderliche Takt von einer externen Taktquelle kommt oder intern erzeugt wird.

Ein Tx/Rx-Control-/Statusregister gibt Auskunft über evtl. Fehler bei der Datenübertragung und ob Daten empfangen wurden und zur Weiterverarbeitung ausgelesen werden sollen oder ob neue Daten ins Senderegister geschrieben werden können.

Senderegister	**Empfangsregister**
Adr. $0013	Adr. $0012
xxxxxxxx	xxxxxxxx

Transmit/Receive-Control and Status-Register (TRCSR)

RDRF	ORFE	TDRE	RIE	RE	TIE	TE	WU

Adr. $0011

RDRF (Receive-Data-Register-Full). Gesetzt, sobald ein empfangenes Zeichen aus dem Empfangsregister abgeholt werden kann.

ORFE (Over-Run-Framing-Error). Gesetzt, wenn ein Framing-Error oder ein Überlauf (empfangenes Zeichen wurde nicht rechtzeitig aus Empfangsregister ausgelesen) aufgetreten ist.

TDRE (Transmit-Data-Register-Empty). Wird gesetzt, sobald das Senderegister "nachgeladen" werden kann.

RIE (Receive-Interrupt-Enable). Durch Setzen dieses Bits wird der Empfänger-Interrupt freigegeben, der bei RDRE=1 od. ORFE=1 ausgelöst wird.

RE (Receiver-Enable). Mit Setzen des Bits wird Port P23 zum Empfangsanschluß des SCI. Der Empfangsteil ist eingeschaltet. (Im ST realisiert.)

TIE (Transmit-Interrupt-Enable). Bei gesetztem Bit ist der Sender-Interrupt freigegeben, der bei TDRE=1 ausgelöst wird.

TE (Transmit-Enable). Mit Setzen dieses Bits wird Port P24 als Senderausgang geschaltet. Der Sendeteil des SCI ist eingeschaltet. (Im ST realisiert.)

WU (Wake-Up). Wird dieses Bit gesetzt, so werden Empfangsdaten so lange ignoriert, bis zehn 1-Bits empfangen werden. Mit einer Folge von zehn 1-Bits wird also der Empfänger erst "aufgeweckt".

Rate- and Mode-Control-Register (RMCR)

–	–	–	–	CC1	CC0	SS1	SS0

Adr. $0010

SS1 SS0
- 0 0 — Baudrate = E/16
- 0 1 — Baudrate = E/128 (im ST verwendet)
- 1 0 — Baudrate = E/1024
- 1 1 — Baudrate = E/4096

(E=1/4 des Systemtakts)

CC1 CC0
- 0 0 keine Funktion
- 0 1 interner Takt (im ST verwendet)
- 1 0 interner Takt an P22 herausgeführt
- 1 1 externer Takt von P22

Die Verbindung zur Außenwelt

Die vier Ports mit ihren insgesamt 29 Portleitungen werden bei der "intelligenten Tastatur" des ST überwiegend benutzt, um eine Tastaturmatrix zu bilden und dann "Kurzschlüsse" (Tastenbetätigungen) in dieser Matrix zu entsprechenden Tastaturcodes umzuwandeln. Mausbewegungen und Joystick-Betätigungen werden ebenfalls in dieser Matrixabfrage berücksichtigt. Lediglich die Fire Buttons/Maustasten sind nicht in die Matrix eingebunden und werden direkt an zwei Ports des 6301 abgefragt.

Beim 6301 kann im Single-Chip-Mode jeder Portanschluß individuell als Ein- oder Ausgang programmiert werden. Dazu existiert für jeden Port ein Datenrichtungsregister (Bit auf High = zugehöriger Portanschluß ist als Ausgang eingestellt).

Port #	Adresse	Bitbelegung								Bedeutung
1	$0002	P17	P16	P15	P14	P13	P12	P11	P10	Port-Datenregister
1	$0000	P17	P16	P15	P14	P13	P12	P11	P10	Datenrichtung
2	$0003	PC2	PC1	PC0	P24	P23	P22	P21	P20	Port-Datenregister
2	$0001	–	–	–	P24	P23	P22	P21	P20	Datenrichtung
3	$0006	P37	P36	P35	P34	P33	P32	P31	P30	Port-Datenregister
3	$0004	P37	P36	P35	P34	P33	P32	P31	P30	Datenrichtung
4	$0007	P47	P46	P45	P44	P43	P42	P41	P40	Port-Datenregister
4	$0005	P47	P46	P45	P44	P43	P42	P41	P40	Datenrichtung

Eine Sonderstellung nimmt der Port 2 im ST ein. Port P23 ist als Empfangsport für serielle Daten geschaltet. Die serielle Ausgabe von Daten erfolgt über Port P24. Im ST sind diese beiden Ports mit den TxDATA- und RxDATA-Anschlüssen des Tastatur-ACIAs verbunden. Immer wenn ein Zeichen von der Tastatur vorliegt, welches dem TOS bekanntgemacht werden sollte, wird dieses zum Tastatur-ACIA im ST gesendet. Dieser löst einen Interrupt über Port 4 des MFP aus und informiert so das Betriebssystem über das Eintreffen eines Tastaturzeichens.

Port P21 fragt im ST die linke Maustaste bzw. den Fire-Button von Joystick #0 ab. Die rechte Maustaste bzw. Fire-Button von Joystick #1 wird über Port P22 erkannt. Die Tasten schließen bei Betätigung einen Kontakt nach Masse, so daß bei Betätigung ein Low am entsprechenden Portanschluß anliegt.

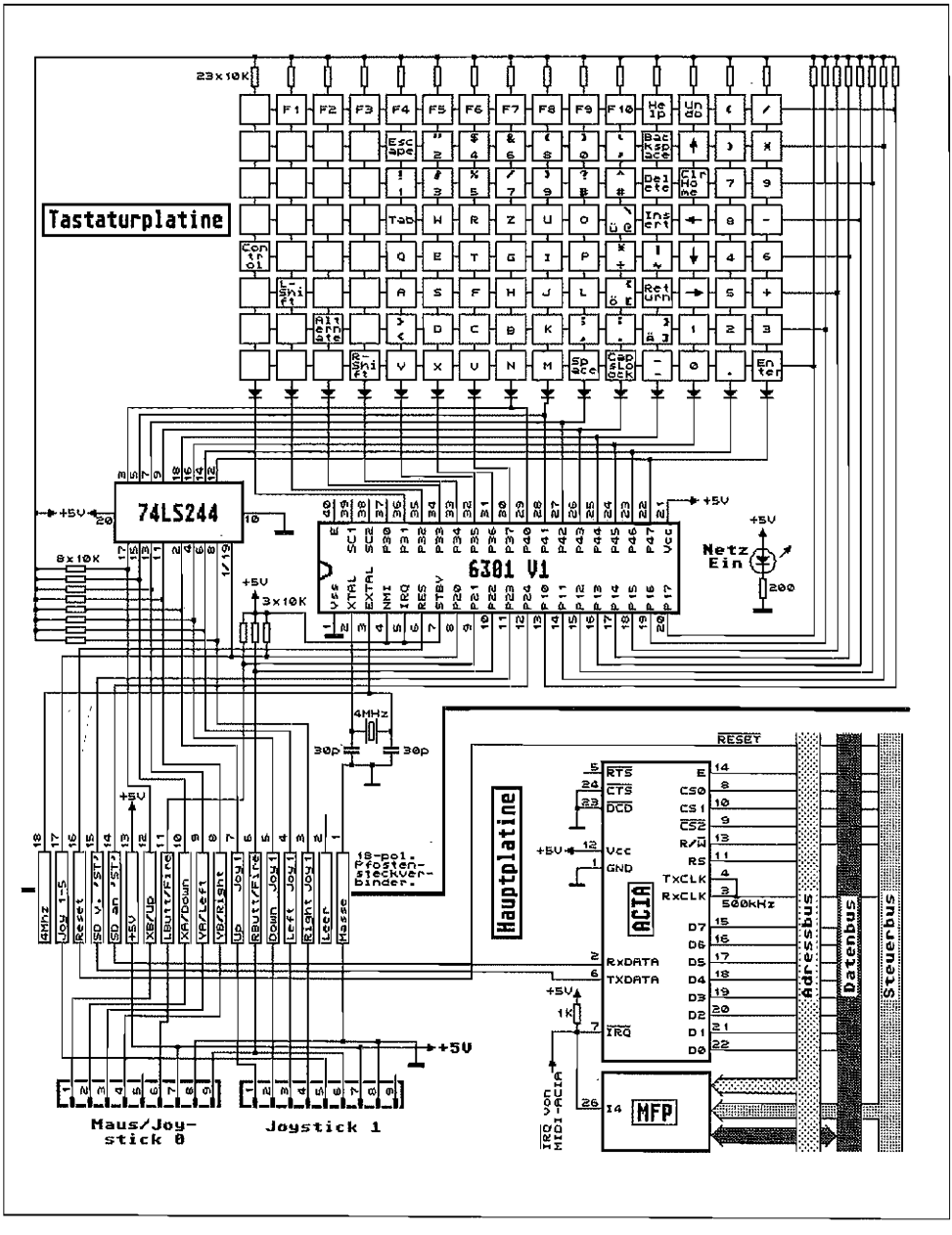


Abb. 7.7: Der Anschluß der intelligenten Tastatur an den ST

Und so funktioniert die Tastaturdekodierung:

- Der Tastaturprozessor legt nacheinander an jeden Port P31..P47 ein Low-Signal. Jede Spalte der Tastaturmatrix bekommt also einmal ein Low angeboten.

- Wird eine Taste gedrückt, gelangt dieses Low an einen der Portanschlüsse P10...P17, welche als Eingang programmiert sind. Aus der Information, welche Spalte gerade ein Low erhält und welche Zeile dieses Low weitergibt, kann die gedrückte Taste in der Matrix ermittelt werden. Der entsprechende Tastencode (Make-Code) wird zum Tastatur-ACIA gesendet.

- Wenn die Taste wieder losgelassen wird, wird für die Taste ein sogenannter "Break-Code" gesendet (Break-Code = Make-Code .or. $80).

Was macht die Maus?

Port P20 ist im ST-Keyboard als Ausgang programmiert. Bei jedem Low an diesem Portausgang wird über den 8fach-Bustreiber (74LS244, siehe Abbildung 7.7) der Zustand der Mausbewegungssensoren (oder Richtungsinformationen von den Joysticks) an die als Eingang programmierten Ports P40..P47 gelegt.

Die so gelieferten Daten der Mausbewegungssensoren werden mit den Daten aus der letzten Mausabfrage verglichen. Aus dem Unterschied zur vorhergegangenen Abfrage wird so die Bewegungsrichtung der Maus ermittelt. Die Richtungsinformation von den Joysticks (deren Richtungskontakte schließen gegen Masse) wird ebenfalls über Port P40..P47 abgefragt.

Die Tastatur des 1040 ST unterscheidet sich ein klein wenig von der Tastatur der 260ST- und 520STM-Computer. Zum einen wird die Verbindung zu den beiden Joystick- und Mausports nicht über eine Steckverbindung auf der Hauptplatine hergestellt, sondern direkt über zwei getrennte Verbindungsstecker. Weiterhin ist eine zusätzliche Leuchtdiode für das eingebaute Disklaufwerk (Floppy-Select-Indicator) vorhanden.

Ebenso unterscheidet sich die Tastatur der MEGA-STs ein wenig von der Tastatur der anderen STs. Sie ist ja nun nicht mehr in das Gehäuse des Computers eingebaut, sondern davon abgesetzt und über ein flexibles Spiralkabel mit der Haupteinheit verbunden.

Wie der Schaltplanauszug für den Anschluß der MEGA-ST-Tastatur verdeutlicht, werden außer den Leitungen für die Stromversorgung (+5V und Masse) nur noch die TXD- (Transmit Data) und RXD-Leitung (Receive Data) benötigt. Die RESET-Leitung zum IKBD-Chip ist entfallen. Maus und Joysticks werden jetzt an der abgesetzten MEGA-ST-Tastatur ange-

schlossen. Durch die fehlende RESET-Leitung vom ST zum IKBD-Chip kann man nun eine "abgestürzte" MEGA-Tastatur nur noch durch einen Kaltstart (Computer ausschalten – etwas warten – Computer wieder einschalten) zur korrekten Funktion bewegen.

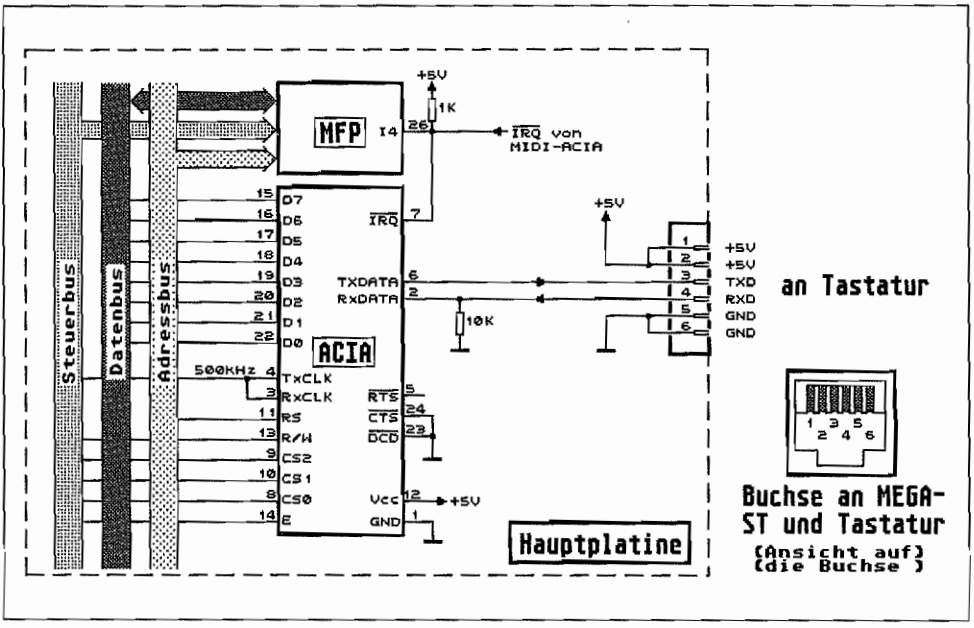


Abb. 7.8: So wird die Tastatur an den MEGA-ST angeschlossen

Ein flinkes Tierchen am ST – Die Maus

Erwähnt wurde sie ja schon bei der Beschreibung der intelligenten Tastatur – die Maus. Ohne sie wäre das Arbeiten nur halb so komfortabel (besonders bei Zeichen- und CAD-ähnlichen Programmen).

Die Atari-Maus arbeitet nach dem opto-mechanischen Prinzip. Die Bewegungsinformation wird über eine Vollgummikugel auf zwei in Kugellagern laufende Walzen übertragen. Die Walzenachsen sind rechtwinklig zueinander angeordnet. Eine Walze wird also bei Bewegung in horizontaler Richtung (X-Richtung) und die andere bei vertikaler Bewegung (Y-Richtung) angetrieben. Auf jeder Walze sitzt nun eine Lochscheibe, welche bei Bewegung zyklisch den Lichtstrom zweier Lichtschranken unterbricht. Die beiden Lichtschranken sitzen sich auf der Lochscheibe genau gegenüber.

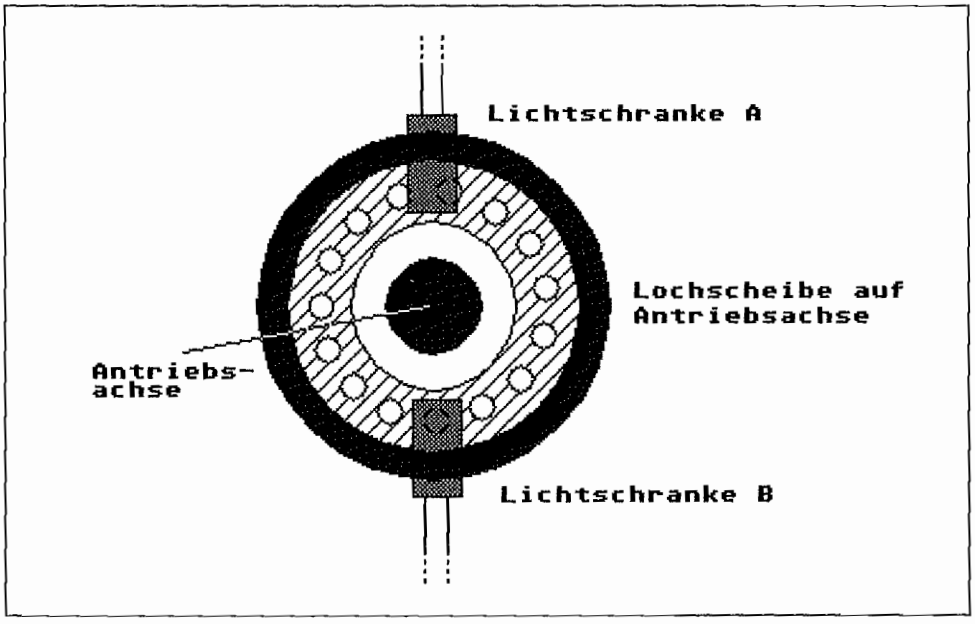


Abb. 7.9: Die optische Abtastung der Bewegung bei der Maus

In der Lochscheibe befindet sich aber eine ungerade Zahl von Löchern, so daß beide Lichtschranken nie gleichzeitig unterbrochen werden. Tastet man nun die Ausgänge der Lichtschranken laufend ab, so kann aus der Information welche der beiden Schranken einer Achse zuerst unterbrochen wird, die Drehrichtung der Achse erkannt werden. Die Häufigkeit der Lichtstrahl-Unterbrechungen in einem Zeitintervall ist ein Maß für die Geschwindigkeit der Mausbewegung. Mit dieser verhältnismäßig einfachen Mechanik und Elektronik erzielt die Atari-Maus immerhin eine Auflösung von ca. vier Schritten/mm. Bewegungsgeschwindigkeiten bis zu 250 mm/Sekunde werden noch korrekt erfaßt. In Abbildung 7.10 ist die Schaltung der verhältnismäßig einfachen Mauselektronik dargestellt. Die Versorgungsspannung wird vom ST geliefert und ist in der Maus nochmals extra mit Glättungselkos gesiebt.

Die von den Fototransistoren gelieferten Impulse werden über die als Schmitt-Trigger geschalteten Komparator-Stufen in "vernünftige", steilflankige Rechteckimpulse umgewandelt, mit denen der IKBD-Chip etwas anfangen kann.

Als einziges "aktives" Bauelement (neben den LEDs und Fototransistoren) findet man in der Maus nur einen 4fach-Komparator (Spannungsvergleicher) des Typs "339", allerdings in seiner "niedlichsten" Form, nämlich in der SMD-Ausführung (wie übrigens auch die verwende-

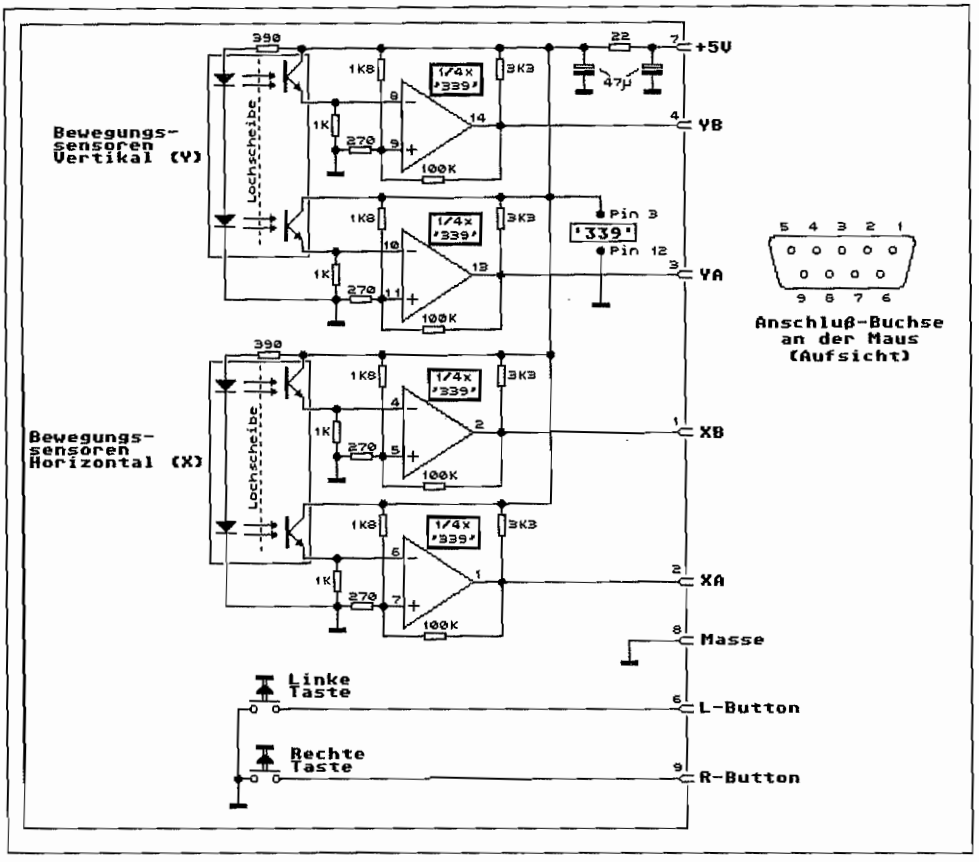


Abb. 7.10: Ohne sie läuft am ST nicht viel: die Maus

ten Widerstände! Ja, ganz recht, was auf der Unterseite der Mausplatine so aussieht wie "Mäusedreck", sind die SMD-Widerstände).

(SMD = Surface-Mounted-Device = zu deutsch etwa: direkt mit der Platinenfläche verbundenes Bauelement; SMDs werden ohne Anschlußdrähte direkt auf die Platinenoberfläche gelötet, was eine sehr kompakte Bauform ermöglicht.)

In Abbildung 7.11 sind zur Verdeutlichung noch die Ausgangssignale der Komparatoren aufgeführt. Man erkennt deutlich, daß bei positiver Bewegungsrichtung der XA- (YA)-Ausgang eher High wird als der XB- (YB)-Ausgang. Bei negativer Bewegungsrichtung ist es dann genau andersrum. So erkennt der Tastaturprozessor dann die Bewegungsrichtung der Maus.

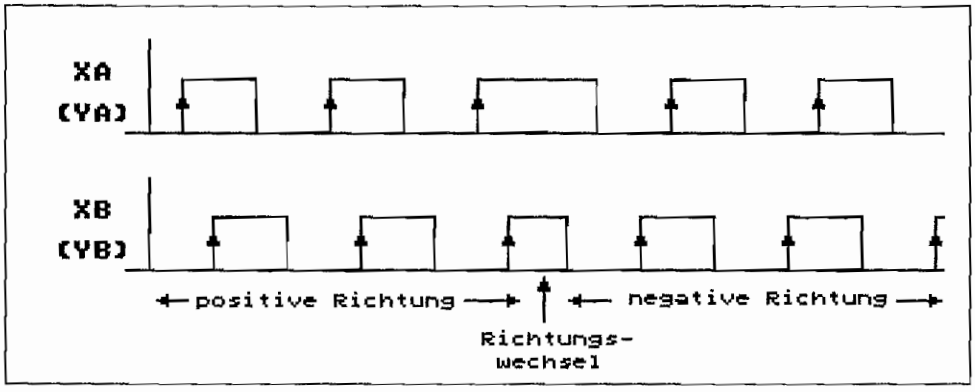


Abb. 7.11: Die Ausgangssignale der Mausbewegungssensoren

Programmierung des IKBDs

Der IKBD-Chip im ST kommuniziert mit seiner Umwelt, wie bereits weiter vorn gesehen, über eine eigene serielle Schnittstelle und kann darüber auch Daten empfangen.

Kommandos an den IKBD kann man sehr leicht mittels der XBios-Funktion "Ikbdws" übertragen. Mit dem Empfangen der Daten ist es schon erheblich komplizierter: Dazu muß man sich zunächst mittels der XBios-Funktion "Kbdvbase" die Adresse der Tabelle der IKBD-Routinen besorgen. Je nachdem, ob man Uhrzeit-, Maus-, Joystick-, Tastatur- oder Statusinformationen abfragen will, muß man an der entsprechenden Stelle seine eigene Empfangsroutine einsetzen. Nur so ist es beispielsweise möglich, die IBKD-Uhrzeit im 1-Sekunden-Takt abzufragen, denn XBios wirft bei der eigenen Abfrage jeweils das unterste Bit weg, um auf das MS/DOS- und GEMDOS-spezifische Datumsformat zu kommen.

Daten werden auf zwei verschiedene Arten empfangen. Da wären zunächst die Meldungen über das Drücken und Loslassen von Tasten, für die die Werte 0 bis 117 (+128 beim Loslassen der Taste) reserviert sind (die IKBD-eigenen Codes, auch Scancodes genannt, finden Sie im Anhang). Weitere Codes leiten jeweils ein sogenanntes "Datenpaket" ein, das mehrere Bytes umfassen und verschiedenste Informationen enthalten kann. Nun jedoch zu den einzelnen IBKD-Funktionen:

Reset (IBKD $80, $01)

Schaltet den IKBD-Chip in den Anfangszustand zurück. Dabei geht die Uhrzeit *nicht* verloren (das passiert nur bei Unterbrechung der Stromzufuhr). Außerdem wird ein Keyboard-Selbst-

test durchgeführt, dessen Erfolg durch den Wert $F1 zurückgemeldet wird. Dabei wird die angeschlossene Tastatur auf Funktionstüchtigkeit getestet. So wird dann für jeden geschlossenen Tastenkontakt angenommen, daß die betreffende Taste defekt ist. Für alle solche Tasten wird der Break-Code gesendet (das heißt, es wird ein Loslassen dieser Taste gemeldet). Dieser Selbsttest wird in der momentanen Version von TOS ignoriert.

Set mouse button action (IKBD $07, Modus)

Mit diesem Kommando kann man festlegen, wie die beiden Tasten der Maus behandelt werden sollen. Dabei kann man einerseits die Maustasten wie "Alt/Insert" und "Alt/ClrHome" behandeln lassen oder festlegen, ob eine Meldung bei Drücken oder Loslassen der Maustaste gesendet werden soll (die beiden untersten Bits sind nur im absoluten Modus relevant, der Standardwert für Modus ist 0):

Modus = 4: Maustasten wie Tastatur behandeln
Modus = 2: Loslassen der Maustaste melden
Modus = 1: Drücken der Maustaste melden

Set relative mouse position reporting (IKBD $08)

Schaltet den relativen Mausbewegungsmodus ein, der auch der vom AES benutzte Standardmodus ist. In dieser Betriebsart wird immer dann ein Mausereignis gemeldet, wenn die Maus um einen mittels "Set mouse threshold" festlegbaren Betrag bewegt oder eine der Maustasten gedrückt wurde. Geliefert werden Status-Pakete folgender Form:

```
typedef struct
{
     har header;      /* 0xF8 bis 0xFB - die beiden untersten Bits
                         geben den Status der Maustasten an */
     char dx,dy;      /* Relative Position, zwischen -128 und 127 */
} RELMAUS;
```

Sollte der Darstellungsbereich von –128 bis 127 für die erfolgte Mausbewegung nicht ausreichen, dann wird ein weiteres Paket geschickt.

Set absolute mouse positioning (IKBD $09, xmax, ymax)

Schaltet auf den absoluten Mausbewegungsmodus. In dieser Betriebsart meldet der IKBD, der Mausposition entsprechend, absolute Koordinaten. Dazu muß man in den beiden 16-Bit-Werten xmax und ymax die Maximalzahl für die beiden Koordinatenrichtungen angeben. Bewegungen unterhalb der Nullposition und oberhalb der angegebenen Grenzen werden ignoriert.

Set mouse keycode mode (IBKD $A, dx, dy)

In dieser Betriebsart kann man mit der Maus das Drücken der Cursor-Tasten emulieren (das bietet sich bei allen TOS-Applikationen mit Cursor-Steuerung an. Aus einem biederen Screen-Editor wird damit ein rasantes Mausprogramm!). Als Parameter muß man die Anzahl von horizontalen bzw. vertikalen Koordinatenschritten angeben, die je einem Tastendruck entsprechen sollten (10 ist dafür ein brauchbarer Wert).

Set mouse threshold (IKBD $B, x, y)

Setzt die Ansprechschwelle für Mausbewegungen für beide Richtungen separat (nur für den relativen Modus relevant). Standardwert nach RESET ist 1.

Set mouse scale (IKBD $C, x, y)

Hiermit kann die Maus-Skalierung für beide Richtungen gesetzt werden. Man legt fest, um wieviel die Maus bewegt werden muß, damit die IKBD-internen Positionszähler um 1 erhöht bzw. vermindert werden. Dieser Befehl kann nur für den absoluten Modus benutzt werden.

Interrogate mouse position (IKBD $D)

Im absoluten Modus kann hiermit die Position der Maus abgefragt werden. Der IKBD liefert ein Statuspaket folgenden Formats zurück:

```
typedef struct
{
     char header;     /* 0xF7 = absolute mouse position header */
     char buttons;    /* Status der Mausknöpfe */
     int  x;          /* X-Koordinate */
     int  y;          /* Y-Koordinate */
} ABSMAUS;
```

In "buttons" werden die untersten vier Bits folgendermaßen benutzt:

Bit 0: rechter Knopf seit letzter Abfrage gedrückt
Bit 1: rechter Knopf seit letzter Abfrage losgelassen
Bit 2: linker Knopf seit letzter Abfrage gedrückt
Bit 3: linker Knopf seit letzter Abfrage losgelassen

Load mouse position (IKBD, $E, $0, x, y)

Im absoluten Modus kann man mit diesem Kommando die internen Koordinaten-Zähler des IBKD setzen (x und y sind 16-Bit-Werte!).

Set Y=0 at bottom (IKBD $F)

Setzt die Orientierung der Y-Achse (Y vermindert sich, wenn die Maus vom Anwender zurückgezogen wird).

Set Y=0 at top (IKBD $10)

Setzt die Orientierung der Y-Achse auf den Standardwert (Y erhöht sich, wenn die Maus "nach unten" gezogen wird).

Resume (IKBD $11)

Mit diesem Kommando kann der Datentransfer vom IKBD wieder gestartet werden, nachdem er mittels "Pause output" gestoppt worden war.

Da der IKBD sowieso bei Erhalt eines Kommandos die Datenübertragung startet, ist "Resume" eigentlich nur ein NOP-Befehl (No Operation).

Disable mouse (IKBD $12)

Schaltet die Mausabfrage ab, die mit "Set relative mouse position reporting", "Set absolute mouse positioning" oder "Set mouse keycode mode" wieder gestartet werden kann.

Pause output (IKBD $13)

Stoppt die Datenübertragung vom IKBD zum ST, bis ein neues gültiges Kommando (wie zum Beispiel "Resume") empfangen wird. So weit es die Buffer im IKBD zulassen, werden alle Meldungen gebuffert und dann bei der Fortsetzung der Übertragung gesendet.

Set joystick event reporting (IKBD $14)

Schaltet den Joystick-Ereignis-Modus ein (Standardeinstellung). In diesem Modus wird jedesmal, wenn ein Joystick-Kontakt geöffnet oder geschlossen wurde, ein entsprechendes Statuspaket gesendet.

```
typedef struct
{
     char header;     /* $FE Joystick 0, $FF Joystick 1 */
     char val;        /* Bits 0-3 für die vier Bewegungsrichtungen,
                         Bit 7 für den Feuerknopf */
} JOYEVENT;
```

Set joystick interrogation mode (IKBD $15)

Schaltet die automatische Meldung von Joystick-Ereignissen ab. In diesem Modus muß man für jede Joystickabfrage ein spezielles Kommando ("Joystick Interrogation") senden.

Joystick interrogation (IKBD $16)

Diese Funktion kann in beiden Joystick-Abfrage-Modi benutzt werden und liefert ein Joystick-Statuspaket.

Set joystick monitoring (IKBD $17, rate)

In diesem Betriebsmodus wird in bestimmten Zeitabständen (berechnet sich zu Rate/100 in Sekunden) der Status der Joysticks gemeldet. Alle anderen Aktivitäten werden gestoppt. Die Statuspakete von zwei Bytes Länge haben folgende Form:

1. Byte:	Bit 0:	Feuerknopf Joystick 1
	Bit 1:	Feuerknopf Joystick 0
2. Byte:	Bit 0–3:	Richtungen Joystick 1
	Bit 4–7:	Richtungen Joystick 2

Set fire button monitoring (IKBD $18)

In diesem Modus wird permanent der Status des Feuerknopfs von Joystick 1 gemeldet. Die Daten werden in gepackter Form in Bytes gesendet (also immer acht Werte innerhalb eines Bytes). Die Abfragerate ist ungefähr so groß, daß in derselben Zeit, in der auch ein Byte gesendet wird, wieder acht neue Werte abgefragt werden. Daher sollte man die ankommenden Bytes möglichst in konstanten Abständen abrufen.

Set joystick keycode mode (IKBD $19, rx, ry, tx, ty, vx, vy)

In diesem Betriebsmodus wird mit Joystick 0 (also einem Joystick, der im Mausport steckt) die Cursor-Tastatur emuliert. Dabei handelt es sich bei den sechs Parametern um Zeitangaben in 1/10 Sekunden für die beiden Richtungen. Mit Schließen eines Joystick-Kontakts beginnt ein interner Zähler zu laufen. Die Arbeitsweise sei an folgendem Beispiel für die X-Richtung veranschaulicht (r=6, t=2, v=1):

Zeit (1/10 sec)	Tastendruck
0	(1) <- Intervalle von t/10
1	–

Zeit (1/10 sec)	Tastendruck
2	(2)
3	–
4	(3)
5	–
6	(4) <- Ablauf von r/10, nun Intervalle von v/10
7	(5)
8	(6)
9	(7)
10	(8)
.	.
.	.

So läßt sich sehr leicht eine Autorepeat-Funktion erreichen. Ist dieser Effekt nicht erwünscht, dann kann man "rx" bzw. "ry" auf 0 setzen.

Disable joysticks (IKBD $1A)

Joystickabfrage beenden. Sie kann mit irgendeinem gültigen, den Joystick betreffenden Kommando wieder in Gang gesetzt werden.

Time-of-day clock set (IKBD $1B, jahr, monat, tag, stunde, minute, sec)

Setzt die Uhr im IKBD-Chip. Jeder der sechs Parameter ist eine zweistellige, gepackte BCD-Zahl.

Beispiel: tag=31 für den 31. eines Monats

Jeder einzelne Wert wird separat auf Plausibilität geprüft. Gibt man beispielsweise für den Monat eine 13 an, wird das Monatsfeld nicht geändert. Akzeptiert wird dagegen ein Datum wie der 30.02.1987!

Interrogate time-of-day clock (IKBD $1C)

Fragt die Uhrzeit des IKBD-Chips ab. Als Resultat wird ein Statuspaket folgender Art gesendet:

```
typedef struct
{
     char header;     /* $FC, Uhrzeit-Statuspaket */
```

```
    char jahr;      /* Jahreszahl in gepackter BCD-Form */
    char monat;     /* Monatszahl in gepackter BCD-Form */
    char tag;       /* Tageszahl in gepackter BCD-Form */
    char stunde;    /* Stundenzahl in gepackter BCD-Form */
    char minute;    /* Minutenzahl in gepackter BCD-Form */
    char sec;       /* Sekundenzahl in gepackter BCD-Form */
} TIMEOFDAY;
```

Memory load (IKBD $20, adresse, anzahl, (daten))

Mit dieser Funktion kann man das RAM des IKBD beschreiben (der RAM-Bereich "erstreckt" sich über den Bereich von $80 und $FF). Dazu übergibt man in "adresse" (16 Bit) die gewünschte Adresse und in "anzahl" die Zahl der zu übertragenden Bytes. Es folgen dann die einzelnen Bytes, wobei der Zeitabstand zwischen den einzelnen Datenbytes maximal 20ms betragen darf. (Null-Bytes können wegen eines Fehlers im IKBD-Betriebssystem nicht übertragen werden!)

Memory read (IKBD $21, adresse)

Mit dieser Funktion kann man den Speicher des IKBD auslesen. Als Parameter übergibt man die gewünschte Anfangsadresse als 16-Bit-Wert. Der IKBD liefert daraufhin folgendes Datenpaket:

```
typedef struct
{
    char header1;   /* $F6 - Header für Statuspakete */
    char header2;   /* $20 - Header für Memory read */
    char data[6];   /* 6 Bytes Speicherinhalt ab adresse */
} MEMREAD;
```

Controller execute (IKBD $22, adresse)

Startet ein Unterprogramm (beginnend bei "adresse", 16-Bit) im IKBD. Um diesen Befehl zu nutzen, müßte man natürlich das Betriebssystem des IKBD kennen oder selbst ein Programm in den "freien" Speicher übertragen haben...

Status inquiries (IKBD $87-$9A)

Mit Hilfe der Statusabfrage-Kommandos kann man den IKBD auffordern, jeweils den Status verschiedener Betriebszustände zurückzumelden. Das Kommando entspricht jeweils einem

der betreffenden SET-Kommandos mit gesetztem achten Bit. Als Resultat liefert der IKBD immer ein 8-Byte-Paket folgender Form:

```
typedef struct
{
     char header;     /* 0xf6 = status inquiry header */
     char vals[7];    /* Fünf weitere Werte */
} IKBDSTAT;
```

Das erste Byte dieser Datenstruktur ist immer das Headerbyte $F6. Die Bedeutung der restlichen sieben Bytes ist für die einzelnen Abfrage-Kommandos jeweils verschieden. Sollten weniger als sieben Bytes belegt sein, dann wird mit Nullen aufgefüllt.

Das Statuspaket ist so geformt, daß die eingegangene Statusmeldung (unter Weglassen des Header-Bytes) direkt wieder an den IKBD geschickt werden kann. Eventuell abschließende Nullen werden dabei vom IKBD ignoriert. Im folgenden wird jeweils die Belegung des vals[]-Arrays angegeben.

Request mouse button action (IKBD $87)

Fragt den Maustastenmodus ab. Liefert {7, Modus, 0, 0, 0, 0, 0 }.

Request mouse mode (IKBD $88, IKBD $89, IKBD $8A)

Fragt den aktuellen Mausbewegungsmodus ab.

Im relativen Modus:	{8,0,0,0,0,0,0}
Im absoluten Modus:	{9,xmax (MSB),xmax (LSB),ymax (MSB),ymax (LSB),0,0}
Im Keycode-Modus:	{10,dx,dy,0,0,0,0}

Wie man sieht, kann man die sieben Bytes, die man nach dem Statusbyte erhält, direkt als Kommando für den IKBD "recyceln".

Request mouse threshold (IKBD $8B)

Fragt die Ansprechschwelle für Mausbewegungen im relativen Modus ab. Liefert {11,x,y,0,0,0,0}.

Request mouse scale (IKBD $8C)

Fragt die Maus-Skalierung ab. Liefert {12,x,y,0,0,0,0}.

Request mouse vertical coordinates (IKBD $8F, IKBD $90)

Liefert { 15, 0, 0, 0, 0, 0, 0 } für Y=0 am unteren Rand und { 16, 0, 0, 0, 0, 0, 0 } für Y=0 am oberen Rand.

Request mouse availability (IKBD $92)

Liefert { 0, 0, 0, 0, 0, 0, 0 } bei eingeschalteter Maus und sonst { 18, 0, 0, 0, 0, 0, 0 }.

Request joystick mode (IKBD $94, IKBD $95, IKBD $99)

Fragt den aktuellen Joystick-Modus ab. Liefert {20,0,0,0,0,0,0} für den normalen Modus ("event reporting"), {21,0,0,0,0,0,0} für den Nachfrage-Modus ("joystick interrogation mode") oder {25,rx,ry,tx,ty,vx,vy} für den Keycode-Modus.

Request joystick availability (IKBD $9A)

Liefert {0,0,0,0,0,0,0} bei eingeschalteter Joystick-Abfrage und sonst {26,0,0,0,0,0,0}.

Wer sich genauer mit dem Innenleben des Tastaturprozessors auseinandersetzen will, sei auf die Artikelserie "Licht in die Geheimnisse des Tatstaturprozessors" in der ST-Computer 3/90..5/90 hingewiesen. Beim Autor Sieghard Schäfer (Hallo Sieghard! Hat mir gut gefallen die Serie) kann man auch ein disassembliertes und kommentiertes Listing vom IKBD-Betriebssystem erhalten.

Kapitel 8: Das Floppy-Disk-Interface

Die ATARI ST-Computer haben alle nötige Hardware zur Steuerung von Floppy-Disk-Laufwerken bereits im Grundgerät integriert. Bei den Computern des Typs ATARI 1040STF und den MEGA STs ist außerdem ein Disklaufwerk eingebaut.

Serienmäßig werden von ATARI für die ST-Computer 3,5-Zoll-Disklaufwerke verwendet. Diese Laufwerke ermöglichen heute eine Speicherkapazität von mindestens 720 KByte je Disk bei einer kleinen, kompakten und gegen äußere Einflüsse relativ gut geschützten Bauform (unbeabsichtigtes Verknicken und Verschmutzen und Beschädigungen durch knabbernde Wellensittiche und Kanarienvögel werden wirkungsvoll verhindert.)

Für die Computer der ST-Serie wird inzwischen nur noch das zweiseitige Disklaufwerk angeboten. Zur Datenspeicherung wird die Vorder- und Rückseite der Diskette verwendet. Die Daten werden hierbei in 80 Spuren (Tracks) auf der Diskette verstreut. Jede Spur ist in neun Sektoren mit je 512 Bytes Datenkapazität aufgeteilt.

Die Datenspeicherung auf Diskette

Die auf einer Disk gespeicherten Informationen sind eine Aneinanderreihung von in verschiedenen Richtungen magnetisierten Partikeln in der Magnetschicht der Diskette. Man kann sich das als eine Aneinanderreihung von winzig kleinen "Stabmagneten" vorstellen, die kreisförmig in einer Spur auf der Diskette Pol an Pol liegen.

Eine solche Spur dreht nun mit 300 Umdrehungen/Minute unter dem Schreib-/Lesekopf des Disklaufwerks. Die unter dem Kopf vorbeirotierenden "Stabmagnete" bewirken, abhängig von ihrer Magnetisierungsrichtung, in dem Kopf Magnetfeldschwankungen, welche zu Stromschwankungen führen, die dann von der Lese-Elektronik ausgewertet werden können. Die Lese-Elektronik im Laufwerk wandelt diese Stromschwankungen in einen seriellen Bitstrom um, der zum ST gesendet wird. So werden Informationen von Diskette gelesen. Beim Beschreiben von Disketten geht man den umgekehrten Weg:

Ein serieller Bitstrom vom ST wird durch die Schreib-Elektronik im Laufwerk in gezielte Stromänderungen für den Schreib-/Lesekopf umgewandelt.

Die durch die Stromänderungen hervorgerufenen Magnetfeldschwankungen magnetisieren die Magnetschicht (die kleinen Stabmagnete) auf der Disk in die gewünschte Richtung und legen so die Informationen dauerhaft dort ab. Dauerhaft aber nur so lange, bis man die Diskette mit dem gerade fertigprogrammierten Superutility nachts um zwei Uhr erschöpft auf die

gestern neu erworbenen Autolautsprecher mit den neuen extra starken Permanentmagneten legt, die man am Wochenende unbedingt noch einbauen will.

Wahrscheinlich werden die kleinen Stabmagnete auf der Disk dann wohlgeordnet in einer Richtung angeordnet sein. Nämlich in jene, in die sie durch das äußere Magnetfeld der Lautsprechermagnete gequetscht wurden.

Kurzum, den gleichen Effekt hätte man auch durch Formatieren der Disk erreicht, die Daten (und damit das Superutility) sind futsch! Also Vorsicht vor fremden Magnetfeldern!

Präzision ist gefragt

Wer schon mal den Metallschuber der 3,5-Zoll-Diskette vorsichtig beiseite geschoben hat (Achtung bei umherfliegenden Kanarienvögeln und Wellensittichen, sie lassen meist in dem Moment etwas hinter sich, in dem man es nicht erwartet, und treffen dann auch noch genau!), wird den ca. 10 x 24 mm^2 großen Ausschnitt in der Diskettenhülle gesehen haben.

In diesem kleinen Ausschnitt bewegt sich der Schreib-/Lesekopf des Laufwerks vom Außenrand der Disk zur Mitte und wieder zurück, um die richtige Spur mit den Daten zu finden!

Wie schon kurz erwähnt, sind auf einer solchen Disk 80 Spuren vorhanden, die der Kopf "anfahren" können muß (auch wenn man mit manchen Laufwerken mehr als 80 Spuren ansteuern kann; Atari garantiert für seine Drives nur maximal 80 Spuren!). Diese 80 Spuren verteilen sich auf eine Spanne von ca. 15 mm. Eine Spur ist also nur ca. 0,19 mm breit! Man sieht daran, mit welch einer Präzision die Mechanik einer solchen Diskstation arbeiten muß.

Der Schreib-/Lesekopf wird auf einem sogenannten Kopfschlitten geführt. Die Positionierung des Schlittens erfolgt durch einen Schrittmotor, welcher den Kopfschlitten Schritt für Schritt hin und her bewegen kann. Der Wechsel von einer Spur zur benachbarten Spur geschieht bei den 3,5-Zoll-Laufwerken üblicherweise in nur 3 Millisekunden!

Die Spuren auf der Diskette werden von außen nach innen durchnumeriert. Die äußere Spur hat die Nummer 0, die innerste Spur trägt die Nummer 79. Jede Spur enthält ca. 6250 Bytes an Informationen.

Und wo fängt man an? – Der Indeximpuls

Für die Anfangsmarkierung einer Spur (wo fängt ein Kreis an?) sorgt der sogenannte Indeximpuls. Er wird einmal pro Umdrehung an immer der gleichen Stelle der Spur vom

Laufwerk erzeugt. Das geschieht durch Abtastung mittels eines Sensors am Antriebselement des Disklaufwerks.

Die neun Sektoren/Spur (9 x 512 = 4608 Bytes) werden von 1..9 durchnumeriert und sind in den möglichen 6250 Bytes eines Tracks eingebettet. "Umrahmt" werden die neun jeweils 512 Bytes langen Sektoren einer Spur von Synchronisations- und Informationsbytes für Spurnummer, Sektornummer, Prüfsummen usw.

Struktur ist wichtig! – Das Track-Layout der ST-Floppies

Die auf Disk abzulegenden Daten werden in Blöcken zu 512 Bytes (Sektoren) angeordnet. Damit der FDC (das ist der Floppy-Disk-Controller; er ist der Dolmetscher zwischen CPU und Laufwerken) jedoch weiß, über welcher Stelle auf der Diskette sich der Kopf gerade befindet, ist es erforderlich, eine bestimmte Struktur auf die Diskette aufzubringen. Es sind Markierungen (Nachgucken lohnt auch mit Lupe nicht, markiert wird magnetisch!) erforderlich, aus denen zu erkennen ist, wo eine Spur beginnt, über welcher Spur (Track) sich der Kopf gerade befindet, welcher Sektor gerade vorbeirotiert, wo die Daten im Sektor anfangen bzw. aufhören usw.

Das beim ATARI ST verwendete Tracklayout ist in Abbildung 8.1 zur besseren Übersicht schematisch dargestellt.

Ein Track beginnt mit 60 Bytes $4E, welche dem Datenseparator (der sitzt im FDC und bröselt den vom Laufwerk eintreffenden Bitstrom in Daten- und Taktinformationen auf) im FDC dazu dienen, sich auf den einlaufenden Bitstrom vom Laufwerk zu synchronisieren. Das mit dem Synchronisieren ist eine aufwendige Sache, denn der Bitstrom kommt natürlich mit Geschwindigkeitsschwankungen daher. Die Disk rotiert ja auch nicht immer mit konstanter Geschwindigkeit!)

Jeder Sektorbeginn wird durch 12 Bytes $00, gefolgt von 3 Bytes $A1 (mit fehlenden Taktimpulsen, deshalb für den FDC besonders auffällig!), angekündigt. Es folgt ein Byte mit $FE (ID-Address Mark), welches dem FDC den Beginn eines Sektor-Headers signalisiert.

Dieser Sektor-Header enthält je ein Byte für die Spurnummer ($00...$4F), Seitennummer ($00 oder $01), Sektornummer ($01...$09) und ein Byte für die Sektorgröße ($02 weist auf 512 Bytes/Sektor hin). Danach folgen zwei Bytes mit einer Prüfsumme über die vorangegangenen 4 Bytes des Sektor-Headers.

Anschließend wird der Datenseparator des FDC wieder mit 22 Bytes $4E und 12 mal $00 zur Synchronisation gefüttert.

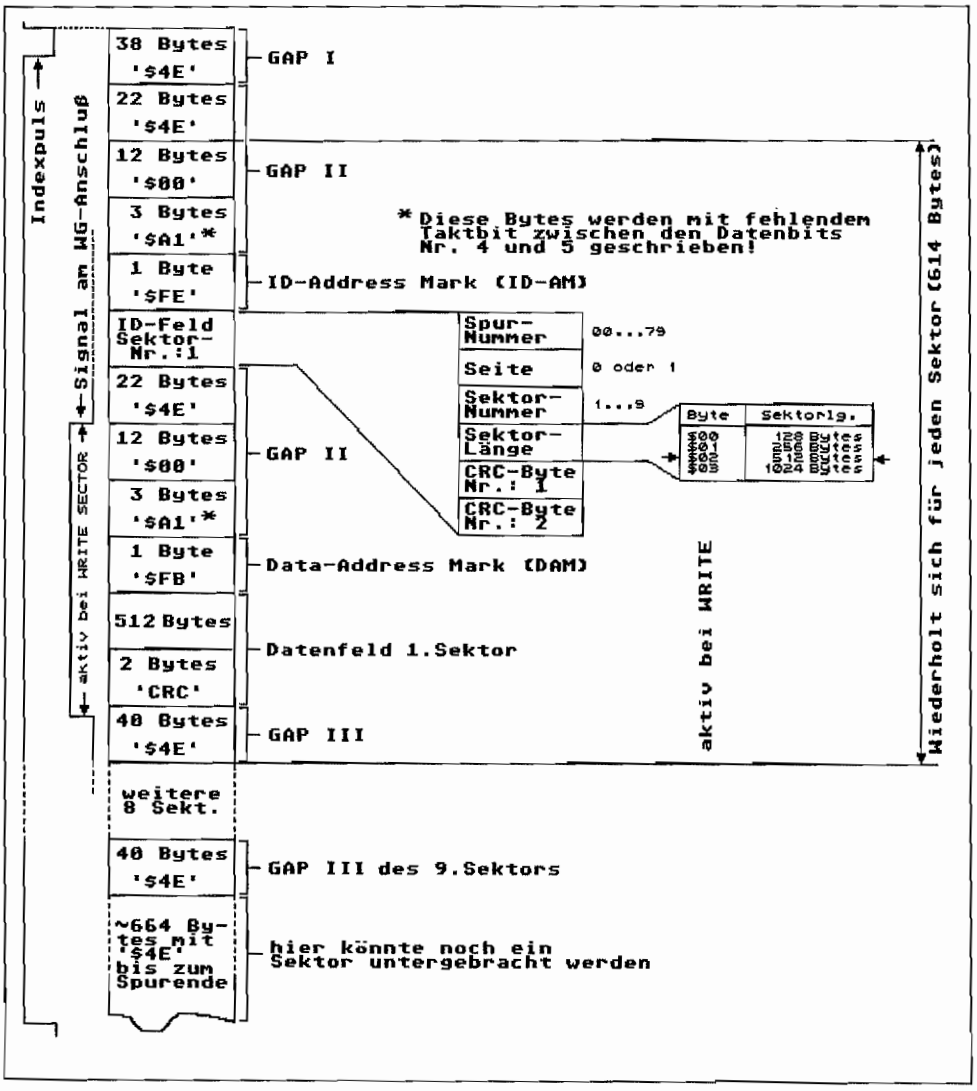


Abb. 8.1: Das Track-Layout beim ST

Drei Bytes $A1 (mit fehlenden Taktimpulsen), gefolgt von einem $FB (die "Data-address-mark", kurz: DAM), leiten dann die eigentlichen 512 Datenbytes des Sektors ein. Diesen folgt natürlich wieder eine Prüfsumme in Form zweier CRC-Bytes (CRC = **C**yclic **R**edundancy

Checking, zu deutsch etwa: zyklische Blockprüfung. Ein Datenblock wird einer laufenden Prüfzeichenbildung unterzogen. Die gewonnene Prüfinformation wird vom FDC als 2 x 8 Bit-Wert abgelegt.).

Bis zu Beginn des nächsten Sektors folgen dann 40 Bytes mit $4E (damit der FDC nicht aus dem Tritt kommt). Dieses Raster wird bei der Formatierung einer Diskette aufgebracht. Hieran läßt sich erkennen, daß so ein FDC eine Menge Dekodierarbeit zu leisten hat, bevor dann endlich die "reinen" Datenbytes zur Verfügung stehen.

Ist die Diskette jedoch erstmal auf diese Weise formatiert, erfolgen alle weiteren Zugriffe nur noch im Sektorformat. Die Markierungen und Lückenbytes auf der Disk werden ebenso wie der Sektor-Header nun nicht mehr angetastet.

Das Datenfeld eines Sektors wird vielfach wieder geändert, wobei eine Modifizierung immer ein komplettes Neubeschreiben des entsprechenden Sektors mit den geänderten Daten erfordert. Die CRC-Bytes nach jedem Datenfeld werden natürlich bei Veränderungen von Datenbytes im Sektor ebenfalls jedesmal neu berechnet und geschrieben.

Alles unter Kontrolle!? – Was der Floppy-Controller können muß

Für den Datenaustausch von Daten zwischen Computer und Diskstation sind eine Vielzahl von Funktionen auszuführen und Kontrollsignale vom Laufwerk zu überwachen.

Um die CPU von dieser Arbeit freizuhalten, werden die Floppy-Disk-Controller (FDC) eingesetzt. Diese hochintegrierten Bausteine sind speziell für die Ansteuerung von Disklaufwerken entwickelte Chips. Sie sind selbst Mikrocomputer, die jedoch nur Disklaufwerke steuern können. Diese Aufgabe beherrschen sie dafür aber meisterhaft. Um zu sehen, was ein solcher Floppy-Disk-Controller alles können muß, werfen wir mal einen Blick auf die Anschlüsse, die ein Disklaufwerk so aufweist.

Zum Glück (oder sollten da die Hersteller etwa voneinander abgekupfert haben?) hat sich bei den Anschlüssen für die Disklaufwerke bei allen Herstellern so eine Art Anschluß-Norm durchgesetzt. Außer den normalen Anschlüssen für die Stromversorgung des Laufwerks (+12V, +5V und Masse; es gibt auch bereits Laufwerke, die mit nur einer Versorgungsspannung von 5 Volt arbeiten), gibt es für die Daten- und Steuerleitungen den sogenannten Shugart-Bus.

Hierbei handelt es sich um eine 34pol. Flachbandkabel-Verbindung über Pfostenfeld-Steckverbinder (dies gilt für 3,5 Zoll-Laufwerke, bei 5,25-Zoll-Laufwerken findet man eine Steck-

verbindung, die direkt auf die Laufwerksplatine gesteckt wird). Am 3,5-Zoll-Laufwerk befindet sich dabei die Stiftleiste, folglich sitzt am Anschlußkabel die zugehörige 34pol. Buchsenleiste. Wie Abbildung 8.2 zeigt, sind jedoch nicht alle 34 Anschlüsse mit verschiedenen Signalen belegt. Zur Unterdrückung von Störimpulsen ist fast jede zweite Leitung auf Masse gelegt.

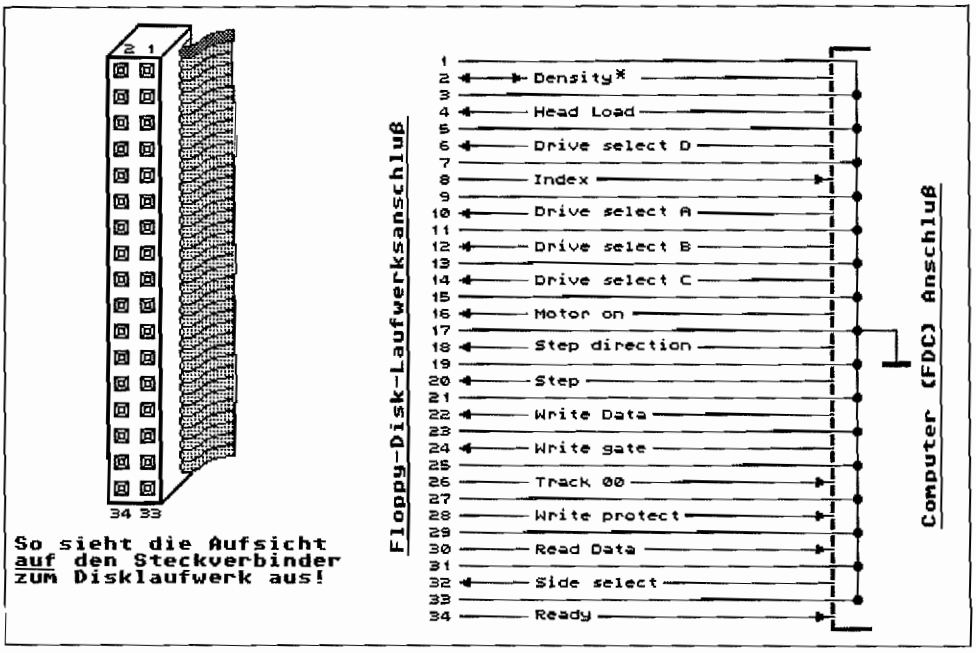


Abb. 8.2: So sieht der normale Shugart-Bus für 3,5-Zoll-Laufwerke aus

Weiterhin gibt es keine Leitungen für beide Signalrichtungen. Jedes Signal wirkt immer nur in eine Richtung, entweder vom FDC zum Laufwerk oder vom Laufwerk zum FDC. Hier nun eine kurze Beschreibung der Anschlüsse, beginnend mit den 12 Signalleitungen vom FDC zum Laufwerk.

Density* (Pin 2)

Im ST nicht benutzt (im TT und Mega STE schon)! Die modernen High-Density-Laufwerke weisen diesen Anschluß auf (mit HD-Laufwerken und speziellen HD-Disketten kann man durch Erhöhung der Taktrate zum Lesen/Schreiben 18 Sektoren/Spur unterbringen. Damit ergibt sich eine Speicherkapazität von 1,44 MByte!). Verwendet wird der Anschluß, je nach Jumpereinstellung auf dem Laufwerk, als Eingang oder Ausgang.

Als *Ausgang*: Das Laufwerk signalisiert dabei über den Anschluß dem Computer, welche Art von Disk eingelegt ist. HD-Disks werden dabei durch eine zusätzliche Öffnung in der Diskhülle erkannt! Welcher Logikpegel für die Signalisierung von HD-Betrieb verwendet wird, läßt sich ebenfalls über Jumper am Laufwerk einstellen.

Als *Eingang*: In dieser Betriebsart "sagt" der Computer, welche Betriebsart das Laufwerk einzustellen hat. Bei welchem Logikpegel HD-Betrieb "gewünscht" wird, ist wieder über Jumper konfigurierbar!

Head load (Pin 4)

Im ST nicht benutzt! Ein Low signalisiert der Laufwerkselektronik "Kopf ab", d. h., der Schreib-/Lesekopf soll auf die Disk abgesenkt werden. Bei 5,25-Zoll-Drives wird dieser Anschluß noch manchmal benutzt, während er bei 3,5-Zoll-Laufwerken seltener anzutreffen ist. Hat man ein Laufwerk mit "Head load" anzuschließen, so kann man den Anschluß mit dem "Motor on"-Anschluß verbinden! So wird "Head load" aktiviert, sobald der Motor gestartet wird. Beim Ansprechen von Drives mit "Head load" sollten FDC-Kommandos mit gesetztem E-Bit gegeben werden, um dem Laufwerk Zeit zu geben, den Kopf abzusenken.

Drive sel. A..D (Pin 10,12,14,6)

Über diese Signalleitungen bekommt die Laufwerkselektronik mitgeteilt, welches Laufwerk gerade angesteuert wird. Das Signal ist Low-aktiv, d. h., ein Low-Pegel auf der entsprechenden Leitung wählt das gewünschte Laufwerk an. Atari benutzt nur Drive select A und B!

Side select (Pin 32)

Bei High-Pegel wird auf Seite 0 (Vorderseite) der Disk gearbeitet. Sobald hier Low anliegt, bekommt der Schreib-/Lesekopf auf der Seite 1 (Rückseite) Arbeit.

Motor on (Pin 16)

Ein Low-Pegel startet den Motor (immer! Egal, ob das Laufwerk selektiert ist oder nicht!)

Write gate (Pin 24)

Signalisierung an die Laufwerkselektronik:

High: Daten werden gelesen.
Low : Daten werden geschrieben.

Write Data (Pin 22) — Der FDC sendet über diese Leitung die zu speichernden Daten in einem seriellen Bitstrom. Die Laufwerkselektronik setzt diesen Datenstrom in entsprechende Magnetfeldschwankungen im Schreib-/Lesekopf um.

Step direction (Pin 18) — Die Richtung, in welcher der Kopf bewegt werden soll, wird hiermit festgelegt.

High: Schrittbewegung nach außen (niedrigere Spurnummer)
Low : Der Kopf soll nach innen (höhere Spurnummer)

Die eigentliche Schrittbewegung wird jedoch erst durch das Signal "Step" ausgelöst!

Step (Pin 20) — Jeder Low-Impuls bewegt den Kopf einen Schritt in die durch "Step direction" vorgewählte Richtung.

Nun zu den Signalen, die vom Laufwerk an den FDC gerichtet sind:

Write protect (Pin 28) — Ein High signalisiert dem FDC, daß die Diskette beschrieben werden kann. Bei Low ist das Schreiben auf Disk gesperrt. Für die Signalisierung wird das Write protect-Fenster in der Diskette durch eine Lichtschranke abgefragt.

Track 00 (Pin 26) — Wenn der Kopf auf Spur 00 positioniert ist, geht dieses Signal auf Low-Pegel. Sonst liegt hier High.

Index (Pin 8) — Bei jeder Diskettenumdrehung wird ein kurzer Low-Impuls erzeugt. Dies geschieht immer an der gleichen Stelle einer Umdrehung und dient zur Markierung des Spuranfangs.

Read Data (Pin 30) — Beim Lesen von Daten ("Write gate" auf High) liefert die Laufwerkselektronik an diesem Anschluß einen Bitstrom mit Daten von der Diskette. Für die Entwirrung dieses Datenstroms, der Daten- und Taktinformationen enthält, und die Umsetzung in Datenbytes ist der FDC zuständig. Der Bitstrom ist durch Drehzahlschwankungen des Diskantriebs nicht immer gleichmäßig, sondern unterliegt Schwankungen, die der FDC bei der Decodierung berücksichtigen muß.

Ready (Pin 34) — Hierüber wird dem FDC mit einem Low signalisiert, daß das angesprochene Laufwerk bereit zur Arbeit ist. Das ist dann der

Fall, wenn eine Disk eingelegt ist. Deshalb wird das Ready-Signal auch verschiedentlich für die Diskwechsel-Erkennung benutzt. Beim ST wird der Anschluß jedoch nicht verwendet.

Dürfen's ein paar Leitungen weniger sein?

Die Konstrukteure des ST waren wohl der Meinung, daß 34 Leitungen viel zuviel sind. Außerdem sorgt ein "eigenes" Steckerformat dafür, daß bevorzugt die von Atari produzierten Diskstationen verkauft werden. Wohl deshalb werden die Diskstationen an den ST über ein 14pol. Anschlußkabel mit einem 14pol. Stecker in der Form der heute üblichen DIN-Stecker angeschlossen.

Beim Öffnen eines originalen Atari-Laufwerks stellt man dann jedoch fest, das im Gehäuse der Diskstation wieder eine Umsetzung vom Atari-Steckerformat auf den gebräuchlichen 34poligen Steckanschluß erfolgt. Die Abbildung 8.3 zeigt den Anschluß von zwei Laufwerken mit der Umsetzung vom 14poligen Atari-Disksteckerformat auf den 34poligen Shugart-Bus im Detail.

Die ST-Computer sind vom Betriebssystem her darauf ausgelegt, zwei Diskettenlaufwerke zu steuern.

Die erste Diskstation (im Desktop das Laufwerk A) wird direkt am ST angeschlossen bzw. ist bereits eingebaut. Die zweite Station (Laufwerk B) wird an der "Out"-Buchse der ersten Diskstation, ebenfalls mit 14pol. Steckverbinder, angeschlossen. Beim ATARI 1040 STF, STE, TT und MEGA ST(E) ist für das zweite Laufwerk (externes Laufwerk) an der Rückseite des ST eine Buchse vorgesehen.

Beim Betrieb von mehr als einem Laufwerk an einem FDC wird jedem Laufwerk eine eigene Identifikation zugeordnet. Dies geschieht auf der Laufwerksplatine durch entsprechendes Einstellen von DIP-Schaltern oder das Versetzen von Steckbrücken (Jumpern). Damit ist gewährleistet, daß sich Laufwerk A auch nur dann angesprochen fühlt, wenn es durch Low am "Drive select A"-Anschluß angesteuert wird. Laufwerk B darf nur reagieren, wenn es am "Drive select B"-Anschluß ein Low erhält, usw.

Atari führt jedoch die "Drive select B"-Leitung vom Computer über die "In"-Buchse (Anschluß 6) auf den "Drive select A"-Anschluß (Anschluß 5) der "Out"-Buchse der Diskstation! Eine dort angeschlossene zweite Diskstation fühlt sich somit dann "auserwählt", wenn "Drive select B" vom ST auf Low gesetzt wird.

Das hat den Vorteil, daß keine Einstellarbeiten an der Diskstation nötig sind, weil alle Stationen als "Drive A" programmiert werden können (sie werden von ihrer "In"-Buchse über

"Drive select A" selektiert). Welche Station nun für den ST dann "Drive A" oder "Drive B" ist, hängt somit nur von ihrer Position in der Anschlußkette zum Computer ab.

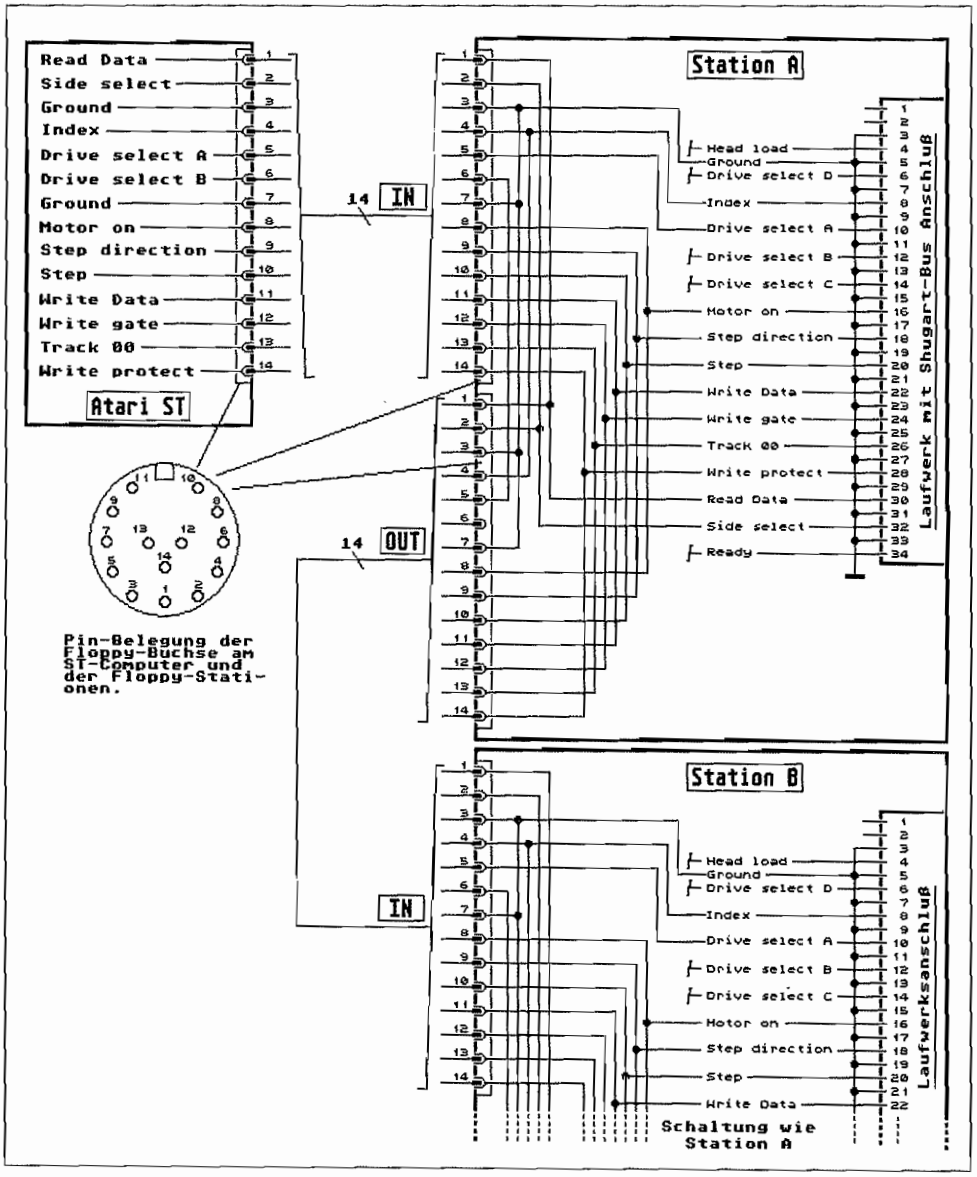


Abb. 8.3: So sind bei den ATARI ST die Diskstationen verschaltet

Dieses Prinzip hat jedoch auch den Nachteil, daß nur max. zwei Floppy-Stationen am ST betrieben werden können.

Bei den Ataris mit eingebautem Laufwerk ist das interne Laufwerk immer als Drive A geschaltet. An der Buchse für das externe Laufwerk ist der Anschluß für "Drive select B" (Anschluß 6) über einen Pull-Up-Widerstand fest an +5V gelegt. Soll das Laufwerk B (externes Laufwerk) selektiert werden, gibt der Computer an "Drive select A" (Anschluß 5) der Floppy-Buchse ein Low aus! Also muß das externe Laufwerk in diesem Fall auch hier wieder als "Drive A" eingestellt sein.

Wie man trotzdem bei Ataris mit eingebautem Laufwerk das interne Laufwerk zu Drive B "degradiert" und das externe Laufwerk zum Drive A "ernennen" kann, ist ausführlich in einem Artikel im "ST-MAGAZIN/68000er", Ausgabe 7/88, beschrieben.

Fremd(körper)laufwerke am ST

Mit Hilfe von zwölf Steuer- und Meldeleitungen steuert nun der ST die angeschlossenen Laufwerke. Die Hauptarbeit übernimmt dabei, wie schon vorher erwähnt, der Floppy-Disk-Controller.

Die Steuerung der "Side select"- und "Drive select A bzw. B"-Leitungen erfolgt aber nicht durch den FDC, sondern wird vom Port A des Sound-Chip (PSG) vorgenommen. Das hat bei Anschluß von Fremdlaufwerken an den ST öfter zu Schwierigkeiten geführt, weil ein Port-Ausgang des PSG bei Low-Signal nur maximal 5 mA treiben kann.

Die "Side select"- und "Drive select"-Eingänge an den normalen Laufwerken besitzen zwar in der Regel "Low-Power-Schottky"-Eingangsstufen, die mit geringen Steuerströmen (etwa 1/4 des Standard-TTL Steuerstroms) auskommen, jedoch sind diese Eingänge meist über 1kΩ-Pull-Up Widerstände an +5V "gebunden". Das schafft der Sound-Chip gerade noch so eben.

Wird jedoch ein zweites Laufwerk angeschlossen, so sollte man bei einem der Laufwerke die Pull-Up-Widerstände für "Side select" und "Drive select" entfernen (sinnvollerweise beim ersten Drive in der Kette der Laufwerke, um den Leitungsabschluß am Ende der Kette zu haben)! Die Port-Ausgänge des PSG wären sonst überlastet; ein sicheres Steuern der Laufwerke ist nicht mehr gegeben.

Man kann auch zur Erhöhung der Ausgangsleistung und zur Entlastung des Sound-Chips Treiberstufen (mit z. B. dem TTL-IC "7407") für die "Side select"- und "Drive select A bzw. B"-Steuerleitungen zwischenschalten. Dieses zusätzliche 14polige IC läßt sich noch relativ problemlos in einer Diskstation unterbringen und aus der Stromversorgung des Laufwerks speisen. Siehe hierzu den Schaltungsvorschlag in Abbildung 8.4.

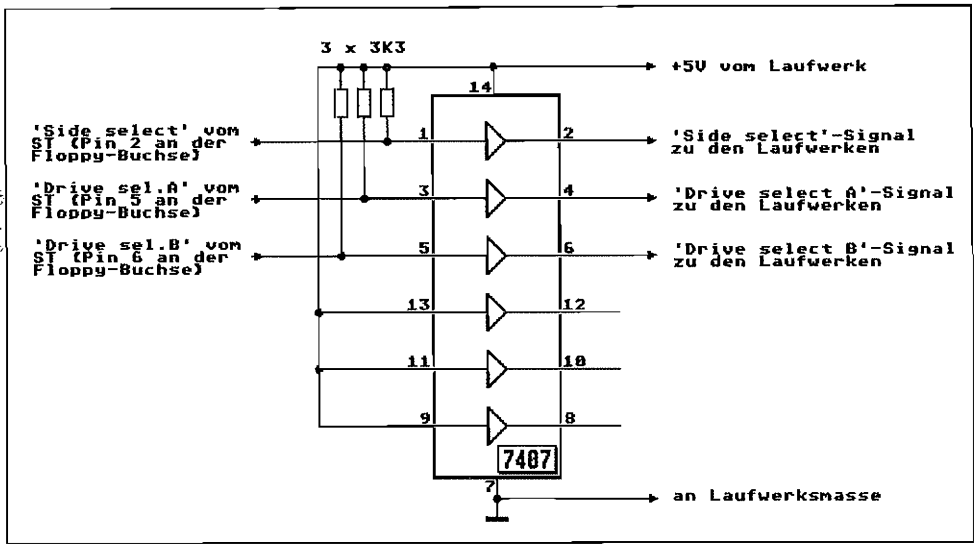


Abb. 8.4: So kann man den Sound-Chip des ST bei der Laufwerksansteuerung wesentlich entlasten

Viel Grips in kleinen Chips – Der FDC

Im ST wird als FDC ein Chip des Typs WD1772 eingesetzt. Dieser Typ ist kompatibel zu FDCs der 179X-Serie, besitzt jedoch bereits einen eingebauten Datenseparator (um die Daten aus dem Strom von Daten- und Taktbits auszufiltern) und erlaubt eine höhere Steprate für die Kopfmotorsteuerung.

Der "1772" kann Sektorgrößen von 128, 256, 512 oder 1024 Bytes/Sektor bearbeiten und das sowohl in einfacher als auch doppelter Dichte. Atari verwendet ein 512er Sektorenformat in doppelter Dichte. Das verwendete Format ist physikalisch kompatibel zum IBM-Diskformat. Schließt man ein 5,25 Zoll-Laufwerk an den ST an, so kann man im Prinzip und in der Praxis auf einem IBM- oder -kompatiblen PC formatierte und beschriebene Disks bearbeiten!

Der FDC liefert an seinen Chip-Ausgängen für "Motor on", "Step", "Step direction" usw. die Steuerpegel genau entgegengesetzt, wie sie von den Diskstationen gebraucht werden. So geht z. B. bei der Funktion "Motor starten" der entsprechende Anschluß (MO = Motor on, Pin 20) des "1772" auf High-Pegel, während das Laufwerk den Motor bei Low-Pegel am "Motor on"-Anschluß startet. Deshalb sind im ST invertierende Treiberstufen (mit dem TTL-IC "7406") zwischengeschaltet, die Ausgangsstufen mit offenen Kollektoren besitzen und so für die Pegelanpassung und den ebenfalls erforderlichen "Dampf" auf den Steuerleitungen zu den Laufwerken sorgen.

Der FDC im ST erhält Hilfe – von der DMA-Einheit

Der FDC ist nicht direkt mit dem Daten- und Adreßbus des ST verbunden. Jeglicher Datenaustausch läuft nur "unter Aufsicht" der DMA-Einheit (siehe Abbildung 8.5).

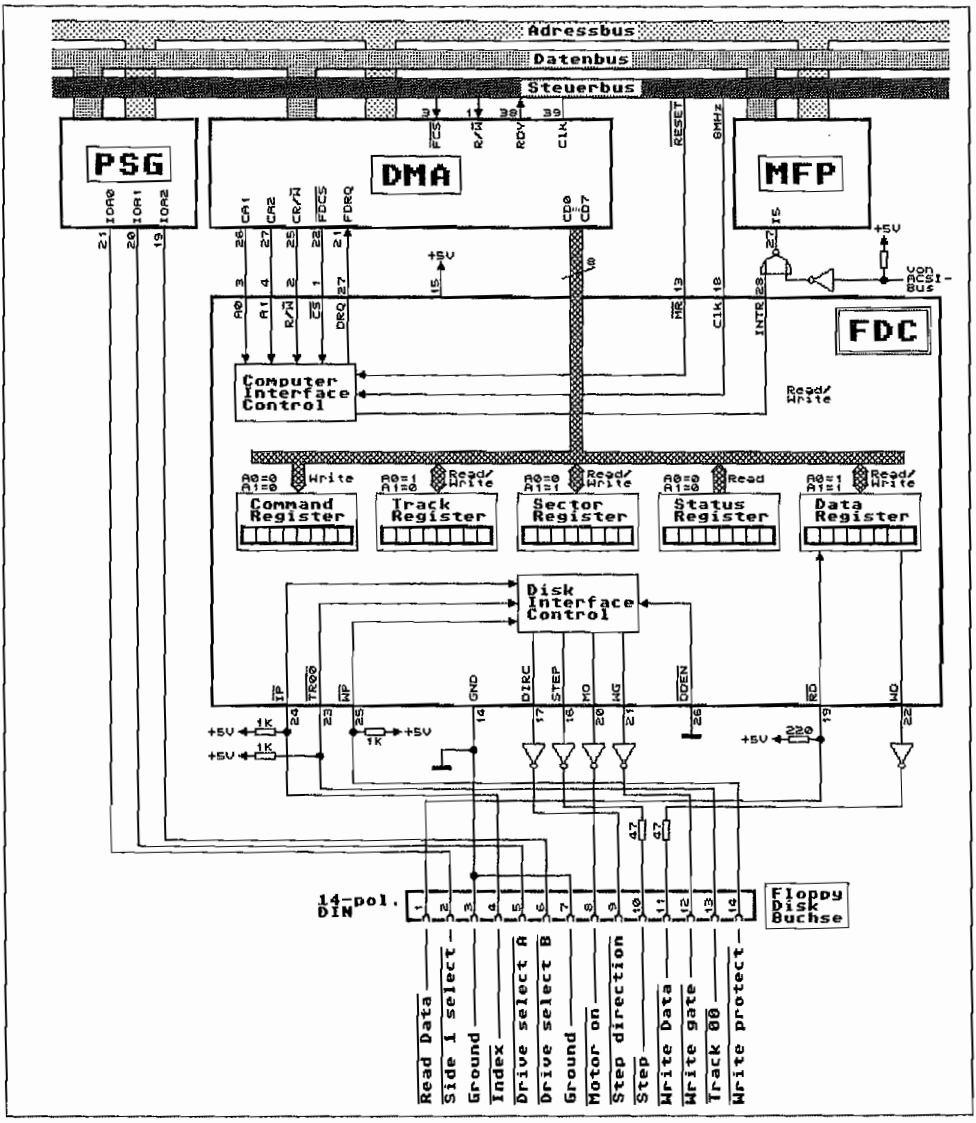


Abb. 8.5: Einbindung des FDC in die Hardware des ST

Über die A0-, A1- und CS-Leitung selektiert der DMA-Baustein die Register des FDC. Der logische Zustand der R/W-Leitung informiert den FDC darüber, ob ein Schreib- oder Lesezugriff auf eines seiner Register erfolgt.

Per (High-)Signalisierung auf der DRQ-Leitung bekommt die DMA-Einheit mitgeteilt, wann ein "Data request" (Datenanforderung) des FDC vorliegt. Hierüber teilt der FDC mit, daß er Daten von der DMA-Einheit erwartet bzw. Daten bereitgestellt hat (beim Lesevorgang), die von der DMA-Einheit abgeholt werden müssen. Das Ende einer Operation meldet der FDC mit einem High auf der INTRQ-Leitung an den Port I5 des MFP (über das NOR-Gatter wird dieses High dann zu einem Low am MFP-Port I5).

Der Anschluß DDEN (Double Density ENable = Doppelte Schreibdichte anwählen, Pin 26 des FDC) ist im ST fest auf Low gebunden. Damit kann der FDC nur mit dem MFM-Aufzeichnungsverfahren (MFM = Modified Frequency Modulation) betrieben werden. Aber es wird ja in modernen Mikrocomputersystemen (und dazu kann man die ST-Serie ja wohl zählen) ausschließlich in doppelter Dichte gearbeitet.

Inzwischen nutzt man ein nicht dokumentiertes Feature des FDC's vom Typ WD1772 aus. So vertragen einige Chips dieses Typs eine Verdoppelung der Taktrate an ihrem Clk-Eingang. Wenn statt mit dem Standard-8-MHz-Takt mit 16 MHz gearbeitet wird, geht alles doppelt so schnell, und der FDC hat auf einmal High-Density. Auf diese Weise verfährt Atari beim TT und Mega STE, um die internen Drives mit High-Density fahren zu können. Nur verwendet man nicht den Original WD 1772-Chip dafür, sondern einen eigenen Chip, der auch laut Spezifikation die Verdoppelung der Taktrate mitmacht!

Die Register des FDC

Wie das Übersichtsbild in Abbildung 8.5 ebenfalls zeigt, besitzt der FDC fünf Register zu je 8 Bit Breite. Je nach auszuführender Operation ist es zunächst erforderlich, einige dieser Register mit den erforderlichen Voreinstellungen zu versehen, bevor in das Kommandoregister das Kommandobyte eingeschrieben wird.

Das *Track-Register* (A0=1, A1=0, R/W) enthält die Nummer der Spur, über der sich der Kopf gegenwärtig befindet. Eine Schreib-/Lese- oder Verify-Operation kann nur erfolgen, wenn der Inhalt des Track-Registers mit der im ID-Feld auf der Disk enthaltenen Spurnummer übereinstimmt. Das Register kann gelesen und beschrieben werden. Während der FDC eine Operation ausführt, sollte kein Zugriff auf das Register durchgeführt werden.

Der FDC besitzt nur ein Trackregister, der ST aber bis zu zwei Laufwerke. Dummerweise befinden sich die Köpfe der beiden Laufwerke selten über der gleichen Spur. Deshalb muß

bei einem Laufwerkswechsel die aktuelle Spurnummer des Laufwerks, mit dem momentan nicht gearbeitet wird, zwischengespeichert werden.

Dies geschieht beim ST im sogenannten Drive Status Block (DSB). Und da maximal zwei Disklaufwerke angeschlossen sein können, gibt es logischerweise zwei DSBs! Das erste Datenwort des jeweiligen DSB enthält die Spurnummer, über der sich der Kopf gerade befinden sollte. Im zweiten Wort findet sich die Steprate, mit der das Laufwerk seinen Schreib-/Lesekopf über die Disk hetzt. Je nach TOS-Version ergeben sich für die Anfangsadresse der beiden DSB andere Werte.

Hier die Adressen für einige TOS-Versionen:

	RAM-TOS 11/1985	**RAM-/ROM-TOS vom 6.2.86**	**Blitter-TOS vom 22.4.87**	**Rainbow-TOS 1.04**
DSB Drive A				
Acurtrack	$6C8	$A06	$A4C	$A4C
Aseekrt	$6CA	$A08	$A4E	$A4E
DSB Drive B				
Bcurtrack	$6CC	$A0A	$A50	$A50
Bseekrt	$6CE	$A0C	$A52	$A52

Man sollte jedoch diese Variablen nicht unbedingt verwenden! Da aber zumindest die Einstellung der Seekrate bei Anschluß von 5,25-Zoll-Laufwerken sehr wichtig ist, hat ATARI in den "Rainbow TOS Release Notes" dann für das ROM-TOS 1.00 und Version 1.02 ("BLITTER-TOS") die Lage der Adressen für "Aseekrt" und "Bseekrt" dokumentiert. Auf andere Art und Weise ist keine getrennte Einstellung der Steprate für die Laufwerke möglich. In TOS-Versionen 1.04 und höher ist die Steprate über den XBIOS-Aufruf #41, "Floprate()", konfigurierbar.

Im *Sector-Register* (A0=0, A1=1, R/W) wird dem FDC die Nummer des anzusteuernden Sektors mitgeteilt. Ein Schreib- oder Lesezugriff erfolgt nur bei Übereinstimmung mit der Sektornummer im ID-Feld des Sektors auf der Disk. Das Sector-Register kann beschrieben und gelesen werden. Auch für dieses Register gilt: Nicht ansprechen, solange eine Operation nicht beendet ist.

Das *Data-Register* (A0=1, A1=1, R/W) wird benutzt, um dem FDC die zu schreibenden Daten Byte für Byte zu übermitteln. Während einer Leseoperation stehen hier die gelesenen Bytes für den Prozessor (beim ST macht das die DMA-Einheit) zum Abholen bereit. Bei einem

"SEEK"-Kommando (Gezieltes Ansteuern einer Spur) wird dem FDC im Data-Register die gewünschte Spur mitgeteilt, die "angefahren" werden soll!

Durch Einschreiben eines Bytes in das *Command-Register* (A0=0, A1=0, Write only) wird der FDC aufgefordert, eine bestimmte Operation auszuführen. Das Register kann nur beschrieben werden. Während einer laufenden Operation darf das Register nicht neu beschrieben werden, es sei denn, die laufende Aktion soll unterbrochen werden.

Aus dem *Status-Register* (A0=0, A1=0, Read only) kann eine Zustandsinformation über das durchgeführte Kommando abgerufen werden. Die Bedeutung der einzelnen Statusbits hängt von dem ausgeführten Kommando ab. Da es sich um ein Zustandsregister handelt, ist nur ein Auslesen möglich und sinnvoll. Es darf auch während einer laufenden Operation ausgelesen werden!

Die Kommandos des ST-Floppy-Controllers

Der FDC kennt insgesamt elf verschiedene Kommandos, die in vier verschiedene Kategorien (Typen) eingeteilt sind. Sie sind in der Abbildung 8.7 aufgeführt.

Die Kommandos werden ins Command-Register geschrieben und unmittelbar nach Beendigung des Einschreibvorgangs ausgeführt. Das Laden des Command-Registers sollte nur bei gelöschtem BUSY-Bit des FDC-Status-Registers durchgeführt werden, da sonst die laufende Operation unterbrochen wird!

Eine Ausnahme bildet hierbei das "Force Interrupt"-Kommando, das jederzeit gesendet werden kann, aber – wie der Name eben schon ausdrückt – sowieso nur zur Unterbrechung von einer gerade laufenden Operation benutzt wird.

Solange ein Kommando bearbeitet wird, ist das BUSY-Bit im Status-Register gesetzt. Bei Operationsende wird es gelöscht und ein Interrupt ausgelöst, der über die IRQ-Leitung zum Port I5 des MFP geleitet wird.

Der Inhalt des Status-Registers gibt dann Aufschluß darüber, ob die Operation korrekt durchgeführt wurde bzw. welche Fehler aufgetreten sind.

Es läßt sich auf zwei Arten feststellen, ob der FDC ein Kommando abgeschlossen hat.

- BUSY-Bit (Bit 0) im Status-Register so lange testen, bis es vom FDC zurückgesetzt wird. Bevor jedoch nach dem Absenden des Kommandos an den FDC (Einschreiben in das Command-Register) zum ersten Mal das Status-Register abgefragt wird, sollte man dem

FDC eine Pause von mindestens 32 µs gönnen. Eher sind die Bits im Status-Register nach Start eines Kommandos nämlich noch nicht gültig!

– Warten!!! Nicht ewig, sondern bis der FDC einen Interrupt auslöst. Dazu wird der Anschluß INTR (Pin 28 des FDC) auf High gesetzt. Als Folge davon wird dann im ST beim Port I5 des MFP ein Low auftreten.

  Also nach Start eines FDC-Kommandos immer mal bei Port I5 des MFP nachschauen, ob schon ein Low anliegt, der FDC ist dann nämlich fertig. Das ist auch die vom Betriebssystem des ST praktizierte Methode.

Bit-Nr.	Bedeutung
7	Zustand des 'Motor on'-Anschlusses ('1' = Motor on)
6	Bei READ SECTOR und READ TRACK ohne Funktion. Bei jedem WRITE-Befehl wird hiermit ein Schreibschutz angezeigt.
5	Typ I-Kommandos: Gesetzt sobald Motor-'Spin up' erfolgt ist. (6 Indexpulse nach 'Motor on') Typ II..III-Kommandos: '1'= Sektor mit 'Deleted'-Data mark. '0'= Sektor mit 'Valid'-Data mark.
4	Gesetztes Bit zeigt an, daß gewünschter Sektor, Spur oder Seite nicht gefunden wurde. (Record not found)
3	Gesetzt wenn CRC-Error festgestellt wurde.
2	Typ I-Kommandos: Zustand des 'Track 00'-Anschluß. ('1' = Kopf über Spur 00) Typ II..III-Kommandos: Wird '1' wenn die CPU (DMA) beim Datentransfer zu langsam reagiert hat. ('Lost Data'-Bit')
1	Typ I-Kommandos: Zustand des Index-Anschluß Typ II..III-Kommandos: Zustand des DRQ-Anschluß. ('1' = Data Register muß 'bedient' werden.)
0	'BUSY'-Bit. Ein Kommando ist in Bearbeitung.

Abb. 8.6: Die Bedeutung der Bits im FDC-Status-Register

Kommandos des Typs I

Hierunter sind alle Kommandos zusammengefaßt, die zur Kopfpositionierung benutzt werden, wie RESTORE, SEEK, STEP, STEP-IN und STEP-OUT.

Hier wird gesteppt!

Alle fünf Kommandobytes des Typs I beinhalten in ihrem Befehlsbyte ein Feld von 2 Bits, die zur Einstellung der Spurwechselzeit (Steprate) dienen. Folgende Stepraten sind möglich:

r1	r0	Steprate	
0	0	6 ms	(bei 40-Spur-Drives (5,25") üblich)
0	1	12 ms	
1	0	2 ms	(bei manchen 3,5"-Floppies möglich)
1	1	3 ms	(bei 80-Spur-Drives (3,5" und 5,25") gängig)

Mit jedem Step-Impuls wird der Kopf um eine Spur weiterbewegt. Die Richtung wird dabei durch den Zustand am "Step direction"-Anschluß signalisiert. Der Pegel am "Step direction"-Anschluß bleibt so lange gleich, bis der Kopf durch ein entsprechendes Kommando in die entgegengesetzte Richtung bewegt werden soll!

Nach Ausführung des letzten Steps in die gewünschte Richtung wird dem Kopf eine "Beruhigungspause" von 30 ms gegönnt, um evtl. Vibrationen in der Kopfpositioniermechanik abklingen zu lassen. Dies geschieht jedoch nur bei gesetztem "V-Bit" (Verify-Bit) oder gesetztem "E-Bit" der Kommandos des Typs II + III.

Überprüfung ist wichtig

Ist also das "V-Bit" gesetzt, so erfolgt nach der Kopfberuhigungszeit eine Überprüfung, ob der Kopf sich denn nun wirklich über der gewünschten Spur befindet.

Dazu wird die Tracknummer aus dem ersten vorbeikommenden ID-Feld gelesen und mit der Spurnummer im Track-Register verglichen. Stimmen diese überein und ist die CRC-Prüfsumme des gelesenen ID-Feldes korrekt, ist die Verify-Prozedur beendet. Der FDC erzeugt dann einen Interrupt als Zeichen der Beendigung des Kommandos und signalisiert im Status-Register den Erfolg der Operation.

Stimmt beim Verify zwar der Track, ergibt die CRC-Überprüfung aber einen Fehler, so gibt der FDC nicht gleich auf, sondern wiederholt die gleiche Überprüfungsprozedur mit dem nächsten ID-Feld.

Findet der FDC bei dieser Verify-Prozedur innerhalb von fünf Diskumdrehungen keine Übereinstimmung zwischen Track-Register und Spurnummer in den ID-Feldern, so wird die Operation mit einem Interrupt beendet und das "Seek Error-Bit" im Status-Register gesetzt.

Auszeit für den Laufwerksmotor

Ist bei den Kommandos des Typs I..III das "h"-Bit *gelöscht* ("Motor on"-Flag), so wird die eigentliche Operation erst gestartet, wenn der FDC der Ansicht ist, daß der Antriebsmotor des Laufwerks mit Solldrehzahl läuft!

Typ	Befehl	Kommandobyte Bit 7	6	5	4	3	2	1	0
I	RESTORE	0	0	0	0	h	V	r1	r0
I	SEEK	0	0	0	1	h	V	r1	r0
I	STEP	0	0	1	u	h	V	r1	r0
I	STEP-IN	0	1	0	u	h	V	r1	r0
I	STEP-OUT	0	1	1	u	h	V	r1	r0
II	READ SECTOR	1	0	0	m	h	E	0	0
II	WRITE SECTOR	1	0	1	m	h	E	P	a0
III	READ ADDRESS	1	1	0	0	h	E	0	0
III	READ TRACK	1	1	1	0	h	E	0	0
III	WRITE TRACK	1	1	1	1	h	E	P	0
IV	FORCE INTERRUPT	1	1	0	1	I_3	I_2	I_1	I_0

Flags

für Kommandos des Typs I

h = 'Motor on'-Flag

h = 0 : 'Spin-up' eingeschaltet
h = 1 : 'Spin-up' ausgeschaltet

u = 'Update'-Flag

u = 0 : Kein 'Update'
u = 1 : Aktualis. des Track Reg.

V = 'Verify'-Flag

V = 0 : kein Verify
V = 1 : Test auf korrekte Spur

r1,r0 = Stepimpuls-Rate

0 0 = 6 ms
0 1 = 12 ms
1 0 = 2 ms
1 1 = 3 ms

für Kommandos des Typs II + III

m = 'Mult. Sector'-Flag

m = 0 : 'Single sector'-Zugriff
m = 1 : 'Multiple sector'-Zugriff

E = 'Head settling'-Flag

E = 0 : Keine Verzögerung
E = 1 : 30ms Verzögerung

a0= 'Data addr. mark'-Bit

a0= 0 : Schreibe 'Normale' DAM
a0= 1 : Schreibe 'Deleted' DAM

P = 'Write Prekompensat.'

P = 0 : 'Prekompensation' Ein
P = 1 : 'Prekompensation' Aus

für Typ IV-Kommandos

'FORCE INTERRUPT'-Bedingungsbits

I_0 = 1 : Nicht zulässig
I_1 = 1 : Nicht zulässig
I_2 = 1 : Interrupt beim nächsten Indexpuls
I_3 = 1 : Sofortige Unterbrechung der Operation
$I_0 \ldots I_3$ = 0 : Operation ohne Interrupt abbrechen

Abb. 8.7: Die Kommandos des Floppy-Controllers WD 1772

Gönnt der FDC über den "Motor on"-Anschluß (Pin 20 des FDC) dem Laufwerksmotor gerade eine Pause, so wird ein Kommando mit *gelöschtem* "h"-Bit zuerst den Drive-Motor über die "Motor on"-Leitung starten und dann fünf Umdrehungen warten (Die Umdrehungen werden mittels der Indexpulse gezählt).

Nach diesen fünf Umdrehungen (ca. 1s bei 300 U/min) geht der FDC davon aus, daß der Laufwerksmotor seine Nenndrehzahl erreicht hat, und fährt mit der Operation fort.

Nach Beendigung der Operation wartet der FDC noch zehn Diskrotationen (wieder Indeximpulse zählen!), bis der Motor gestoppt wird. Damit wird verhindert, daß der Motor evtl. ständig läuft und der Kopf einen "Graben" in die Diskette "pflügt".

Wird ein Kommando mit gelöschtem "h"-Bit gegeben und läuft der Motor noch auf Nenndrehzahl ("Motor on"-Bit im Status Register noch gesetzt), so entfällt natürlich die Hochlaufzeit für den Laufwerksmotor von 1s. Das hört sich in der Theorie ganz simpel an, in der Praxis treten jedoch hin und wieder Probleme auf.

So funktioniert die ganze Zählerei mit den Indeximpulsen natürlich nur dann, wenn das Laufwerk auch selektiert ist. Die Diskstation ständig selektiert zu lassen ist aber auch nicht die ideale Lösung. Vor allem dann nicht, wenn mit mehr als einem Laufwerk gearbeitet wird. Dann laufen ja auf der Indeximpuls-Leitung die Indeximpulse aller angeschlossenen Laufwerke gleichzeitig beim FDC auf! Das TOS versucht das Problem folgendermaßen zu lösen:

In jedem Vertikal-Blank-Interrupt wird überprüft, ob der FDC inzwischen den Laufwerksmotor per "Motor on"-Leitung abgeschaltet hat. Ist das der Fall, wird das Laufwerk deselektiert!

Das führt jedoch zum nächsten Problem. Die VBlank-Interrupt-Routine zur Überprüfung der "Motor on"-Leitung muß ja dazu auf das Status-Register des FDC zugreifen (Bit 7 des Status-Registers zeigt ja den Zustand der "Motor on"-Leitung an) und "stolpert" einer evtl. gerade laufenden Diskoperation dazwischen.

Also hat ATARI in seinen Betriebssystemvariablen ein Flag eingerichtet ("flock" = Floppy-Disk-lock (Floppy-Verriegelung) an Adr. $43E), das immer dann gesetzt ist (auf <>0), wenn gerade eine Diskoperation stattfindet. Die Interrupt-Routine im V-Blank schaut immer zuerst bei "flock" nach, ob ein Zugriff auf den Disk-Controller zulässig ist. Bei gelöschtem Flag (flock=0) darf die V-Blank-Interrupt-Routine auf die Register des FDC zugreifen.

Wenn keine Disk eingelegt ist, weil z. B. von Harddisk gebootet wurde, kann das Deselektieren aber gar nicht klappen, weil (mangels Disk) keine Indeximpulse gezählt werden können. Damit schaltet der FDC den Laufwerksmotor aber auch nicht ab, und als Folge davon kann die VBlank-Routine auch das Laufwerk nicht deselektieren!

RESTORE – Zurück an den Anfang

Einer der wichtigsten Positionierungsbefehle ist das Kommando RESTORE. Es sorgt dafür, daß der Kopf so lange schrittweise in Richtung "Heimat" (Home-Position = Spur 00) bewegt wird, bis über die "Track 00"-Leitung (Pin 23 des FDC) eine Signalisierung (Low-Pegel) erfolgt. Das bedeutet dann nichts anderes, als daß der Kopf nun Spur 00 erreicht hat.

Ist das der Fall, so wird das Track-Register auf $00 gesetzt und ein Interrupt ausgelöst. Sollte jedoch nach 255 Step-Impulsen immer noch keine "Track 00"-Signalisierung erfolgt sein, so wird die Operation "interrupted" und im Status-Register das "Record not found"-Bit gesetzt (nur wenn mit "Verify" gearbeitet wird!).

Benutzt wird der RESTORE-Befehl immer vor einem Zugriff auf die Disk, wenn nicht genau bekannt ist, über welcher Spur sich der Kopf befindet (z. B. nach dem Einschalten des Systems). Die "V"-,"h"-,"r1"- und "r0"-Bits wirken auch hier wie bereits beschrieben.

Wer suchet, der findet – Der SEEK-Befehl

Die SEEK-Operation (Suchen) erlaubt das gezielte Anfahren einer Spur.

Der FDC erwartet dabei im Data-Register (!) die Nummer der Spur, über die der Kopf positioniert werden soll, während sich im Track-Register die Spurnummer der augenblicklichen Kopfposition befindet. Es werden dann vom FDC so lange Step-Impulse in der nötigen Richtung ausgegeben, bis der Inhalt des Track-Registers (welches jeden Step mitzählt) mit der gewünschten Spurnummer im Data-Register übereinstimmt.

Ein "Verify" über der gewünschten Spur ist natürlich auch hier möglich durch Setzen des V-Bits. Nach Abschluß der Operation wird wieder ein Interrupt erzeugt und die Status-Bits im Status-Register entsprechend gesetzt.

Mehr Speed durch weniger Kontrolle – SEEK ohne Verify

Dieses gesetzte Verify-Bit sorgt im TOS dafür, das die Diskzugriffe nicht mit der eigentlich möglichen Geschwindigkeit ablaufen. So wird nämlich bei jedem (!) Disk-Zugriff des TOS, der ein SEEK erfordert (und das sind die meisten!), eine SEEK-Routine aufgerufen, die für die Positionierung des Kopfs sorgt. Bei "SEEK mit Verify" wird über der gewünschten Spur noch zur Kontrolle ein ID-Feld eingelesen und die darin enthaltene Spurnummer mit der gewünschten Tracknummer verglichen.

Das kostet Zeit, die man sich eigentlich sparen kann, weil die Positionierung zum einen wenig fehleranfällig ist und außerdem beim normalerweise anschließend folgenden Zugriff auf Sektoren der angesteuerten Spur ja sowieso eine Überprüfung auf die korrekte Adresse (Sektor und Spur!) erfolgt.

Durch Rücksetzen des Verify-Bits in dieser SEEK-Routine des TOS kann man Diskzugriffe um bis zu 50% beschleunigen! Bei TOS im RAM läßt sich das Bit problemlos ändern (beim RAM-TOS von 11/85 ist die Speicherstelle $68C3 von $14 auf $10 zu ändern. Im RAM-TOS vom 06.02.87 ist die interessante Speicherstelle bei $7A1D).

Für TOS im ROM existieren Hilfsprogramme, die sich beim Start des Systems ins TOS einklinken und den Ansprung der ROM-SEEK-Routine in eine modifizierte SEEK-Routine im RAM umleiten, welche dann ohne Verify arbeitet.

Es gibt auch sogenannte FASTLOAD-EPROMS, die anstelle der Original-ROMs eingesetzt werden und bei denen das Verify-Bit nicht gesetzt ist. Im TOS 1.00 ist das die Speicherstelle $0DC7, im TOS 1.02 an Adresse $0EC7 und im TOS 1.04 bei $1517. Die Adreßangaben gelten jeweils für ROM U7 (ST), ROM U67 (STF) bzw. ROM U10 (MEGA ST).

STEP IN, STEP OUT – Schritt für Schritt über die Diskette

Mit jedem STEP-Kommando gibt der FDC einen Step-Impuls aus. Die Bewegungsrichtung ("Step direction") ist dabei die gleiche wie bei dem zuletzt ausgeführten Schritt. Ist das "Verify"-Bit gesetzt, erfolgt nach Ablauf der Zeit für die Steprate (r1-, r0-Bits im Kommandobyte) und Kopfberuhigung eine Überprüfung, ob der Kopf auf dem richtigen Track gelandet ist.

Bei den STEP-Kommandos existiert im Kommandobyte noch das sogenannte "Update"-Bit ("U"-Bit). Ist es gesetzt, wird das Track-Register je nach Steprichtung um 1 erhöht oder erniedrigt. Das Track-Register ist somit also immer up(to)date.

Um aber den Kopf nicht immer nur in eine Richtung bewegen zu können (STEP ändert ja nicht die Bewegungsrichtung), gibt es noch die STEP-IN- und STEP-OUT-Befehle. Wie der Name schon sagt, wird hier der Kopf einen Schritt nach innen (zur Diskettenmitte) oder nach außen (in Richtung Track 00) bewegt. "Update" und "Verify" arbeiten auch hier wie schon zuvor erläutert.

Nach Abschluß eines der Step-Kommandos wird wieder ein Interrupt ausgelöst. Der Status der Operation kann natürlich wieder aus dem Status-Register ausgelesen werden.

Kommandos des Typs II – Jetzt geht's an die Daten

Die FDC-Kommandos des Typs II dienen dem sektorweisen Datenzugriff auf die Disk.

Bevor ein READ- oder WRITE SECTOR-Kommando ausgeführt werden soll, muß das Sector-Register mit der Nummer des zu bearbeitenden Sektors geladen werden. Erst dann wird

das READ- oder WRITE SECTOR-Kommando gegeben. Wie schon bei den Typ I-Kommandos wird auch hier durch ein gesetztes "BUSY"-Bit im Status-Register signalisiert, daß der FDC fleißig ist. Ist auch noch das "E"-Bit gesetzt, wird der FDC mit dem Beginn des Datenzugriffs warten, bis die Kopfberuhigungszeit von 30 ms abgelaufen ist. Aber dann... geht erst mal die Sucherei los!

Der FDC versucht zunächst, ein ID-Feld auf der Spur zu finden. Die Spur- und Sektornummer des ID-Feldes muß natürlich mit den Daten im Track- und Sector-Register übereinstimmen.

Wenn dann auch noch die CRC-Prüfsumme über das ID-Feld stimmt, kann endlich der dem ID-Feld folgende Datensektor gelesen oder beschrieben werden. Für die Suche nach einem passenden ID-Feld hat der FDC fünf Diskumdrehungen Zeit, sonst gibt es einen Interrupt mit gesetztem "Record not found"-Bit im Status-Register.

Neben dem schon bekannten "Motor on"-Flag gibt es bei diesem Kommando-Typ noch ein "m"-Bit ("READ/WRITE Multiple sectors"):

- Bei *gelöschtem* "m"-Bit wird nur ein Sektor gelesen oder geschrieben.

- Bei *gesetztem* "m"-Bit werden alle Sektoren ab der Sektornummer im Sector-Register so lange gelesen oder beschrieben, bis der FDC z. B. per "Force interrupt"-Kommando unterbrochen wird oder ein Fehler auftritt. Das Sector-Register wird dabei kontinuierlich mit jedem gelesenen/geschriebenen Sektor heraufgezählt ("Auto updating"), bis die Sektornummer im Sector-Register größer als die größte Sektornummer in der Spur ist.

Beim ST würde der FDC also bei einer "READ MULTIPLE SECTORS"-Operation, beginnend mit dem Sektor 5, dann abgebrochen, wenn die Sektoren 5..9 gelesen wurden und der FDC vergeblich nach einem Sektor 10 Ausschau hielte (Sektornummer 1..9 sind ja normal).

Immerhin blieben dem FDC jedoch fünf Diskumdrehungen Zeit, den Sektor 10 vielleicht doch noch zu finden. Erst dann würde die Operation mit gesetztem "Record not found"-Bit im Status-Register abgebrochen. Um sich diese Wartezeit zu sparen, wird man den FDC mit "Force interrupt" dann unterbrechen, wenn genug Daten geschrieben oder gelesen wurden.

READ SEKTOR

Folgendes spielt sich ab, nachdem der FDC ein READ SECTOR-Kommando empfangen hat:

- Der FDC schaltet auf BUSY. ("BUSY"-Bit im Status Register setzen). Evtl. wird gewartet, bis der Motor mit Solldrehzahl läuft ("h"-Bit) und sich der Kopf wieder beruhigt hat ("E"-Bit).

- Ein passendes ID-Feld (Track, Sektor, CRC-Bytes) wird gesucht. Ist es gefunden, muß in einem der folgenden 43 Bytes eine "Data-address-mark" (DAM) gefunden werden. Für diese Sucherei hat der FDC fünf Umdrehungen Zeit (sonst erfolgt ein Interrupt mit gesetztem "Record not found"-Bit im Status-Register).

- Nach der gefundenen DAM werden Datenbytes eingelesen. Das erste Byte der Daten gelangt ins Data-Register. Der FDC signalisiert durch High am DRQ-Ausgang (Data Request = Pin 27 des FDC) und durch Setzen des "DRQ"-Bits im Status-Register, daß ein Datenbyte abgeholt werden muß.

- Sollte das Data-Register nicht rechtzeitig vor dem Eintreffen des nächsten Datenbytes ausgelesen worden sein, wird das vorhergehende Byte im Data-Register mit dem von Disk neu gelesenen Byte überschrieben. Im Status-Register wird dann das "Lost data"-Bit gesetzt.

- Es werden alle Datenbytes des Sektors nacheinander eingelesen. Sollte zu allem Übel auch noch die CRC-Prüfung einen Fehler ergeben, so wird das "CRC error"-Bit gesetzt und die Operation unterbrochen (auch wenn es sich um ein "READ MULTIPLE SECTORS"-Kommando handelt).

- Nach Abschluß der Leseoperation wird durch das "Sector type"-Bit im Status-Register signalisiert, ob die gelesenen Daten als gültig oder gelöscht zu deuten sind. *Gültige Daten* werden durch eine "Data address mark" = $FB vor den Sektordaten gemeldet. Das "Sector type"-Bit im Status-Register ist dann gelöscht. Ein Sektor mit *als "gelöscht" geltenden Daten* wird durch ein gesetztes "Sector type"-Bit angezeigt ("Data address mark"=$F8). Beim ST wird nur mit "gültigen" Sektoren gearbeitet, weil das TOS sich selbst merken kann, welche Sektoren frei und welche mit gültigen Daten belegt sind.

Soviel zum Kommando READ SECTOR.

Wie die Pfiffigen unter den Lesern jetzt wahrscheinlich schon messerscharf geschlossen haben, geht es nun weiter mit dem Kommmando

WRITE SECTOR

Bei diesem Kommando läuft folgender Vorgang ab:

- "BUSY"-Bit-Setzen, "Motor on"-Flag und evtl. Kopfberuhigung abwarten funktioniert genauso wie bei READ SECTOR.

- Die "Write protect"-Leitung wird abgefragt. Wenn die Disk im angesprochenen Laufwerk schreibgeschützt ist, geht's gar nicht erst weiter. Dann wird nämlich "interrupted" und im Status-Register zur Information das "Write protect"-Bit gesetzt.

- Wenn der Schreibzugriff erlaubt ist, erfolgt genau wie bei READ SECTOR die Suche nach einem Sektor mit passendem ID-Feld. Hat der FDC ein passendes ID-Feld mit korrekter CRC-Prüfsumme gefunden, wird nach zwei weiteren Lückenbytes der DRQ-Anschluß aktiviert und das "DRQ"-Bit im Status-Rgister gesetzt. Das erste zu schreibende Datenbyte muß innerhalb der nächsten neun Lückenbytes ins Data-Register geladen werden, sonst wird mit gesetztem "Lost Data"-Bit unterbrochen.

- 22 Bytes nach dem letzten CRC-Byte des ID-Felds (bei MFM) wird der FDC den "Write gate"-Anschluß aktivieren und zwölf Bytes mit $00 schreiben. Dann folgt die "Data-address-mark" (DAM=$FB bei "a0"-Bit=0, DAM=$F8 bei "a0"-Bit=1).

- Nun erst folgen die eigentlichen Datenbytes, welche von der CPU rechtzeitig ins Data-Register transportiert werden müssen. Die Signalisierung, wann wieder ein Byte fällig ist, geschieht über den DRQ-Anschluß (Pin 27 des FDC) und das "DRQ"-Bit im FDC-Status-Register. Gerät die CPU (im ST macht das der DMA-Baustein) bei der Datenbereitstellung "aus dem Tritt", wird für jedes fehlende Byte ein $00-Byte geschrieben. Dieser Fehler wird dann mit gesetztem "Lost Data"-Bit im Status-Register angezeigt.

- Nach dem letzten Byte des Datenfeldes eines Sektors wird die CRC-Prüfsumme geschrieben. Dann kommt noch ein Byte mit $FF (wer weiß wofür?), und anschließend erfolgt der Interrupt, der das Ende der Operation anzeigt.

Damit ist ein Sektor geschrieben. Beim WRITE SECTOR-Kommando kann das sogenannte Prekompensations-Bit ("P"-Bit) verwendet werden. Bei gelöschtem "P"-Bit erhöht sich damit die Datensicherheit auf den inneren Spuren, auf denen die Bits (und damit die kleinen "Stabmagnete") wesentlich näher beieinanderliegen als auf den äußeren Spuren.

Um "Bitwanderungen" durch zwei zu nah nebeneinanderliegende "Stabmagnete" von vornherein zu erschweren (nebeneinanderliegende gleiche Magnetpole stoßen sich ja bekanntlich ab, während sich ein ungleiche Pole gar nicht nah genug kommen können!), werden gleiche Pole etwas näher beieinander geschrieben (sie werden sich schon von allein voneinander entfernen). Ungleiche Pole werden etwas weiter voneinander entfernt aufgezeichnet, weil sie ja doch das Bestreben haben, sich einander zu nähern.

Beim ST arbeitet ATARI immer mit gelöschtem "P"-Bit, egal auf welche Spur zugegriffen wird. Die Schreibvorkompensation ist also immer eingeschaltet!

Diagnose- und Formatierkommandos – Typ III-Befehle

Mit den Kommandos des Typs III hat man die Möglichkeit, mehr Informationen von der Diskette zu holen bzw. darauf zu schreiben als mit den Sektorzugriffen der Typ II-Kommandos.

Adresse gesucht! – READ ADDRESS

Das Kommando READ ADDRESS ermöglicht es, nur das nächste "vorbeikommende" ID-Feld eines Sektors (eben seine Adresse auf der Disk) auf einer Spur einzulesen. Dies ist nur 6 Bytes lang (Spur-, Seiten-, Sektornummer, Sektorlängenkennung und die zwei CRC-Bytes). Auch bei READ ADDRESS besteht natürlich wieder die Möglichkeit, mit dem "Motor on"-Flag ("h"-Bit) und einer Kopfberuhigungszeit ("E"-Bit) zu arbeiten.

Nach Empfang des Kommandos setzt der FDC wie gehabt das "BUSY"-Status-Bit und wird dann das nächste ID-Feld Byte für Byte an das Data-Register übermitteln. Von dort müssen diese Bytes nach und nach, entsprechend der Signalisierung mit der DRQ-Leitung bzw. dem DRQ-Status Bit, abgeholt werden (bei verspätetem Auslesen wird wieder "Lost Data" signalisiert!).

Die Gültigkeit der gelesenen Bytes wird natürlich per CRC-Checksumme überprüft und entsprechend im Status-Register angezeigt (Bit 3 = "CRC Error"-Bit). Zusätzlich wird die gelesene Tracknummer des ID-Felds ins Sector-Register (!) kopiert und steht dort für evtl. Vergleiche zur Verfügung. Das Ende der READ ADDRESS-Operation wird auch wieder ganz normal über Interrupt und Reset des "BUSY"-Bits angezeigt.

WRITE TRACK

ist der eigentlich wichtigste Befehl überhaupt, über den der Floppy-Controller verfügt. Nur mit diesem Befehl ist es nämlich möglich, *alle* erforderlichen Informationen auf die Disk zu bekommen.

Mit WRITE TRACK wird eine Diskette formatiert!

Dazu legt man sich in einem Speicherbereich des Computers Byte für Byte ein "Doppel" der zu formatierenden Spur an. Dieser "Memory Track" enthält, einschließlich Markierungs- und Synchronisationsbytes, alle Informationen für den zu schreibenden Track. Und diese Spur sieht dann folgendermaßen aus (zur besseren Übersicht ein Auge auf Abbildung 8.8 halten):

Da wären also zunächst 60 Bytes mit $4E, dann zwölfmal die $00, dreimal die $A1 mit fehlenden Taktflanken zur Identifizierung des ersten ID-Feldes, dann die ID-Address-Mark ($FE) selbst,... Moment mal, wie war das gerade mit den fehlenden Taktflanken? Wo kommen die denn her, bzw. wie läßt man die denn weg?

Fehlende Flanken sind auffällig – Markierungen auf der Disk

Um solche Marken mit fehlenden Taktflanken zu schreiben, gibt es beim WRITE TRACK-Befehl des FDC bestimmte Bytes, die der FDC nicht als "normale" zu schreibende Bytes

ansieht, sondern als *Befehlsbytes* interpretiert, die besondere Reaktionen im FDC auslösen. Bekommt der FDC beim WRITE TRACK-Befehl eines der folgenden Bytes übermittelt, so werden die nachstehend beschriebenen Reaktionen ausgelöst:

Byte	Funktion
$00...$F4	Schreibe normales Byte $00...$F4
$F5	Schreibe $A1 mit fehlenden Taktflanken und starte CRC-Prüfsummenbildung
$F6	Schreibe $C2 mit fehlenden Taktflanken
$F7	Schreibe Inhalt des Prüfsummengenerators auf Disk (2 CRC-Bytes)
$F8...$FF	Schreibe normales Byte $F8...$FF

Diese Angaben gelten für die Betriebsart "doppelte Schreibdichte" (MFM-Betrieb). Für den Betrieb in "einfacher Schreibdichte" (FM-Betrieb), die beim FDC des ST ja nur über einen Eingriff in dessen Hardware eingeschaltet werden kann, haben die Bytes $F8..$FF noch eine besondere Funktion. Darauf soll hier jedoch nicht weiter eingegangen werden.

Mit diesen Informationen läßt sich also nun eine komplette Spur im Speicher nachbilden, welche dann mit dem WRITE TRACK-Befehl auf Disk geschrieben werden kann.

Beim ST sieht so eine "Spur im Speicher" dann wie in Abbildung 8.8 aus.

Hat man auf der Disk erst mal eine solche Struktur aufgebracht, kann der FDC nun die nötigen Markierungen und Synchronisationsinformationen für die sektorweise Datenspeicherung finden.

Eine ganze Spur im Griff – READ TRACK

Mit dem READ TRACK-Befehl wird der FDC aufgefordert, den kompletten Inhalt der Spur, über welcher der Kopf gerade positioniert ist, zu lesen.

Alle Bytes, einschließlich GAPs (Lückenbytes zur Synchronisation), ID-Felder, DAMs und natürlich Datenbytes werden, beginnend mit der fallenden Flanke des Indexpulses, bis zum nächsten Indexpuls innerhalb einer Diskumdrehung gelesen!

Das Ende der Operation wird wie gehabt per Interrupt und durch Löschen des "BUSY"-Bits gemeldet.

Ein evtl. aufgetretener "Lost data"-Fehler durch nicht rechtzeitiges Reagieren auf die DRQ-Signalisierung wird natürlich wieder im Status-Register angezeigt.

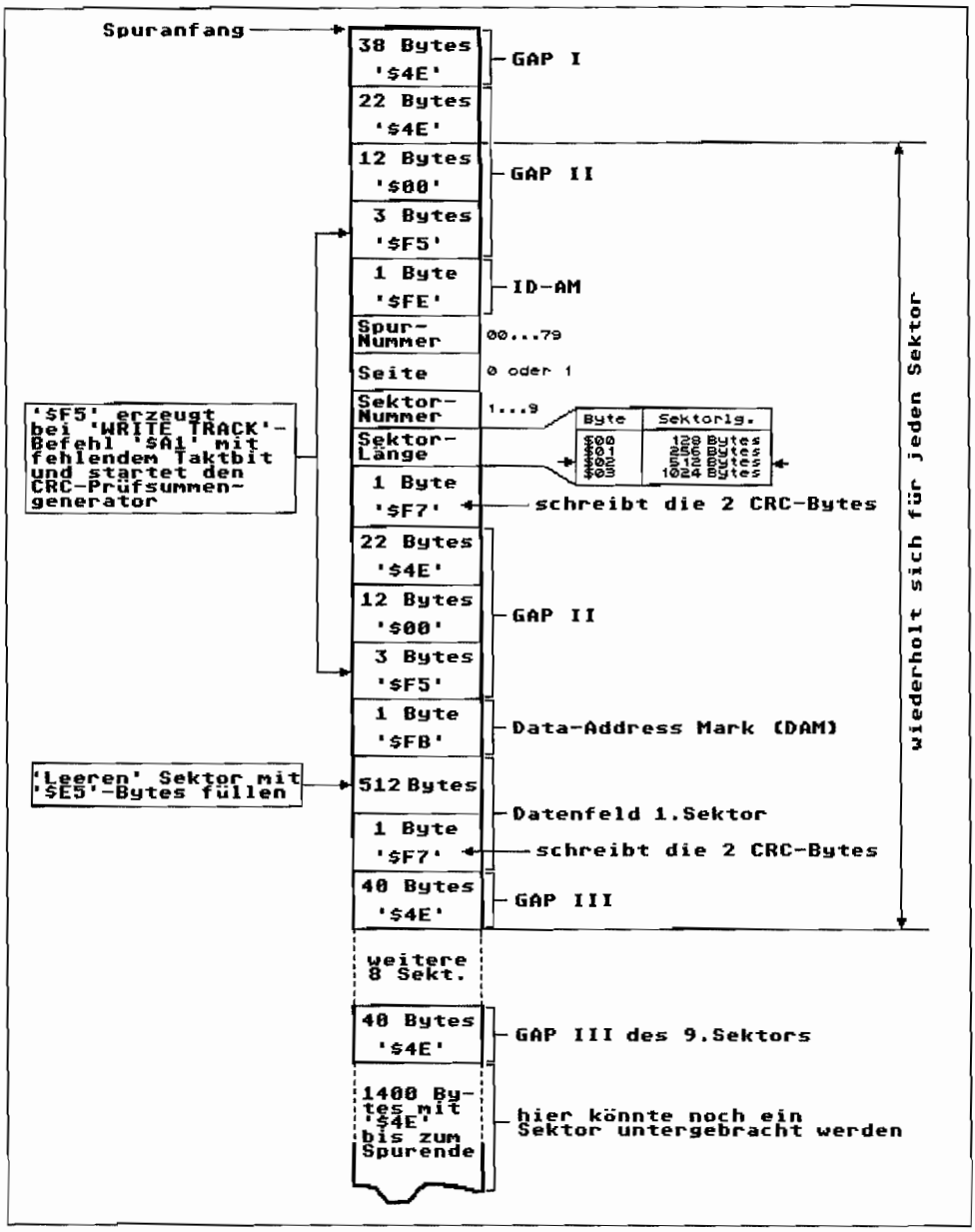


Abb. 8.8: So wird ein Track im Speicher zusammengebaut und dann mit dem Befehl WRITE TRACK auf Disk geschrieben (Formatierung)

Fehler bei der Befehlsausführung – Der Bug bei READ TRACK

Für Diagnoseanwendungen wäre der READ TRACK-Befehl geradezu ideal, wenn er denn korrekt funktionieren würde!

Und an den Fehlfunktionen ist der sogenannte Address Mark-Detector des FDC schuld, der bei diesem Befehl ständig eingeschaltet ist. Der AM-Detector des FDC ist zuständig für das Erkennen jener bereits beim WRITE TRACK erwähnten Bytes, die mit fehlenden Taktinformationen ($A1 und $C2) geschrieben werden.

Zwar beginnt der FDC mit der fallenden Flanke des Indeximpulses mit dem Lesen der Daten. Jedoch fängt er meist mitten in einem Byte an, da der FDC in dem eintreffenden Bitstrom vom Laufwerk zunächst nicht erkennen kann, wo denn ein Byte anfängt und wo es aufhört. Die ersten Bytes der Spur werden also falsch eingelesen.

Erst durch Erkennen des ersten Markierungsbytes (durch die fehlende Taktinformation ja besonders herausragend) rastet die Synchronisation ein. Leider kümmert sich der ständig aktive AM-Detector anscheinend nicht um bereits erkannte Bytegrenzen und rastet sich auch immer dann neu ein, wenn er in einer einlaufenden Bitfolge meint, wieder ein Markierungsbyte ($A1 oder $C2) erkannt zu haben.

Durch diese zusätzlichen vermeintlichen Markierungsbytes synchronisiert sich der FDC viel öfter auf Markierungen, die eigentlich gar keine sind und damit dann auch falsch!

Fazit: Den Befehl READ TRACK am besten gar nicht benutzen!

Kommandos des Typs IV

Manchmal ist er (der FDC) kaum zu halten – Der FORCE INTERRUPT-Befehl

Nun zum letzten Kommando des Floppy Controllers. Es läßt sich nicht den vorhergehenden Typen I..III zuordnen und bekommt damit eine eigene Typzuordnung.

Mit dem FORCE INTERRUPT-Kommando wird eine laufende Operation abgebrochen. Dieser Befehl wird hauptsächlich benutzt, um ein READ/WRITE MULTIPLE SECTORS-Kommando zu unterbrechen oder um das Status-Register auf Typ I-Statussignalisierung zurückzuschalten. Bei Typ II..III-Befehlen haben einige Status-Bits ja eine andere Bedeutung als bei Typ I-Kommandos. FORCE INTERRUPT kann *jederzeit* an den FDC gesendet werden. Über die Bits 2 und 3 des Befehlsbytes wird dem FDC mitgeteilt, wann er denn die laufende Operation abwürgen soll. Folgende Möglichkeiten gibt es:

Befehlsbyte	Funktion
$D8	Sofortige Unterbrechung
$D4	Unterbrechung bei nächstem Indeximpuls
$D0	Keine Interruptsignalisierung, aber Operation trotzdem abbrechen (Clear Immediate Interrupt)

Empfängt der FDC ein FORCE INTERRUPT-Kommando, so geht der INTRQ-Anschluß auf High, sobald die Bedingung für den Interrupt erfüllt ist. (Nein! Diesmal braucht der FDC das "BUSY"-Bit nicht setzen, es ist ja von der zu unterbrechenden Operation noch auf High und wird durch FORCE INTERRUPT zurückgesetzt!)

Beim $D4-Interrupt (Indeximpuls Interrupt) geht der INTRQ-Anschluß (Pin 28 des FDC) also beim nächsten Indeximpuls auf High (gut zur Drehzahlmessung geeignet!). Wird ein $D8-Interrupt (Immediate Interrupt) ausgelöst, "springt" INTRQ sofort nach Empfang des Befehls auf "High".

Das führt zu der Frage, wann denn die INTRQ-Leitung normalerweise wieder zurück auf Low geht? Interrupts treten ja nicht nur per FORCE INTERRUPT, sondern auch nach jeder abgeschlossenen Operation auf.

Das "Löschen" der INTRQ-Leitung geschieht sozusagen automatisch durch Auslesen des Status-Registers oder beim Laden des Command-Registers mit einem neuen Befehl. Aber wie immer gibt es auch hier eine Ausnahme von der Regel. Nach einem "FORCE Immediate INTERRUPT" läßt sich nur durch einen $D0-Interrupt (Unterbrechung ohne Unterbrechungs-Signalisierung) die INTRQ-Leitung löschen (das ist dann die Unterbrechung der Unterbrechung)!

Bevor man nach einem FORCE INTERRUPT wieder ein Kommando an den FDC gibt, sollte man mindestens 16 µs warten. Wenn nicht, wird FORCE INTERRUPT nicht ausgeführt!

Beim ST ist alles ganz anders – Die FDC-Programmierung

Nein, ganz so schlimm ist es nun doch nicht. Man hat jedoch nicht die Möglichkeit, direkt auf die FDC-Register zuzugreifen.

Zwischen CPU und FDC hat ATARI den DMA-Baustein (DMA = Direct Memory Access oder Datentransfer Mal Anders) gesetzt. Das hat den großen Vorteil, daß die CPU z. B. bei

Diskzugriffen nicht ständig an der DRQ-Leitung des FDC horchen muß, ob wieder ein Datenbyte vom FDC abgeholt oder dorthin gebracht werden muß.

Das erledigt die DMA-Einheit viel eleganter. Sie braucht nur zu wissen, wie viele Sektoren (Bytes) ab welcher Adresse im Speicher auf Disk geschrieben oder von dort eingelesen werden sollen, und schon "geht die Post ab".

Wie im Prinzip eine FDC-Operation unter Zuhilfenahme der DMA-Einheit abläuft, soll hier kurz erläutert werden.

Eine Beschreibung der einzelnen Register des DMA-Bausteins sind im Teil II, Kapitel 1, "Die Zentraleinheit", zu finden.

- VBlank-Diskroutine vom DMA- und FDC-Baustein fernhalten ("flock"<>0 setzen! Das muß natürlich, wie die gesamte FDC-Programmierung, im Supervisormodus geschehen, da "flock" (Adr. $43E) im Supervisor-RAM liegt!).

```
st      flock                   ; "flock" auf <> setzen
```

- Floppy-Laufwerk und -Seite über Port A des PSG (Programmable Sound Generator) selektieren. Man benutzt dazu die XBIOS-Funktion #29 ("Offgibit") und #30 ("Ongibit").

  Die Steuerleitungen zum Laufwerk sind Low-aktiv, also wird mit einem Low (= Bit im Port A löschen) auf der entsprechenden "Drive select"- bzw. "Side select"-Leitung die Laufwerks-/Seiten-Auswahl durchgeführt!

  Funktion der Bits zur Disksteuerung im Port A des PSG:

```
Bit 0: 0 = Seite 1,
       1 = Seite 0 angewählt
Bit 1: 0 = Drive A selektiert
Bit 2: 0 = Drive B selektiert
```

  Beispiel:

```
move.w      #$07,-(SP)      ; Beide Drives deselektieren
move.w      #$1E,-(SP)      ; und Seite 0 anwählen
trap        #14             ; "Ongibit" aufrufen
addq.l      #4,SP

move.w      #$FA            ; Seite 1 und Drive B selektieren
```

```
move.w      #$1D
trap        #14             ; "Offgibit" aufrufen
addq.l      #4,SP
     ...
```

- Der DMA-Einheit muß mitgeteilt werden, wo die zu schreibenden/lesenden Daten im Speicher stehen bzw. nach dem Lesen von der Disk stehen sollen.

  Die Anfangsadresse dieses Speicherbereichs wird der DMA-Einheit im DMA-Base- und Counter-Register byteweise übergeben.

```
$FF8609   -   High-Byte    ("dmahigh")
$FF860B   -   Mid-Byte     ("dmamid")
$FF860D   -   Low-Byte     ("dmalow")
```

  Achtung! Hierbei ist wichtig, daß die Adresse in der Reihenfolge Low-Byte -> Mid-Byte -> High-Byte an das DMA-Base-und Counter-Register übergeben wird.

  Beispiel:

```
move.l      #STRTADR,d7     ; Anfangsadr. für DMA holen
move.b      d7,dmalow       ; Low-Byte setzen
asr.l       #8,d7           ; Um 1 Byte nach rechts schieben
move.b      d7,dmamid       ; Mid-Byte setzen
asr.l       #8,d7           ; Um 1 Byte nach rechts schieben
move.b      d7,dmahi        ; High-Byte setzen.
     ...
```

- Der 32-Byte FIFO-Buffer der DMA-Einheit muß gelöscht werden. Das geschieht durch "Kippeln" mit Bit 8 im DMA-Mode-Register ("fifo" an Adr. $FF 8606).

  Gleichzeitig wird mit diesem Bit eingestellt, in welche Richtung der DMA-Datentransport denn laufen soll.

  $90, $190, $90 — Wenn diese Word-Folge ins DMA-Mode Register geschrieben wird, ist anschließend der FIFO-Buffer gelöscht und der DMA-Baustein auf "Lesen" programmiert.

  $190, $90, $190 — Diese Word-Folge löscht ebenfalls den FIFO-Buffer, schaltet die DMA-Einheit jedoch auf schreibenden DMA-Zugriff.

Beispiel:

```
lea         fifo,A5          ; Zum Code sparen
move.w      #$190,0(A5)      ; FIFO-Buffer löschen
move.w      #$90 ,0(A5)      ; und DMA-Baustein auf
move.w      #$190,0(A5)      ; LESEN schalten
     ...
```

Ein Transfer von Daten zwischen Speicher und FIFO-Buffer im DMA-Baustein findet immer nur statt, wenn mindestens 16 Bytes im FIFO "aufgelaufen" sind. Darunter tut sich überhaupt nichts.

Bei Sektorgrößen von 512 Bytes gibt es da keine Schwierigkeiten. Will man jedoch z. B. nur ein ID-Feld von Disk lesen (6 Bytes), so kommen diese sechs Bytes gar nicht erst im Speicher an. Da hilft nur eins: Mehrere ID-Felder lesen.

- Die Zahl der zu transportierenden Datenblöcke zu 512 Bytes (Sektoren) wird der DMA-Einheit im Sector-Counter-Register ("diskctl" an Adr. $FF 8604) mitgeteilt. Bei READ/WRITE TRACK sind das ca. 6250 Bytes, also rund 13 Blöcke.

  Beispiel:

  ```
  move.w      #13,diskctl      ; 13 Sektoren übertragen
  ```

- Nun wird beim DMA-Chip um "Audienz" beim FDC gebeten, d. h., im DMA-Mode-Register wird durch Löschen von Bit 4 die "Durchreiche" zu den FDC-Registern geöffnet. Durch diesen Kniff wird statt des Sector-Counter-Registers unter Adr. $FF 8604 ("diskctl") ein Register des FDC zugänglich.

  Welches FDC-Register nun unter "diskctl" erscheint, wird durch die Bits 1 und 2 des DMA-Mode-Registers bestimmt. Diese steuern nämlich die Adreßleitungen, mit denen der DMA-Baustein dem FDC-Chip mitteilt, welches Register dieser denn bitte schön zum Zugriff bereithalten soll.

  Die auf der nächsten Seite folgenden Kommandoworte für das DMA-Mode-Register wählen eines der FDC-Register aus.

  Wenn nach dem Setzen der FDC-Register schreibender DMA-Betrieb vorgesehen ist, muß zusätzlich das Bit 8 im DMA-Mode-Register gesetzt werden (siehe auch weiter vorn, bei der Beschreibung für das Löschen des FIFO-Buffers).

Word im DMA-Mode Register	angewähltes Register des Floppy Controllers
$(1)80	Command/Status-Register (Read=Status)
$(1)82	Track-Register
$(1)84	Sector-Register
$(1)86	Data-Register

Die Register des FDC werden auf die für die gewünschte Operation erforderlichen Werte gesetzt (Track-, Sector-Register).

Beispiel:

```
move.w      #$86,fifo       ; Zugriff auf Data-Register
move.w      #$29,diskctl    ; Track-Nr. $29 ins Data-Register
move.w      #$80,fifo       ; Zugriff auf Command-Register
move.w      #$14,diskctl    ; SEEK-Befehl mit Verify an FDC
     ...
```

- Das Ende des Kommandos wird dann vom FDC per Interrupt an den MFP, Port I5, signalisiert. Also empfiehlt es sich, von Zeit zu Zeit Bit 5 des Port Registers des MFP ("gpip" an Adr. $FF FA01) zu testen. Bei Low von Bit 5 des Ports ist der FDC fertig.

- Das DMA-Status-Register ("fifo" an Adr. $FF 8606, READ) gibt Auskunft, ob der Datentransfer über den DMA-Baustein ordnungsgemäß abgelaufen ist. Interessant ist nur Bit 0. Bei einem Fehler ist dieses auf Low. Die weitere Bitbelegung ist bei der Beschreibung der DMA-Register zu finden (Teil II, Kapitel "Die Zentraleinheit"). Vor Zugriff auf das DMA-Status-Register sollte lt. Atari-Vorgaben über das DMA-Mode-Register der Zugriff auf das Sector-Counter-Register eingestellt sein (Bit 4 im DMA-Mode Register gesetzt).

  Beispiel:

```
move.w      #$90,fifo       ; Sector-Cnt.-Reg. selektieren
move.w      fifo,D7         ; DMA-Status -> Register D7
     ...
```

- Aber nicht nur bei der DMA-Operation kann ein Fehler aufgetreten sein. Deshalb schnell noch das FDC-Status-Register ansehen. Dazu muß dieses natürlich erst wieder in der "Durchreiche" der DMA-Einheit erscheinen. Also wird mit den bereits vorgestellten Kommandowörtern über das DMA-Mode-Register das FDC-Status-Register angewählt.

Beispiel:

```
move.w #     $80,fifo          ; FDC-Status-Reg. selektieren
move.w       diskctl,D7        ; FDC-Status -> Register D7
     ...
```

– Wenn wir fertig sind, dürfen die VBlank-Diskroutinen nun wieder auf die DMA-Einheit zugreifen ("flock" auf 0 setzen)

```
clr.w        flock             ; "flock" = 0 setzen
```

Kapitel 9: Das Atari Computer System Interface (ACSI)

Die Computer der ST-Serie verfügen zusätzlich zu den schon mehr oder weniger üblichen Standard-Schnittstellen über eine Atari-spezifische Schnittstelle: den ACSI-Bus. Atari hat hier eine Schnittstelle zum Anschluß von "intelligenten" schnellen Peripheriegeräten wie z. B. Harddisks, Laserdrucker oder CD-ROMs definiert.

Der ACSI-Bus besteht im wesentlichen aus einem bidirektionalen 8-Bit-Datenbus und einigen Steuer- und Meldeleitungen. Sein Verhalten und das verwendete Protokoll ähneln stark dem des SCSI-Busses (Small Computer System Interface-Bus). ATARI hat den SCSI-Bus auf seine Anforderungen zurechtgestutzt und verwendet nur SCSI-Kommandos der Klasse 0, wegen einer Abweichung im ersten Befehlsbyte.

Die Betreuung dieser Schnittstelle liegt im ST bei der DMA-Einheit. Deshalb sind lt. Atari-Spezifikation auch Datentransfer-Geschwindigkeiten von über 8 MBit/s möglich. Abbildung 9.1 skizziert kurz die Komponenten am ACSI-Bus.

Erforderlich ist zunächst der Initiator. Im Gegensatz zum SCSI-Bus gibt es beim ACSI-Bus nur einen davon und das ist der ST. Er verwaltet den ACSI-Bus.

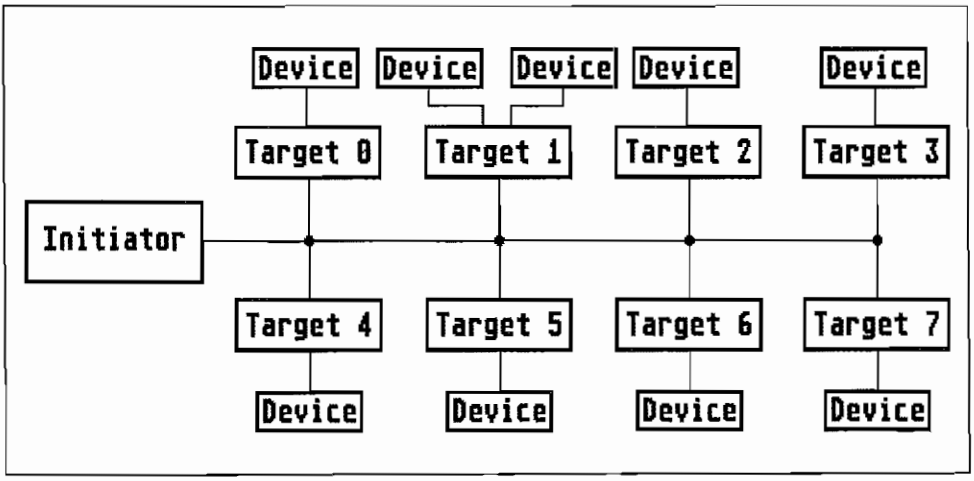


Abb. 9.1: Der ACSI-Bus mit seinen verschiedenen Komponenten

Bis zu acht Targets (Zieleinheiten) sind möglich. Ein Target enthält einen eigenen Controller zur Steuerung der zum Target gehörigen Geräte (z. B. Harddisk). Dabei ist es durchaus denkbar, daß ein Target mehrere Geräte (z. B. Festplatten) steuert.

Folgende Eigenschaften müssen die angeschlossenen Targets aufweisen:

- Ein an ein Target gesandtes Kommando darf nicht "in der Luft hängen bleiben", bis z. B. der Controller im Target bereit ist. Mit Ausnahme des letzten müssen alle Bytes eines Kommandoblocks durch einen Interrupt vom Target quittiert werden.

  Ein "DRIVE NOT READY"-Error muß also sofort nach Empfang gesendet werden, oder der Initiator erklärt das angesprochene Target für "nicht anwesend".

- Jedes Target testet sich nach einem RESET selbst und muß alle erforderlichen Grundeinstellungen vornehmen. Ein eigenes Selbsttest-Kommando gibt es nicht.

- Der Initiator kann nach einer festgelegten Zeit "Timeout" signalisieren und (sollte) einen RESET für das Target mit dem "Timeout"-Fehler auslösen (können). Ein softwaremäßiger RESET einzelner Targets durch den ST ist jedoch nicht möglich. Wenn RESET, dann nur per Druck auf die RESET-Taste des ST und für alle Targets.

- Wenn ein Target eine Operation abgeschlossen und zum Abschluß ein Statusbyte gesendet hat, ist der ACSI-Bus freigeschaltet.

Als ACSI-Busanschluß am ST benutzt ATARI eine 19pol. Buchse des Typs "DB 19S". Sie sieht zwar ähnlich wie der Centronics-Anschluß aus, es besteht jedoch durch die unterschiedliche Polzahl keine Verwechslungsgefahr.

Der SCSI-Bus verwendet hingegen eine 50polige Verbindungsleitung zwischen den einzelnen Komponenten, wobei eine große Zahl der Leitungen als Masseleitungen dienen und somit als Abschirmung gegen Störeinstrahlungen.

Diese bei Atari fehlenden Masseleitungen führen dazu, daß die Signalleitungen kurz ausfallen müssen, um Störeinstrahlungen gering zu halten. Zwischen einzelnen Devices sind maximal 24 Inch (ca. 60 cm) zulässig. Die Gesamtleitungslänge darf ca. 1,8 m nicht überschreiten, wobei nicht mehr als vier Targets am ACSI-Bus angeschlossen sein dürfen.

Die Belegung der Buchse und die Einbindung des ACSI-Busses in die ST-Hardware zeigt der nachfolgende Schaltungsauszug.

Auf dem ACSI-Bus wird mit TTL-Pegeln gearbeitet.

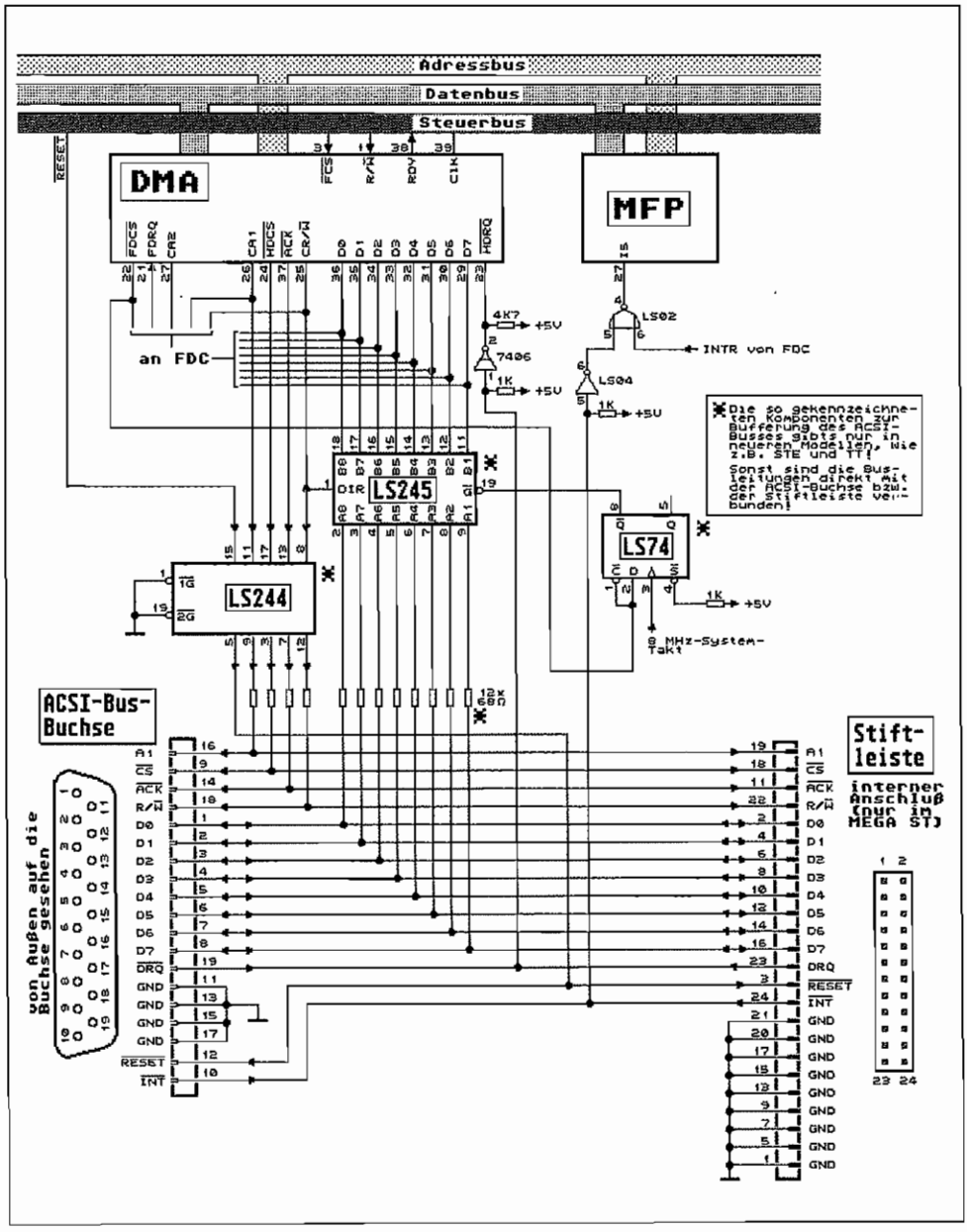


Abb. 9.2: So läuft der ACSI-Bus am ST auf

Die einzelnen Anschlüsse habe dabei folgende Funktion:

$\overline{\text{RESET}}$ Mit einem Low auf dieser Leitung werden die angeschlossenen Targets vom Initiator zurückgesetzt. Das RESET-Signal ist 12 µsec "lang" aktiviert.

R/$\overline{\text{W}}$ Damit signalisiert der Initiator die Datentransportrichtung. Bei Low schreibt der Initiator an ein Target. Liegt hier High, erwartet der Initiator Informationen von einem Target. Das Signal wirkt nur bei der Command-Phase (dazu später mehr). Das Handshaking während des Datenaustauschs läuft über ACK und DRQ. Die gewünschte Datentransportrichtung wird dem Target während der Command-Phase über die gesendeten Command-Bytes "beigebracht".

$\overline{\text{ACK}}$ Auch dieses Signal ist wieder Low-aktiv und an die Targets gerichtet. Hiermit quittiert der Initiator ein _DRQ (Data ReQuest = Bereitschaft zum Datenaustausch) von einem Target.

$\overline{\text{CS}}$ Dieses Low-aktive Signal wird für das "Chip Select" des Target-Command- (während Command-Phase) oder Target-Status-Registers (während Status-Phase) benutzt. Wird vom Initiator an die Targets gerichtet.

$\overline{\text{A1}}$ Vom Initiator an Target gerichtetes Signal. Mit einem Low auf A1 wird die Command-Phase "eingeläutet".

$\overline{\text{DRQ}}$ Während des DMA-Datentransfers (nicht während der Command-Phase!) signalisiert das angesprochene Target mit einem Low auf dieser Leitung, daß ein weiteres Byte auf den Bus gelegt werden soll (vom Initiator zum Target) oder gelegt wird (vom Target zum Initiator).

$\overline{\text{INT}}$ Während der Command-Phase quittiert das angesprochene Target mit einem Low-Impuls auf dieser Leitung den Empfang eines Bytes. Der Abschluß einer DMA-Operation auf dem ACSI-Bus und der Beginn der Status-Phase wird ebenfalls per Low auf dieser Leitung signalisiert.

D0..D7 Richtig geraten! Das ist der Datenbus des ACSI.

Soviel zur Bedeutung der einzelnen Bus-Leitungen.

Anschließend nun einiges über den Bus-Fahrplan (die einzelnen Bus-Phasen) des ACSI.

- Der Datentransport erfolgt asynchron und hält sich an ein Request/Acknowledge-Protokoll (Datenanforderung mit anschließender Empfangsbestätigung). Für jedes zu

übertragende Byte erfolgt ein Handshake (nach dem Schema: Vorsicht, ein Byte kommt! Datenbyte übertragen – Empfangsbestätigung geben).

- Der Informationsaustausch auf dem ACSI-Bus läuft immer nach dem gleichen Schema ab.

  Command-Phase -> Data in/-out-Phase -> Status-Phase ->
  Command-Phase -> Data in/-out-Phase ...

- Ein RESET hat die höchste Priorität und muß immer ausgeführt werden.

- Die Targets müssen den Bus ständig "im Auge" behalten und auf evtl. Kommandos umgehend reagieren. Ein "Timeout" sollte laut Atari nach drei Sekunden erfolgen, nachdem bei einem Command-Byte-Transfer keine Bestätigung vom Target erfolgt ist.

- Ein Target "antwortet" nur, wenn es "angesprochen" wird.

- Eine laufende Operation eines Targets kann nur per RESET unterbrochen werden. Der Initiator sollte per Software ein Resetsignal auf der RESET-Leitung für die Targets auslösen können (der ST kann's nicht! Ein RESET wird nur erzeugt, wenn der ST ebenfalls zurückgesetzt wird.)

- Ein angesprochenes Target hat den Bus für sich, bis es das Status-Byte zurücksendet. Das sollte nicht länger als vier Sekunden ab Operationsbeginn auf sich warten lassen, da sonst der Initiator ein "Timeout" erzeugt und das Target für "nicht anwesend" erklärt. Wobei die Festlegung der Timeout-Zeit von der Software her erfolgt, die den ACSI-Bus bedient!

Zur Verdeutlichung der einzelnen Bus-Phasen folgen nun einige Zeitdiagramme.

Die Command-Phase

Der Initiator kündigt durch Low auf der "A1"-Leitung eine Command-Phase an. Während der Übertragung des ersten Command-Bytes bleibt "A1" auf Low (siehe Abbildung 9.3).

Mit einem Low auf der "CS"-Leitung signalisiert der Initiator den angeschlossenen Targets die Gültigkeit der Daten auf dem Datenbus. Auf der "R/W"-Leitung liegt ein Low, weil Daten vom Initiator an die Targets geschrieben werden.

Alle Targets haben für sich das erste Command-Byte daraufhin zu untersuchen, ob der Initiator mit "ihnen sprechen" möchte. Das "auserwählte" Target muß umgehend den Empfang des

Command-Bytes bestätigen (mit einem Low-Impuls auf der "_INT"-Leitung). Geschieht die Bestätigung nicht rechtzeitig, kann ein "Time-out" vom Initiator signalisiert werden. Das gesuchte Target wird dann als "nicht anwesend" registriert.

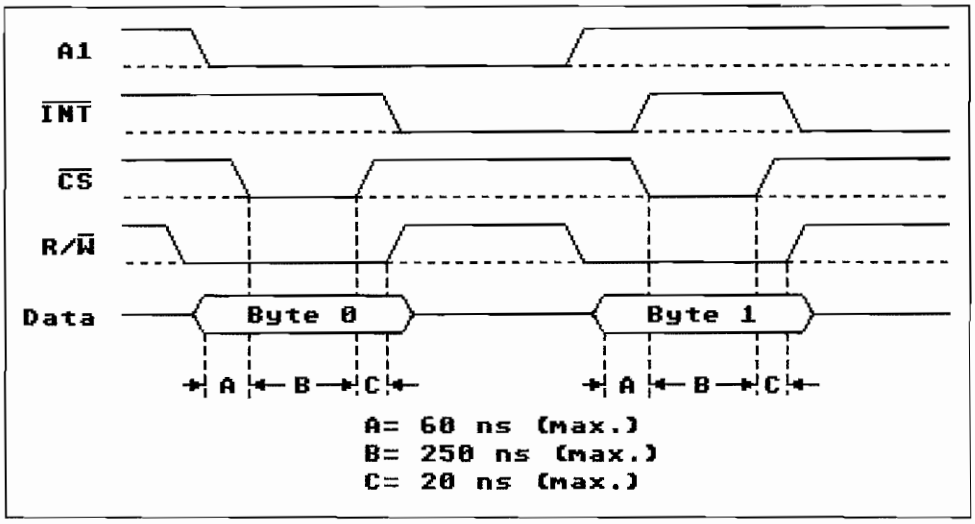


Abb. 9.3: Die Command-Phase

Hat das angesprochene Target den Empfang des ersten Command-Bytes mit Low auf der "_INT"-Leitung quittiert, so existiert nun quasi ein festgeschalteter Datenkanal zwischen Initiator und diesem Target. Die weiteren fünf Command-Bytes werden mit High auf der "A1"-Leitung übertragen. Alle anderen Targets halten sich bis zum Beginn der nächsten Command-Phase (die wird ja wieder mit einem Low auf der "A1"-Leitung angekündigt) aus dem Datenaustausch heraus.

Beim ST wird der ACSI-Bus von der DMA-Einheit bedient. Mit deren Unterstützung ist sowohl ein Einzelbyte-Transfer als auch ein blockorientierter Datenaustausch (ein ganzes Paket mit Bytes wird übertragen) möglich. Die Datenbytes werden in der Command-Phase noch per "Handbetrieb" (im Einzelbyte-Transfer) übertragen. Der "Turbolader" (Datenaustausch in DMA-Betriebsart) wird erst während der "Data in/out-Phase" zugeschaltet! Das angewendete Prinzip für den Einzelbyte-Transfer ist das gleiche wie bei der Übergabe von Daten an die Register des Floppy-Disk-Controllers (FDC).

Beim *FDC*-Registerzugriff kann man ja mit gelöschten Bits 3+4 im DMA-Mode-Register ("fifo" an Adr. $FF8606) eine "Durchreiche" zu einem mit den Bits 1+2 im DMA-Mode-Regi-

ster auszuwählenden FDC-Register öffnen. Die "Durchreiche" ist dabei das sogenannte Controller-Access-Register ("diskctl" an Adr. $FF 8604). Siehe hierzu auch Teil II, Kapitel 1, Abschnitt "DMA im ST".

Für einen Einzelbytezugriff auf den *ACSI-Bus* muß im DMA-Mode-Register dagegen das Bit 3 gesetzt und Bit 4 gelöscht sein. Wird nun ein Byte in das Controller-Access-Register geschrieben, gelangt es von dort direkt auf die Datenleitungen des ACSI-Busses. Hat man vorher im DMA-Mode-Register das Bit 1 gelöscht, wurde damit die "A1"-Leitung des ACSI-Busses auf Low gezogen, was ja für die angeschlossenen Targets soviel wie "Achtung, das erste Command-Byte kommt" bedeutet. Nur für die Ausgabe des ersten Command-Bytes ist "A1" auf Low!

Die Signale "_CS" und "R/W" werden von der DMA-Einheit intern erzeugt. "R/W" legt die Datenrichtung fest und ist für die Ausgabe eines Command-Bytes auf Low (es werden ja Daten ans Target geschrieben).

Das angesprochene Target hat den Empfang jedes Command-Bytes dann mit einem Low-Impuls auf der "_INT"-Leitung zu quittieren (sonst "Timeout"!). Dieser Interrupt läuft im ST ja, wie schon beim FDC beschrieben, am Port I5 des MFP auf. Der Zustand des Ports I5 kann an Adresse $FF FA01 ("gpip"), Bit 5, abgefragt werden. Das letzte Command-Byte wird allerdings nicht mehr per Low-Impuls auf der "_INT"-Leitung vom Target quittiert, sondern es wird direkt zur nächsten Phase übergegangen!

Data-in/out-Phase

Während der Data-in/out-Phase (siehe Abbildung 9.4 und 9.5) steuert die DMA-Einheit des ST den Datentransfer selbsttätig. Es wird dabei im Blockmodus gearbeitet (Übertragung von Datenbytes in Blocks zu 512 Bytes).

"A1" ist dabei ständig auf "High". Die Synchronisation der Datenübertragung und damit die Geschwindigkeit wird ausschließlich durch das Target bestimmt. Durch Low auf der "_DRQ"-Leitung signalisiert das Target dem Initiator, daß Daten gebraucht werden (in der Data-out-Phase) oder daß Daten vom Target geliefert werden können (in der Data-in-Phase).

Mit Low auf der "_ACK"-Leitung quittiert der Initiator, daß Daten nun auf dem Bus bereitgestellt sind (Data-out-Phase) oder die Daten vom Target übernommen wurden (Data-in-Phase).

Wenn man sich die angegebenen Zeiten z. B. bei der Data-out-Phase betrachtet, so läßt sich ausrechnen, daß die Zykluszeit zur Übertragung eines Bytes max. 240 ns + 250 ns + 240 ns = 730 ns beträgt. Es können also mindestens 1/730 ns = 1,37 MByte/Sek. übertragen werden.

In der Data-in-Phase kann man unter günstigen Verhältnissen sogar ca. 1,8 MByte/Sek. erreichen (wenn der Target-Controller da mitmacht)!

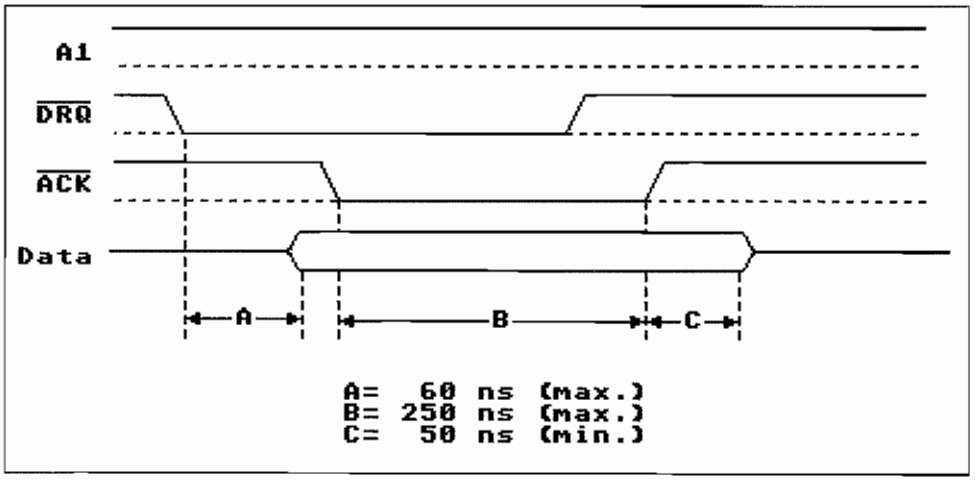


Abb. 9.4: Die Data-in-Phase

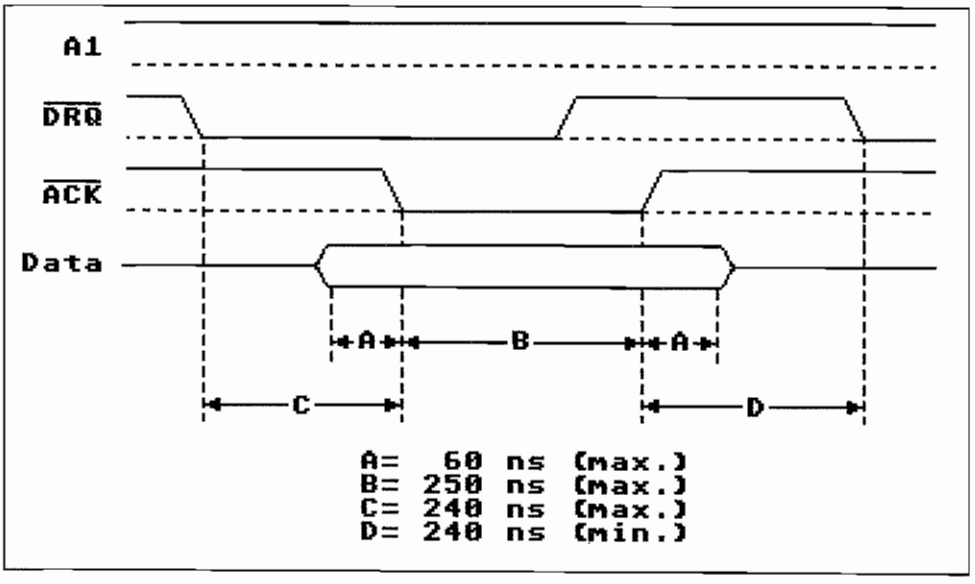


Abb. 9.5: Die Data-out-Phase

Die Status-Phase

Nach Abschluß einer Data-in/out-Phase wird per Interrupt (Low auf der "_INT"-Leitung) das Ende dieser Phase signalisiert. Das Status-Byte für die Operation wird dann vom Target an den Initiator übermittelt.

Im ST wird dieses in das Controller-Access-Register geschrieben und kann von dort ausgelesen werden ("diskctl" an Adr. $FF 8604). Dazu muß natürlich wieder die "Durchreiche" zum ACSI-Bus geöffnet werden (DMA-Mode-Register, Bit 3 gesetzt und Bit 4 gelöscht!).

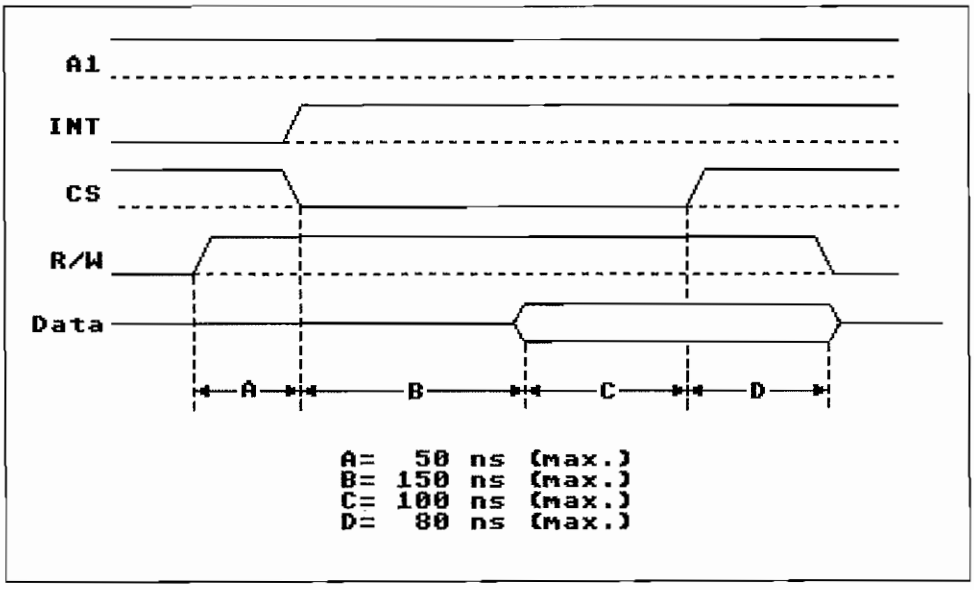
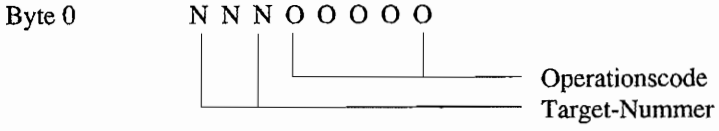
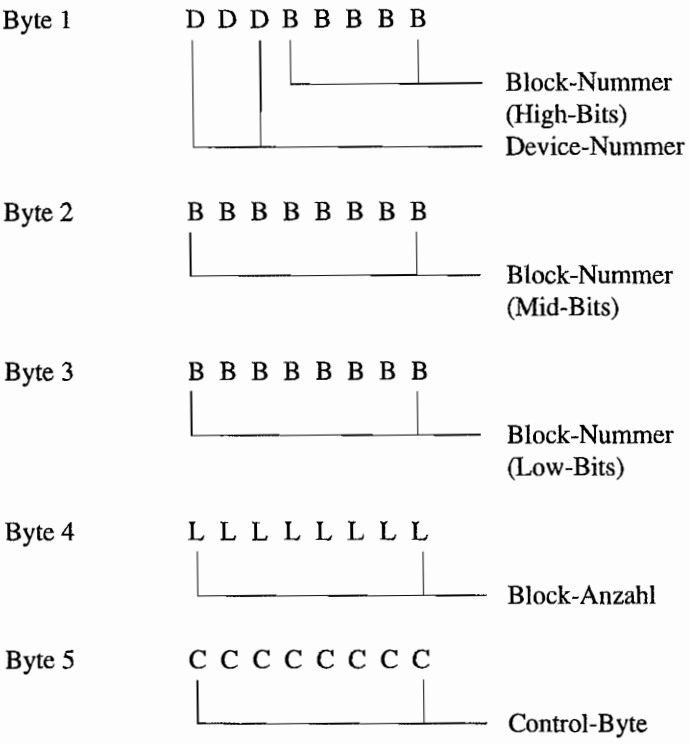


Abb. 9.6: Die Status-Phase (Target -> Initiator)

Der ACSI-Command-Descriptor-Block

Während der Command-Phase werden dem Target (in der Mehrzahl der Fälle dürfte das z. Zt. die Hard-Disk sein) sechs Bytes übermittelt. Dieser Block hat folgenden Aufbau:





Die *Target-Nummer* bezeichnet eines der acht möglichen Targets (Target 0..7) am ACSI-Bus. Alle angeschlossenen Targets überprüfen direkt nach Eintreffen des ersten Command-Bytes anhand dieser Target-Nummer, ob sie angesprochen sind. Das angesprochene Target bestätigt dann unmittelbar dem Initiator per _INT-Signalisierung den Empfang und wartet auf die folgenden Bytes des Blocks. Die Target-Nummer kann am Target-Controller mit DIP-Schaltern oder Jumpern (Steckbrücken) eingestellt werden.

Der *Operationscode* enthält den Code für das auszuführende Kommando. Mit den fünf zur Verfügung stehenden Bits sind max. 32 verschiedene Kommandos möglich.

Die *Device-Nummer* wählt eines von acht möglichen Geräten des Targets aus, auf welches sich das Kommando beziehen soll. Ist das Target ein Hard-Disk-Controller, so könnte damit eins von maximal acht Hard-Disk-Laufwerken angesprochen werden.

Die unteren fünf Bits und die folgenden 16 Bits der Commando-Bytes 2+3 bilden eine 21 Bit *Block-Nummer* für einen Datenblock. Im Falle einer Hard-Disk ist das der log. Sektor, auf den zugegriffen werden soll.

Der log. Sektor wird dann vom Hard-Disk-Controller in eine Zylinder-, Kopf- und phys. Sektornummer umgesetzt.

Das vierte Byte des Command-Descriptor-Blocks enthält die *Block-Anzahl*. Dieser 8-Bit-Wert gibt an, wie viele Blocks (à 512 Bytes, also z. B. Sektoren einer Hard-Disk) übertragen werden sollen. Dieser Wert muß von Null verschieden sein (warum sollte man auch Null Blocks = 0 Bytes übertragen wollen?).

Das fünfte Byte (*Control-Byte*) ergänzt den Operationscode des Command-Descriptor-Blocks um evtl. spezielle Informationen. Es ist jedoch bei den Hard-Disk-Commands unbenutzt und auf Null gesetzt.

Nachstehend ist der Befehlssatz wiedergegeben, den der Adaptec-Controller in der Hard-Disk SH204/SH205/MEGAFILE 30/60 mindestens verstehen sollte. Für die Bezeichnung der Bits im Command-Descriptor-Block werden die folgenden Buchstaben benutzt:

N = Target-Nummer B = Block-Nummer
D = Device-Nummer L = Block-Anzahl

Der Befehlssatz des Hard-Disk-Controllers

Befehl:	**Test Drive Ready ($0)**							
Byte 0:	N	N	N	0	0	0	0	0
Byte 1:	D	D	D	0	0	0	0	0
Byte 2:	0	0	0	0	0	0	0	0
Byte 3:	0	0	0	0	0	0	0	0
Byte 4:	0	0	0	0	0	0	0	0
Byte 5:	0	0	0	0	0	0	0	0

Testet, ob die Device-Nr. DDD des Targets NNN angeschlossen und bereit ist. Wenn das der Fall ist, wird ein entsprechendes Status-Byte (=0) zurückgeliefert.

Befehl:	**Restore to Zero ($1)**							
Byte 0:	N	N	N	0	0	0	0	1
Byte 1:	D	D	D	0	0	0	0	0
Byte 2:	0	0	0	0	0	0	0	0
Byte 3:	0	0	0	0	0	0	0	0
Byte 4:	0	0	0	0	0	0	0	0
Byte 5:	0	0	0	0	0	0	0	0

Lesekopf auf Spur 0 positionieren.

Befehl:	**Request sense ($3)**								
Byte 0:	N	N	N	0	0	0	1	1	
Byte 1:	D	D	D	0	0	0	0	0	
Byte 2:	0	0	0	0	0	0	0	0	
Byte 3:	0	0	0	0	0	0	0	0	
Byte 4:	0	0	0	0	0	1	0	0	(4 Bytes zurückliefern!)
Byte 5:	0	0	0	0	0	0	0	0	

Wie wir noch sehen werden, meldet das Target eventuelle Fehler über ein gesetztes Bit 1 im Status-Byte. Genauere Informationen über die Art des Fehlers kann man mit diesem Kommando erhalten.

Es werden vier Bytes mit Statusinformationen vom Target-Controller eingelesen. Die vier Bytes werden per DMA-Betrieb zum ST übertragen, womit wieder das Problem mit dem FIFO-Puffer der DMA-Einheit auftritt (erinnnern Sie sich noch an READ ADDRESS bei der Beschreibung des Floppy-Disk-Controllers?)!

Damit überhaupt Bytes aus dem FIFO-Buffer der DMA-Einheit im Speicher des ST ankommen, muß man ja mindestens 16 Bytes übertragen. Also sollte der Befehl mindestens viermal ausgeführt werden!

Die vier gelesenen Bytes haben dabei folgende Bedeutung:

Byte-Nr	**Inhalt**	**Bedeutung**
0	V F F F F F F F	Fehlernummer log. Blockadresse in den Bytes 1..3 gültig
1	X X X B B B B	High-Bits der Blockadresse Reserviert
2	B B B B B B B	Mid-Bits der Blockadresse
3	B B B B B B B	Low-Bits der Blockadresse

Im folgenden sehen Sie die in Byte 0 angezeigten Fehlernummern und deren Bedeutung:

Fehler-Nummer	Bedeutung
	Drive Errors
$00	No sense (Alles klar! Kein Fehler)
$01	No Index (keine Indeximpulse gefunden)
$02	No Seek complete (SEEK noch nicht beendet)
$03	Write fault (Schreibfehler)
$04	Drive not ready (Drive noch bei der Initialisierung?)
$06	No Track $00 (Spur $00 nicht zu finden)
	Controller Errors
$10	ID ECC Error (ECC-Fehler im ID-Feld)
$11	Uncorrectable Data Error (nicht korrigierbarer Fehler im Datenfeld eines Sektors)
$12	ID Address Mark not found
$13	Data Address Mark not found
$14	Record not found (Sektor nicht zu finden)
$15	Seek Error
$18	Data Check in No Retry Mode
$19	ECC Error during Verify
$1A	Bad Block (Zugriff auf ausgemappten Block)
$1C	Unformatted or Bad format
	Command Errors
$20	Invalid Command (Ungültiges Kommando)
$21	Invalid Address (Ungültige Blockadresse)
$23	Volume Overflow (Ungültige Block-Endadresse)
$24	Invalid Argument
$25	Invalid Drive Number (falsche Gerätenummer)
$26	Byte zero Parity Check
$28	Cartridge changed (Platte wurde gewechselt)
$2C	Error Count overflow (Fehlerzähler hat die vereinbarte Zahl von Fehlern überschritten)
	Sonstige Fehler
$30	Controller Selftest failed

Befehl: **Format drive ($4)**

Byte 0:	N	N	N	0	0	1	0	0	
Byte 1:	D	D	D	G	G	0	F	0	
Byte 2:	V	V	V	V	V	V	V	V	(Virgin-Byte)
Byte 3:	I	I	I	I	I	I	I	I	(Interleave-Faktor High-Byte)
Byte 4:	I	I	I	I	I	I	I	I	(Interleave-Faktor Low-Byte)
Byte 5:	0	0	0	0	0	0	0	0	

Mit nur einem Befehl kann man die gesamte Harddisk formatieren. Dabei greift der Hard-Disk-Controller entweder auf die Format-Informationen zurück, die vorher mit dem Befehl "Mode select" übertragen wurden, oder liest sich diese von der Platte.

Bei gelöschtem "F"-Bit in Byte 1 werden die Sektoren mit einem "Virgin"-Wert von $6C beschrieben, während bei gesetztem "F"-Bit der Wert in Byte 2 als "Virgin"-Byte genommen wird.

Sind die beiden "G"-Bits im Byte 1 gesetzt, so kann dem Festplattencontroller noch eine max. 1024 Byte lange Liste mit Daten über defekte Stellen auf der Platte im DMA-Betrieb hinterhergeschickt werden. Der Controller überspringt diese Defektstellen bei der Formatierung. Bei dem z. Zt. verwendeten Controller in den Atari-Hard-Disks ist diese Liste wie folgt aufgebaut:

Byte	Bedeutung
$00	Null (Reserviert)
$01	Null (Reserviert)
$02..$03	Länge der Defektliste in Bytes
$04..$06	Zylindernummer des ersten Defekts
$07	Kopfnummer (normalerweise 0..3)
$08..$0B	Entfernung d. Defekts v. Indeximpuls in MFM-Bytes
$0C..$0E	Zylindernummer des nächsten Defekts
$0F	Kopfnummer
$10..$13	Entfernung d. Defekts v. Indeximpuls in MFM-Bytes
...	...

Die Bytes 3..4 des Command-Descriptor-Blocks enthalten den Wert für den Interleave-Faktor (max. 16).

Befehl: Read Sector ($8)

Byte 0:	N	N	N	0	1	0	0	0	
Byte 1:	D	D	D	B	B	B	B	B	(log. Blockadresse, High-Bits)
Byte 2:	B	B	B	B	B	B	B	B	(log. Blockadresse, Mid-Bits)
Byte 3:	B	B	B	B	B	B	B	B	(log. Blockadresse, Low-Bits)
Byte 4:	L	L	L	L	L	L	L	L	(Blockanzahl)
Byte 5:	0	0	0	0	0	0	0	0	

Es wird ab der in den Bytes 1..3 angegebenen log. Blockadresse die in Byte 4 angegebene Zahl von Blöcken mit je 512 Bytes gelesen.

Befehl: Write Sector ($A)

Byte 0:	N	N	N	0	1	0	1	0	
Byte 1:	D	D	D	B	B	B	B	B	(log. Blockadresse High-Bits)
Byte 2:	B	B	B	B	B	B	B	B	(log. Blockadresse, Mid-Bits)
Byte 3:	B	B	B	B	B	B	B	B	(log. Blockadresse, Low-Bits)
Byte 4:	L	L	L	L	L	L	L	L	(Blockanzahl)
Byte 5:	0	0	0	0	0	0	0	0	

Es wird ab der in den Bytes 1..3 angegebenen Blockadresse die in Byte 4 angegebene Zahl von Blöcken (mit je 512 Bytes) geschrieben.

Befehl: Seek Block ($B)

Byte 0:	N	N	N	0	1	0	1	1	
Byte 1:	D	D	D	B	B	B	B	B	(log. Blockadresse, High-Bits)
Byte 2:	B	B	B	B	B	B	B	B	(log. Blockadresse, Mid-Bits)
Byte 3:	B	B	B	B	B	B	B	B	(log. Blockadresse, Low-Bits)

Befehl: **Seek Block ($B)**

Byte 4:	0	0	0	0	0	0	0	0
Byte 5:	0	0	0	0	0	0	0	0

Die in den Bytes 1..3 angegebene Blockadresse wird angesteuert.

Befehl: **Mode select ($15)**

Byte 0:	N	N	N	1	0	1	0	1	
Byte 1:	D	D	D	0	0	0	0	0	
Byte 2:	0	0	0	0	0	0	0	0	
Byte 3:	0	0	0	0	0	0	0	0	
Byte 4:	L	L	L	L	L	L	L	L	(Parameteranzahl in *Bytes*!)
Byte 5:	0	0	0	0	0	0	0	0	

Dem Controller wird eine Liste mit Parametern für das entsprechende Plattenlaufwerk übermittelt.

Die Länge der Liste steht in Byte 4 des Command-Descriptor-Blocks.

Per DMA wird dem Controller vom ST dann ein Parameterblock mit folgendem Inhalt geschickt:

Byte	Bedeutung
$00	Reserviert (=$00)
$01	Reserviert (=$00)
$02	Reserviert (=$00)
$03	Länge der folgenden Extent-Descriptor-List (=8 Bytes)

Die im Anschluß folgende Extent-Descriptor-List hat folgenden Aufbau:

Byte	Bedeutung
$00	Density (=$00)
$01	Reserviert (=$00)
$02	Reserviert (=$00)
$03	Reserviert (=$00)
$04	Reserviert (=$00)

Byte	Bedeutung
$05	Sektorgröße (High-Byte)
$06	Sektorgröße (Mid-Byte)
$07	Sektorgröße (Low-Byte)

Die Sektorgröße bei ATARI-Festplatten beträgt 512 Bytes. Anschließend an diese beiden Blöcke darf noch eine Liste mit Laufwerksparametern angehängt werden, die folgendermaßen aufgebaut sein muß:

Byte	Bedeutung
$00	Format-Code (1=Festplatte, 2=Wechselplatte)
$01..$02	Anzahl der Zylinder (bei SH205/MEGAFILE 30 normalerweise 612...664)
$03	Zahl der Schreib-/Leseköpfe (bei SH205/MEGAFILE 30 = 4)
$04..$05	Nummer des Zylinders, ab dem mit reduziertem Schreibstrom gearbeitet wird
$06..$07	Nummer des Zylinders, ab dem mit Schreib-Precompensation gearbeitet wird
$08	Abstand der Landezone vom ersten (oder letzten) Datenzylinder. Bei gesetztem Bit 8 liegt die Landezone am äußeren Rand der Platte, sonst jenseits der innersten Spur. Die unteren sieben Bits geben an, wie viele Zylinder vom ersten/letzten Datenzylinder entfernt die Landezone für die Köpfe eingerichtet ist.
$09	Steprate. Bei Atari-Hard-Disks sind drei Einstellungen möglich: $00 = 3 Millisekunden $01 = 28 Mikrosekunden (Buffered Seek!) $02 = 12 Mikrosekunden (Buffered Seek!) = Standardeinstellung
$0A	In diesem Byte sind nur die Bits 2 und 3 relevant.

Bytewert	*Bedeutung*
0 0 0 0 0 0 0 0	Softsektorierte Wechselplatte
0 0 0 0 0 1 0 0	Softsektorierte Festplatte = Standardeinstellung
0 0 0 0 1 0 0 0	Hardsektorierte Wechselplatte (?)
0 0 0 0 1 1 0 0	Hardsektorierte Festplatte (?)

$0B	Anzahl der Sektoren/Spur. Üblicherweise ist dieser Wert bei der SH205 auf 17 eingestellt. Bei einem Interleave > 1 kann man auch 18 Sektoren/Spur unterbringen und so die Plattenkapazität erhöhen. Dabei muß man aber mit einem etwas langsameren Zugriff rechnen (wegen des höheren Interleave-Faktors).

Wenn diese Parameter mit dem Mode-select-Kommando übertragen wurden, kann mit einem nachfolgenden Format-Befehl ($4) die Platte mit den eingestellten Parametern formatiert werden. Nach dem Formatieren werden die neuen Parameter in Spur 0 der Platte abgelegt.

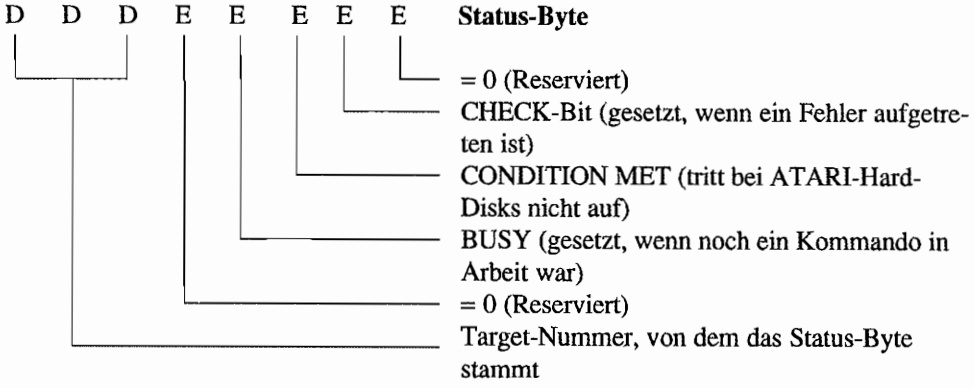
Befehl: **Mode sense ($1A)**

Byte 0:	N	N	N	1	1	0	1	0	
Byte 1:	D	D	D	0	0	0	0	0	
Byte 2:	0	0	0	0	0	0	0	0	
Byte 3:	0	0	0	0	0	0	0	0	
Byte 4:	L	L	L	L	L	L	L	L	(Parameteranzahl in *Bytes*!)
Byte 5:	0	0	0	0	0	0	0	0	

Dieser Befehl funktioniert genau umgekehrt wie Mode select. Es werden die Parameter der Platte aus der Spur 0 ausgelesen und per DMA an den ST übermittelt.

Der Stand der Dinge – Das Status-Byte

Wie ja schon mehrmals vorher erwähnt, wird jede Operation mit dem Senden eines Status-Bytes an den ST abgeschlossen. Dieses Status-Byte enthält in den ersten drei Bits die Nummer des Targets, von dem das Status-Byte stammt. Die niedrigwertigen fünf Bits enthalten einen Error-Code, wobei jedes Bit eine bestimmte Information über den Stand der ausgeführten Operation liefert.



Leider wird durch einen Fehler im Hostadapter der SH204 auch bei Fehlern immer ein Status-Byte von $00 an den ST übermittelt. Bei der SH205 erhält man ein korrektes Status-Byte.

Der ST ergreift die Initiative

Eine Datenübertragung mit DMA-Betrieb (Data-in/out-Phase) auf dem ACSI-Bus mit dem ST als Initiator läuft nach untenstehendem Schema ab. Weitere Einzelheiten zu den einzelnen Registern der DMA-Einheit sind in Teil II, Kapitel 1, Abschnitt "DMA im ST" zu finden.

Die Registerzugriffe müssen natürlich (ebenso wie auf die Sytemvariablen) im Supervisormodus ablaufen.

- Um die VBlank-Disk-Routine vom DMA-Baustein fernzuhalten, wird "flock" an Adresse $43E gesetzt.

- Das DMA-Base und Counter-Register ("dmahigh" an Adr. $FF 8609, "dmamid" an Adr. $FF 860B und "dmalow" an Adr. $FF 860D) wird mit der Anfangsadresse für den zu übertragenden Datenblock geladen (Reihenfolge Low-, Mid-, High-Byte beachten!).

- Löschen des FIFO-Buffers und des DMA-Status-Registers der DMA-Einheit durch "Klappern" (0-1-0- oder 1-0-1-Zustandswechsel) mit dem R/W-Bit (Bit 8) im DMA-Mode-Register ("fifo" an Adr. $FF 8606). Sinnvollerweise klappert man mit dem R/W-Bit so, daß nach dem Löschen des FIFO-Buffers die Datentransportrichtung auch gleich richtig eingestellt ist. Der "Turbo-Lader" (DMA-Einheit) bleibt vorläufig noch gesperrt (Bit 7 im DMA-Mode-Register gesetzt), indem man einen DMA-Betrieb mit dem FDC "vortäuscht".

- Im DMA-Sector-Counter-Register ("diskctl" an Adr. $FF 8604) wird der DMA-Einheit die "Arbeitsmenge", d. h. die Anzahl der zu übertragenden 512-Byte-Blocks, mitgeteilt. Um Zugriff auf das Sector-Counter-Register zu bekommen, muß im DMA-Mode-Register das Bit 4 gesetzt sein!

- Der Command-Descriptor-Block muß ausgegeben werden. Dazu muß die "Durchreiche" zum ACSI-Bus geöffnet werden. Also im DMA-Mode-Register das Bit 3 setzen (wir wollen an den ACSI-Bus) und Bit 4 löschen ("Durchreiche" öffnen).

- Ausgabe des ersten Command-Bytes mit Low auf der "A1"-Leitung des ACSI-Busses (Bit 1 im DMA-Mode-Register steuert die "A1"-Leitung und muß also in diesem Fall gelöscht werden). Warten auf Target-Empfangsbestätigung durch Low auf der INT-Leitung (evtl. "Timeout" auslösen), die am MFP, Port I5 aufläuft (Adresse $FF FA01 = "gpip", Bit 5 testen!).

- Ausgabe der weiteren fünf Command-Bytes im "Handbetrieb" (ohne DMA-Hilfe, Byte für Byte) mit High auf der "A1"-Leitung (Bit 1 im DMA-Mode-Register gesetzt) und Abwarten der Quittierung vom Target über die "_INT"-Leitung (wieder über Bit 5 in

"gpip" abfragbar). Nach dem Senden des letzten Kommandobytes kommt zur Quittierung *kein* Signal über "_INT", sondern erst wenn das Kommando abgeschlossen wurde!

Für die Ausgabe der Command-Bytes empfiehlt Atari eine "move.l"-Anweisung. Dadurch kann man die im Adreßraum des ST unmittelbar hintereinanderliegenden Controller-Access-Register ("diskctl" an Adr. $FF 8604) und DMA-Mode-Register ("fifo" an Adr. $FF 8606) "gleichzeitig" beschreiben. Es ist wohl schon mal vorgekommen, daß der DMA-Baustein des ST bei zwei unmittelbar aufeinander folgenden Word-Zugriffen auf DMA-Mode-Register und dann auf das Controller-Access-Register, die CS-Leitung des ACSI-Busses zweimal aktiviert hat. Das führte dann im Target-Controller verständlicherweise zu einiger Verwirrung.

Im High-Word des Long-Zugiffs steht dann der Wert für das Controller-Access-Register (der auf den ACSI-Bus gegeben wird) und im Low-Word bereits die Einstellung des DMA-Mode-Registers für das nächste Command-Byte.

- Nach Senden des Command-Descriptor-Blocks wird durch Löschen von Bit 7 im DMA-Mode-Register der "Turbo-Lader" auf den ACSI-Bus geschaltet. Das wird in der Regel mit der Ausgabe des letzten Command-Bytes gleich miterledigt!

- Nun läuft alles weitere nicht mehr "per Hand" sondern über den "Turbo-Lader" (die DMA-Einheit des ST), bis das DMA-Sector-Counter-Register auf Null gezählt ist. Es sind dann alle vorgesehenen Datenblocks übertragen. Solange darf der DMA-Baustein aber nicht gestört werden.

- Das Ende der DMA-Operation wird durch Low auf der "_INT"-Leitung an den Port I5 des MFP ("gpip" an Adr. $FF FA01) gemeldet. Die CPU sollte also immer "ein Auge" auf diesen Port haben.

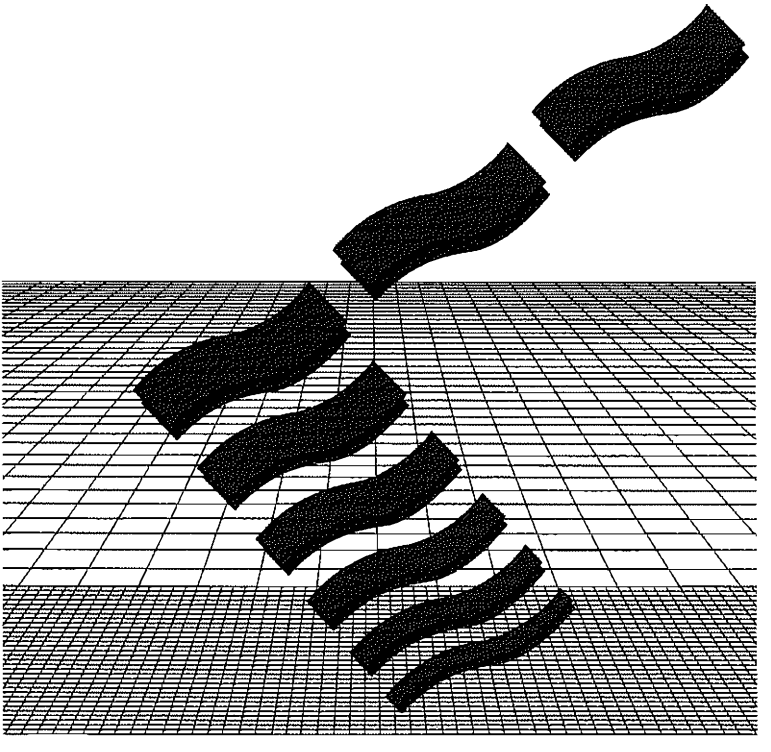
- Zum Testen auf Erfolg oder Mißerfolg der ganzen Aktion wird das Status-Byte des Target-Controllers ausgelesen. Dazu ist der "Durchgriff" auf das Controller-Access-Register durch Setzen des Bits 3 des DMA-Mode Registers zu öffnen (Bit 4 muß gelöscht sein). Aus dem Controller-Access-Register ("diskctl" an Adr. $FF 8604) kann nun das Status-Byte ausgelesen werden.

- Der DMA-Status der Operation kann ebenfalls überprüft werden ("fifo" an Adr. $FF 8606 auslesen). Die Bits 0..2 sind bei korrekt abgeschlossener Operation gesetzt.

- Nun wird wieder auf Floppy-Zugriff umgeschaltet und damit der "Turbo-Lader" vom ACSI-Bus getrennt ($0080 ins DMA-Mode-Register schreiben). "flock" wird zurückgesetzt, um die VBlank-Interrupt-Routine wieder an die Floppy zu lassen.

So läuft der Datentransport zwischen ST als Initiator und einem Target ab.

Teil III



Die Hardware des TT

Kapitel 1: Schneller, höher, weiter – Der Prozessor MC68030

In den ATARI TT-Computern findet sich ein Prozessor der Motorola MC68030-Serie. Der Systemtakt für den Prozessor liegt, ebenso wie der des Coprozessors vom Typ MC68882, bei 32 MHz.

Der MC68030 ist aufwärtskompatibel zu seinen Vorgängern, verfügt jedoch über einen wesentlich höheren Integrationsgrad und eine neue Architektur.

Hier nur eine stichpunktartige Auflistung der Eigenschaften und Möglichkeiten des MC68030:

- Harvard-Architektur (intern getrennte Busse für Adressen und Daten).
- Getrennte On-Chip-Cache-Speicher für Daten und Befehle (je 256 Byte).

Damit ist die interne, gleichzeitige Bearbeitung von Befehlen und Daten möglich.

- Integrierte PMMU (Paged Memory Management Unit) zur Umsetzung von Zugriffen auf logische Adressen in entsprechende physikalische Adressen (wichtig in Multitasking-Systemen!).
- Vollständiger 32 Bit (nicht gemultiplexter!) Adreßbus.
- 32-Bit-Datenbus.
- Erweiterter Bus-Controller mit der Unterstützung von drei verschiedenen Busbetriebsarten (Asynchron, Synchron und Burst Data Transfer).
- Direkte Unterstützung des Coprozessors.

Da die Möglichkeiten der MC68030 sehr vielfältig sind, würde es den Rahmen dieses Buches sprengen, auf alle Eigenschaften einzugehen.

Wer sich intensiv und erschöpfend mit diesem Prozessor beschäftigen möchte, sei auf das "MC68030 Enhanced 32 Bit Mikroprocessor User's Manual" hingewiesen. Auf nahezu "halber Profibuchstärke" wird dort nur der Prozessor von vorn bis hinten beleuchtet, ohne dabei noch besonders intensiv auf so "Nebensächliches" wie den Befehlssatz einzugehen!

Hardware

Im Gegensatz zur MC68000-CPU in z. B. den ST(E)-Maschinen reicht ein 64pol. Gehäuse längst nicht mehr für alle Anschlüsse aus (die gehen ja allein für den Adreß- und Datenbus schon drauf!). Verwendet wird ein quadratisches 13 x 13 Pin-Grid-Gehäuse mit 122 Anschlüssen! Eine Vielzahl von Pins dienen der Betriebsspannungszufuhr und dem Masseanschluß, um eine bessere Störunterdrückung zu erreichen.

In der nachfolgenden Abbildung ist dargestellt, welche Signale (nach Funktionen geordnet) beim MC68030 herausgeführt sind.

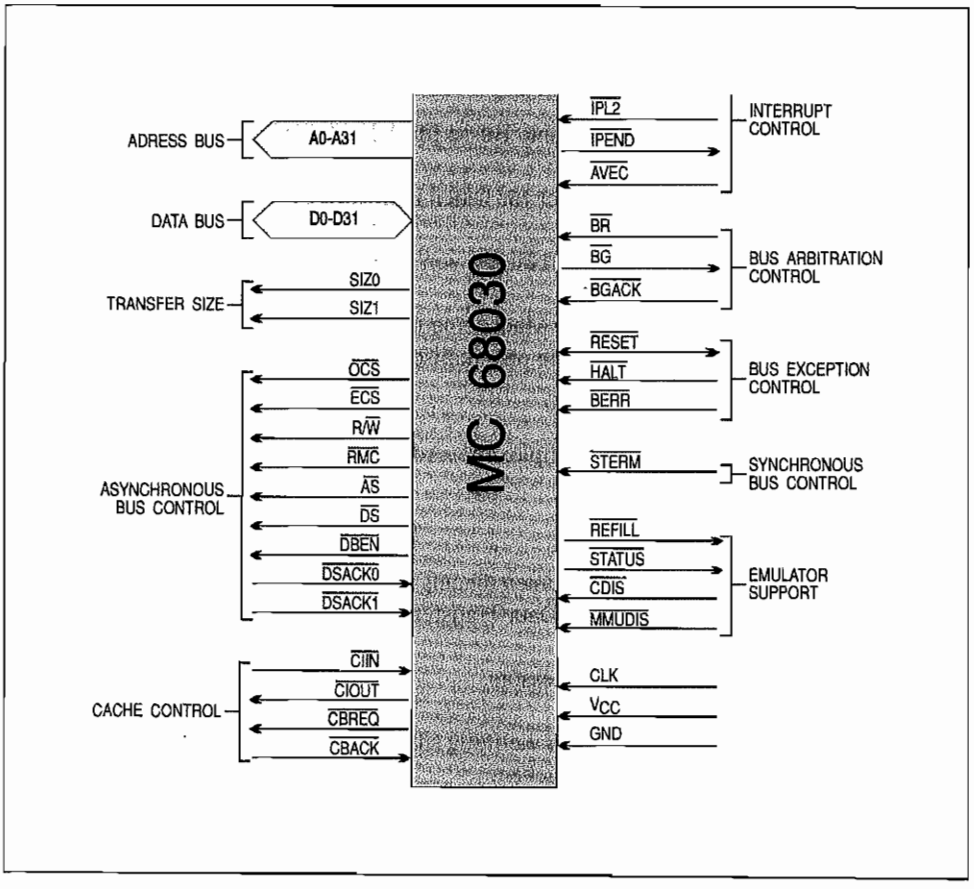


Abb. 1.1: Prozessorsignale, nach Funktionsgruppen geordnet

Hier nun eine kurze Beschreibung der Signale und deren Bedeutung bzw. Verwendung im ATARI TT (oder eben auch nicht!):

Function Codes (FC0 .. FC3) — Hiermit gibt die CPU bekannt, auf welchen Adreßraum (Supervisor Program/Data, User Program/Data usw.) im laufenden Buszyklus zugegriffen wird. Beim Functioncode $7 (alle Bits gesetzt) erfolgt ein Zugriff auf den sogenannten CPU address space (spezieller Adreßbereich der CPU). Das passiert unter anderem bei allen CPUs der MC680XX-Serie bei der Interruptbestätigung und bei den MC68020/030-CPUs während der Ausführung von Coprozessor-Operationen.

32 Bit-Adreßbus (A_0 .. A_{31}) — Ausgabe der Zugriffsadresse für den laufenden Buszyklus. Three-state-Ausgänge (Neben Low- und High kann der Anschluß auch noch Hochohmig geschaltet werden!).

32 Bit-Datenbus (D_0 .. D_{31}) — Transport der Daten mit 8, 16, 24 oder 32 Bit pro Buszyklus. Three-state, bidirektional!

Transfer Size (SIZ0, SIZ1) — Statussignal über die im laufenden Buszyklus auf dem Datenbus übertragenen Bytes (1 .. 4 Bytes!). Three-state-Ausgang.

Bus-Steuersignale

Operand Cycle Start (_OCS) — Low-aktives Ausgangssignal, mit dem der Beginn des ersten externen Buszyklusses (Zugriff auf RAM, ROM usw.) bei einem *Operanden*-Transfer angezeigt wird. Wird *nicht* im TT verwendet!

External Cycle Start (_ECS) — Hiermit wird der Beginn *jedes Buszyklusses* angezeigt. Low-aktiver Ausgang. Wird *nicht* im TT verwendet.

Read/Write (R/_W) — Art des Buszyklusses. High bei Read, Low bei Write-Zugriff. Three-state-Ausgang.

Read-Modify-Write Cycle (_RMC) — Identifiziert mit einem Low einen Buszyklus, der nicht unterteilbar ist. Es wird etwas in die CPU gelesen, dort modifiziert und wieder "weggeschrieben". Three-state-Ausgang. Wird *nicht* im TT verwendet.

Adress Strobe (_AS)

Auch schon vom MC68000 bekanntes, low-aktives Signal, mit dem die Gültigkeit der Adresse im laufenden Buszyklus angezeigt wird. Three-state-Ausgang.

Data Strobe (_DS)

Low-aktives Signal. Beim *Lese*zyklus wird damit signalisiert, daß die externe Einheit Daten auf den Datenbus zu legen hat. Während eines *Schreib*zyklusses zeigt die CPU mit einem Low die Gültigkeit der Daten auf dem Datenbus an. Wird *nicht* im TT verwendet.

Data Buffer Enable (_DBEN)

Low-aktives Ausgangssignal für Systeme mit externen (schnellen) Datenbuffern (so eine Art externe Cache!). Damit können diese Buffer freigegeben werden. Wird *nicht* im TT verwendet.

Data Transfer and Size Acknowledge (_DSACK0, _DSACK1)

Eingänge, Low-aktiv. Für asynchrone Buszyklen verwendet.

Bei einem *Lese*zyklus "sagt" die externe Einheit (z. B. I/O-Baustein) darüber Bescheid, wenn die gewünschten Daten auf dem Datenbus bereitstehen und in welchem Format (8-, 16- oder 32-Bit-Format) das der Fall ist.

Beim *Schreib*zyklus quittiert die ext. Einheit die Übernahme der Datenbits und in welchem Format das geschehen ist.

Bei z. B. einem Langwort-Lese-Zugriff (32 Bit) auf einen 8-Bit-Port wird der Port mit einer "Antwort" von _DSACK0 = Low und _DSACK1 = High jeden Zugriff quittieren.

Die CPU wird die ersten acht Bits einlesen und einen weiteren Zyklus starten, um von diesem Port die nächsten 8 Bits zu erhalten. Das geht dann so lange (vier Zyklen), bis alle vier Bytes in der CPU angekommen sind!

Synchronous Termination (_STERM)

Dieser Eingang wird als Ende-Signal (Synchronous Termination) für *synchrone* Buszugriffe benutzt.

*Lese*zyklus: Mit _STERM auf Low wird der CPU mitgeteilt, daß die Daten auf dem Datenbus mit der nächsten fallenden Taktflanke ge"lachted" (eingespeichert) werden können.

*Schreib*zyklus: Mit _STERM = Low signalisiert die externe Einheit die Übernahme der Daten vom Datenbus.

Cache-Steuersignale

Cache Inhibit Input (_CIIN)
: Mit einem Low an diesen Eingang wird verhindert, daß die Daten dieses Lesezyklusses in die internen Caches übernommen werden.

Cache Inhibit Output (_CIOUT)
: Dieses Low-aktive Ausgangssignal signalisiert an eventuelle ext. Caches, die Daten des laufenden Buszyklusses zu ignorieren. Wird *nicht* im TT verwendet.

Cache Burst Request (_CBREQ)
: Mit einem Low-Signal fordert die CPU das Auffüllen einer Zeile (vier Langworte) im Cache durch Burst-Mode an. Three-state-Ausgang.

Cache Burst Acknowledge (_CBACK)
: Die angesprochene Einheit (RAM) kann im Burst-Mode arbeiten und ist in der Lage, mindestens noch ein Langwort für die int. Caches zu liefern.

Steuersignale für Interrupts

Interrupt Priority Level (_IPL0 .. _IPL2)
: Low-aktive Eingänge, über die, wie beim MC68000, die "Dringlichkeit" des Interrupts signalisiert wird.

Interrupt Pending (_IPEND)
: Mit einem Low bestätigt die CPU, daß sie die Interruptanforderung intern registriert hat und dieser Interrupt eine höhere Priorität besitzt, als die gegenwärtige Interrupt-Maske aufweist.

  Bei der nächsten Gelegenheit (wenn intern der nächste Befehl an die Reihe kommt) wird dann der Interrupt bearbeitet. Wird *nicht* im TT verwendet.

Autovector (_AVEC) — Low-aktives Eingangssignal, daß die CPU bitte schön einen Autovektor zu erzeugen hat (beim Interrupt-Acknowledge-Zyklus).

Steuersignale für die Buszuteilung

Bus Request (_BR) — Ein externer Baustein (z. B. DMA) möchte gern den Bus "für sich" haben. Dieser Eingang ist Low-aktiv.

Bus Grant (_BG) — Low-aktiver Ausgang für die Signalisierung, daß der Prozessor die Buskontrolle nach Ende des laufenden Buszyklusses abgeben wird. (Bestätigung auf das Bus-Request-Signal.

Bus Grant Acknowledge (_BGACK)
Über diesen Eingang erfährt die CPU, daß der Bus jetzt von jemand anders übernommen wurde. Die neue Busmaster-Einheit beläßt dieses Signal auf Low, bis sie den Bus wieder freigibt.

Signale für die Bus-Ausnahmesteuerung

Reset (_RESET) — Bidirektionaler Anschluß!

Wenn von außen ein Low-Signal angelegt wird, geht der Prozessor in den Grundzustand zurück (Dauer des Signals mind. 520 Taktperioden). Wenn der Prozessor einen RESET-Befehl bearbeitet, bleibt der interne Status der CPU erhalten, und an diesem Anschluß wird ein Low-Impuls zum Rücksetzen der externen Einheiten ausgegeben.

Halt (_HALT) — Mit einem Low an diesem Anschluß wird die CPU "aufgefordert", alle Bus-Aktivitäten einzustellen. In Verbindung mit dem _BERR-Signal bedeutet das die Aufforderung, den letzten Buszyklus nochmal zu wiederholen.

Bus Error (_BERR) — Mit einem Low an diesem Eingang wird der Prozessor auf eine ungültige Busoperation hingewiesen (zum Beispiel, weil eine ext. Einheit innerhalb einer gewissen Zeit nicht auf einen Buszyklus reagiert hat).

Signale für Emulator-Unterstützung

Für manche Emulationen und zur Fehlersuche ist es zwingend notwendig zu wissen, wie weit die Abarbeitung von Befehlen *im* Prozessor gediehen ist (durch die Verwendung der Caches ist das nicht so einfach festzustellen). Deshalb sind einige Signale vorgesehen, die dabei "behilflich" sind.

Cache Disable (_CDIS)
: Mit einem Low an diesem Eingang wird der MC68030 "angewiesen", seine internen Caches *nicht* zu verwenden. Die Cacheinhalte werden dabei *nicht* gelöscht, sondern stehen nach Übergang des Signals auf High wieder unverändert zur Verfügung.

MMU Disable (_MMUDIS)
: Durch ein Low an diesem Eingang wird die interne MMU außer Funktion gesetzt. Dabei werden die Einträge im *Address-translation-cache* (*ATC* = Cachespeicher für die Adreßübersetzungstabelle) *nicht* gelöscht. Nach Entfernen des Low-Signals (also bei High) steht der MMU-Cache wieder unverändert bereit.

Pipeline Refill (_REFILL)
: Mit diesem Signal zeigt der Prozessor an, wenn die interne Befehlspipeline wieder neu gefüllt wird (bei Änderungen im Programmablauf zum Beispiel bei bedingten Verzweigungen). Three-state-Ausgang.

Internal Microsequencer Status (_STATUS)
: Mit einem Low unterschiedlicher Dauer signalisiert die CPU an diesem Anschluß, ob sie gerade am Ende einer Befehlsbearbeitung ist (Low für einen Taktzyklus), ob eine Trace- oder Interrupt-Ausnahmeverarbeitung ansteht (Low für zwei Taktzyklen) oder daß eine andere Ausnahmeverarbeitung begonnen wird (Low für drei Taktzyklen).

Nicht zu vergessen ist natürlich der Taktanschluß, der im TT mit 32 MHz und TTL-Pegel gespeist wird.

Der MC68030 im TT

Der TT ist von seinem Aufbau her (ROM, RAM usw.) auf 16 MHz ausgelegt. Atari hat jedoch schon kurz nach Auslieferung der ersten TT-Rechner mit 16 MHz an Softwarehäuser eine

Takterhöhung für die CPU auf 32 MHz vorgesehen. Damit laufen also prozessorinterne Vorgänge mit der doppelten Taktfrequenz ab. Bei RAM- und ROM-Zugriffen sind aufgrund des restlichen Systemdesigns jedoch entsprechende Waitstates beim Zugriff erforderlich.

Da das TT-Platinenlayout zunächst für eine 16-MHz-CPU vorgesehen war, ist für die Umrüstung auf 32 MHz von Atari eine eigene Aufsatzplatine zur Anwendung gekommen. Auf dieser Platine befinden sich sowohl die CPU als auch einige PALs und FlipFlops in SMD-Bauweise zur Anpassung an das "langsamere Umfeld". Über einen PGA-Stecker wird die Platine direkt in die Fassung für den MC68030 auf dem Motherboard gesteckt. Die Zuführung des 32-MHz-Taktes erfolgt über eine separate Taktleitung vom Taktoszillator auf der Hauptplatine. In Zukunft wird das Platinenlayout des Motherboards gleich für den 32-MHz-Betrieb des Prozessors und des Coprozessors ausgelegt sein. Die Anpassungsplatine entfällt dann.

Der interne Aufbau des MC68030

soll hier nur kurz durch eine Übersicht über die Register gestreift werden.

Im User-Modus stellt sich der MC68030 genau wie der "gute alte" MC68000 im ST(E) dar. Die gleichen acht Daten- und acht Adreßregister mit je 32 Bits Breite sind auch hier zu finden. Programmzähler und die unteren acht Bits des Statusregisters (das Condition Code-Register) sind genauso aufgebaut und zu betrachten wie beim MC68000!

Es sind jedoch eine Anzahl von neuen Adressierungsarten hinzugekommen, wie z. B. die Möglichkeit, sowohl Daten- als auch Adreßregister als Indexregister zu verwenden. In der Supervisor-Betriebsart kommen aber die zusätzlichen Register der CPU erst richtig zum Tragen.

Eine Besonderheit gegenüber dem MC68000 ist das *Vektor Base Register (VBR)*. Es enthält die Anfangsadresse der Tabelle für die Ausnahmevektoren (1 KByte lang). Zu dem Eintrag in diesem Basisregister wird dann der Versatz für den angesprochenen Ausnahmevektor dazuaddiert und so der Pointer auf die Adresse gebildet, in der die Anfangsadresse für die Ausnahmeverarbeitungs-Routine abgelegt ist. Damit muß also die Tabelle für die Ausnahmevektoren nicht mehr zwingend an der absoluten Adresse $8 beginnen wie beim MC68000 (was sie im TOS des TT aber weiterhin tut)!

Im Supervisor-Modus stehen nun zwei Stackpointer zur Verfügung. Im "normalen" Supervisor-Modus, wie er z. B. nach einem RESET eingestellt ist, wird der *Interrupt Stack Pointer (ISP)* verwendet. Umschalten auf den *Master Stack Pointer (MSP)* läßt sich im Supervisor-Modus durch Setzen des M-Bits im Status-Register. Die Belegung des Statusregisters ist nur im Statusbyte (Supervisor-Modus) geändert (erweitert) worden und sieht wie auf der übernächsten Seite aus.

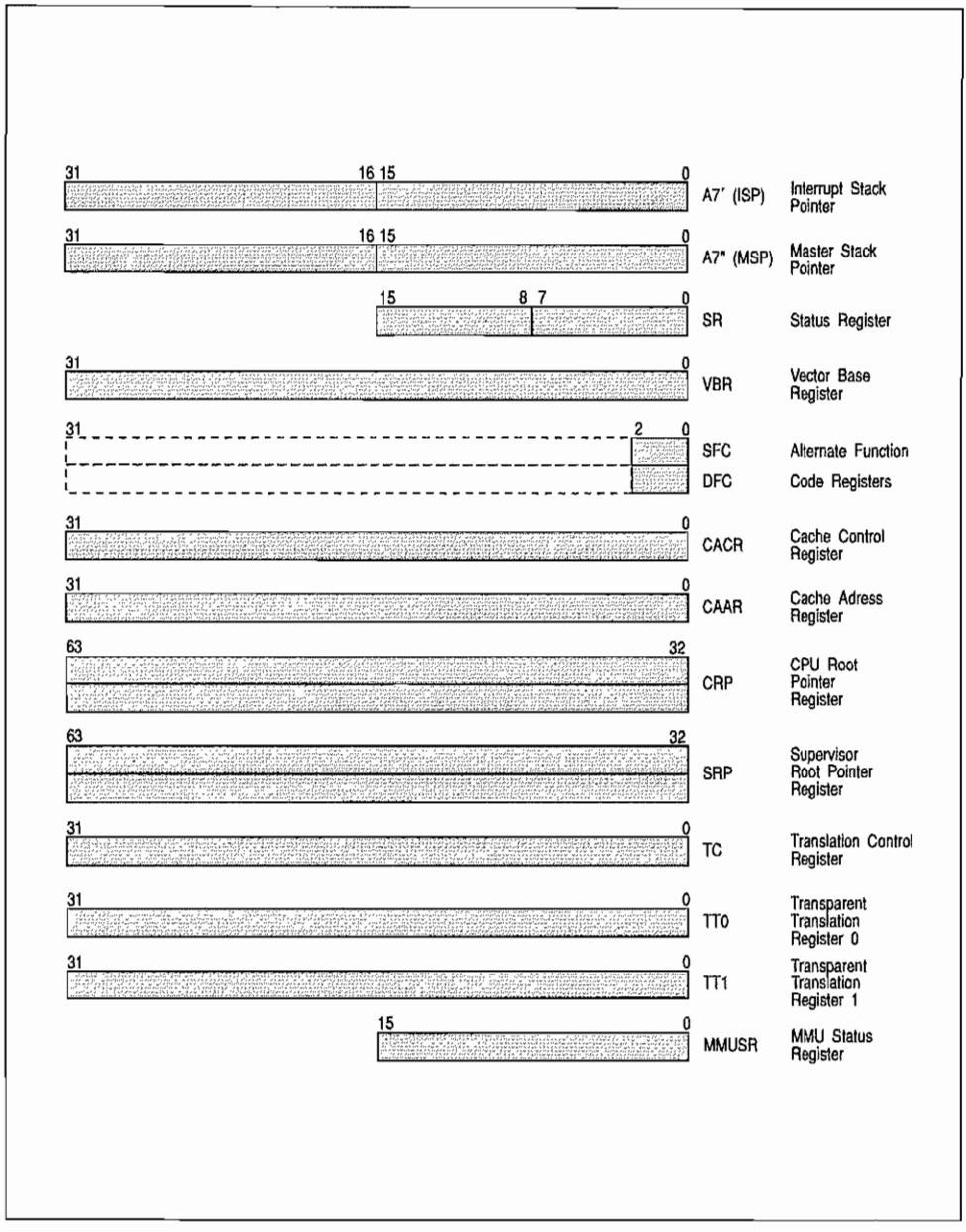
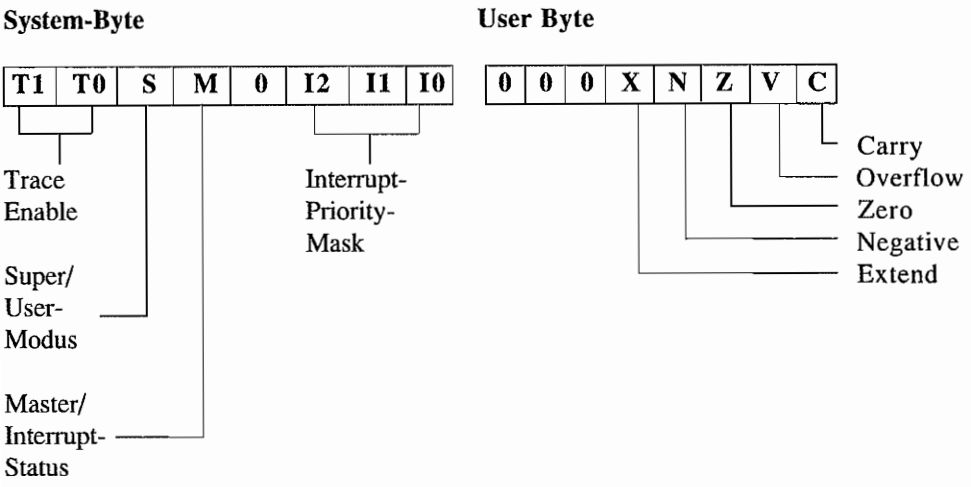


Abb. 1.2: Der Registersatz im Supervisor-Modus



Mit den Alternate-Function-Code Registern (SFC und DFC) kann unter bestimmten Bedingungen der durch den 32 Bit-Adreßbus auf vier GByte erweiterte Adreßraum nochmal vervielfacht werden. Mit den je drei Bit-Funktionscodes sind damit *acht* Adreßräume zu je vier GByte unterscheidbar (das sollte erst mal reichen!).

Geschwindigkeitsgewinn durch Caches

In den auf dem Chip integrierten Caches werden immer ein Teil der bereits verarbeiteten Daten und die im Moment in Bearbeitung befindlichen Befehle zwischengespeichert. Das hat den Vorteil, daß auf diese Informationen bei nochmaligem Bedarf (z. B. während einer Programmschleife) ohne einen externen Buszyklus zugegriffen werden kann! Das spart natürlich Zeit und erhöht die Geschwindigkeit nicht unerheblich (um ca. 10..30%). Für das Füllen der Caches wird vorzugsweise der sogenannte "Burst fill mode" verwendet. Damit wird jeweils immer ein ganzer Cacheeintrag von 128 Bit (vier Langworte) mit einer sehr geringen Zahl von Taktzyklen gefüllt (günstigstenfalls in *fünf* Taktzyklen, ohne Waitstates, sonst entsprechend länger)! Zu dieser Betriebsweise muß natürlich die Speicherhardware in der Lage sein (was im TT teilweise der Fall ist).

Caches können nach verschiedenen Prinzipien arbeiten. "Voll-assoziativ-Caches" vergleichen zur Überprüfung, ob z. B. der Befehl unter einer bestimmten Adresse im Cache vorliegt, die gesamte Adresse. Im MC68030 wird nach dem "Teil-assoziativ-Prinzip" gearbeitet. Dabei wird immer nur ein Teil der gewünschten Adresse verglichen. Es wird nämlich unter einem Cacheeintrag immer ein Block von vier Langwords verwaltet. Damit reduziert sich der erforderliche Hardwareaufwand in der Vergleicherlogik nicht unerheblich, ohne einen merklichen

Verlust in der Trefferquote und der Geschwindigkeit hinnehmen zu müssen. Der MC68030 verwendet aus diesem Grunde auch bevorzugt den bereits erwähnten Burst-fill-Modus, um so jeweils mit einem schnellen Zugriff auf den externen Speicher einen Cacheeintrag aufzufrischen.

Ein Cache ist also nur bei wiederholbaren Programmsequenzen effektiv. Dabei darf die Sequenzlänge aber nicht die Größe des Cache übersteigen, weil sonst die Cacheeinträge ständig überschrieben werden. Gleiches gilt für Sprünge außerhalb der Adreßdistanz des Cache. Ein paar Nachteile lassen sich dabei durch geschickten Umgang mit dem *Cache Control Register (CACR)* und dem *Cache Address Register (CAAR)* vermeiden.

0	0	WA	DBE	CD	CED	FD	ED

WA = Write Allocate
DBE = Data Burst Enable
CD = Clear Data Cache
CED = Clear Entry in Data Cache
FD = Freeze Data Cache
ED = Enable Data Cache

0	0	IBE	CI	CEI	FI	EI

IBE = Instruction Burst Enable
CI = Clear Instruction Cache
CEI = Clear Entry in Instruction Cache
FI = Freeze Instruction Cache
EF = Enable Instruction Cache

Das Cache-Control-Register
(Das High-Word des Registers ist Null!)

Mit dem Cache Control Register ist das Sperren (ED- bzw. EI-Bit), Einfrieren (FD- bzw. FI-Bit) und das Löschen des gesamten Caches (CD- bzw. CI-Bit) möglich. Unter Einfrieren versteht man, daß sich der Cache wie ein ROM verhält. Bei Treffern werden wohl die Cacheinformationen gelesen, aber nicht überschrieben.

Es können auch einzelne Cacheeinträge, die über das Cache Address Register (CAAR) bestimmt werden, durch Manipulation mit dem CED- bzw. CEI-Bit gezielt gelöscht werden. Durch die spezielle Organisation des Caches ist es lediglich erforderlich, die Bits 7..2 des Adreßeintrags in die Bits 7..2 des CAAR einzuschreiben.

Damit ist der entsprechende Cache-Eintrag zum Löschen identifiziert! Die anderen Bits im 32-Bit großen CAAR sind für zukünftige Verwendung durch Motorola reserviert (und werden zur Zeit noch nicht ausgewertet!).

Das DBE- bzw. IBE-Bit signalisiert der Bus-Control-Logik des Prozessors, ob der Burst-fill-Modus möglich ist.

Adressen gibt's, die gibt's gar nicht – Die PMMU

Hauptsächliches Einsatzgebiet für *Memory Management Units (MMU)* ist die Adreßumsetzung von logischen Adressen in real existierende, also physikalische Adressen, bei Multitasking-Systemen. Dabei kommt es halt vor, daß mehrere Prozesse (Tasks) parallel bearbeitet werden sollen. Da in Systemen mit nur einem Prozessor dieser sich immer nur zur gleichen Zeit um einen Prozeß kümmern kann, muß eine Umschaltung zwischen einzelnen Tasks z. B. zeitgesteuert erfolgen. Um die Umschaltzeiten gering zu halten, ist es natürlich von Vorteil, daß die Tasks sich im Arbeitsspeicher befinden. Das würde bedeuten, das diese Tasks alle an unterschiedlichen Adressen stehen müssen und Überlappungen ebenfalls nicht erlaubt sind.

Da diese Bedingungen sehr einschränkend sind, weicht man auf den Einsatz einer MMU zwischen Prozessor und Speicher aus. Diese sorgt dafür, daß unterschiedliche Tasks mit gleichen logischen Adressen arbeiten können, welche dann durch die MMU durch Umsetzung in entsprechende physikalische Adressen umgerechnet werden.

Eine weitere Einsatzmöglichkeit für MMUs sind virtuelle Speicherkonzepte. Damit sind Zugriffe auf Adressen möglich, die eigentlich gar nicht real (als Halbleiterspeicherstelle) existieren, sondern nur "scheinbar" (virtuell) vorhanden sind. Zugriffe auf solche virtuellen Adressen geschehen meistens auf Massenspeicher wie Harddisks.

Der Zugriff auf so eine virtuelle Adresse führt also zunächst zu einer Fehlersignalisierung, weil die Adresse real nicht existiert. Dann sorgt das Betriebssystem durch eine Ausnahmebehandlung dafür, daß die benötigten Daten dieser Speicherstellen vom Massenspeicher in einen freien Speicherbereich im RAM geladen werden. Die MMU sorgt dann nach entsprechender Programmierung dafür, daß der Zugriff auf die virtuelle Adresse auch auf die real existierende Speicherstelle im RAM umgerechnet wird!

Die im MC68030 integrierte MMU benutzt für die Adreßumsetzung Tabellen für die physikalischen Adressen, die im Speicher angelegt werden. Die logische Adresse dient dabei als Index in dieser Tabelle. Damit hier nicht jedesmal ein ext. Buszyklus erforderlich wird, ist zur Beschleunigung noch ein Cachespeicher hinzugefügt worden (der *Address Translation Cache, ATC*), in dem häufig benutzte Umsetzungen aufbewahrt werden (22 Einträge umfaßt der ATC!).

Erst wenn die Umsetzung mittels des ATC nicht vorgenommen werden kann, wird die im RAM liegende Tabelle benutzt. Außerdem existieren für Zugriffe auf Speicherblöcke, die keiner Umsetzung unterliegen sollen, zwei *Transparent Translation Register (TT0* und *TT1*).

Jede einzelne log. Adresse verbraucht für die Umsetzung in ihre physikalische Adresse natürlich nicht einen ganzen Eintrag.

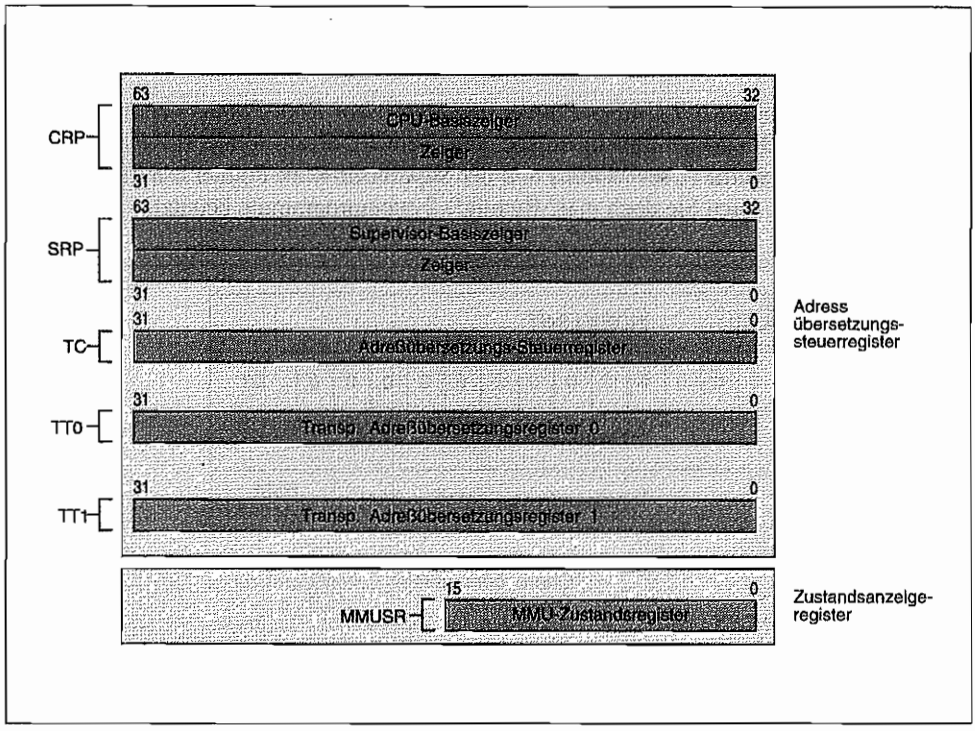


Abb. 1.3: Das Programmiermodell der MMU

Vielmehr wird der Speicherbereich der CPU in gleich große Blöcke (Pages) unterteilt (beim MC68030 können diese 256 Byte..32 KByte groß sein, im TT-TOS wird die PMMU auf 32 KByte-Pages eingestellt), und die Adreßumsetzung erfolgt dementsprechend blockweise. Deshalb wird die Memory Management Unit im MC68030 auch als *Paged Memory management Unit (PMMU)* bezeichnet. Jede Page von log. Adressen wird also in einen gleich großen Block von physikalischen Adressen umgerechnet. Das Ganze läßt sich durch Gliederung in mehrere Umsetzungsebenen (maximal fünf sind möglich) sehr flexibel handhaben und spart außerdem drastisch Speicherplatz für die Umsetzungstabellen ein. Am Ende der "Pointerei" durch die einzelnen Ebenen findet die MMU dann endlich die Adresse der gewünschten Page (als Page-Deskriptor bezeichnet).

So existieren denn zwei grundsätzliche Typen von Datenstrukturen, sogenannte Deskriptoren. Der *Page-Deskriptor* enthält die gewünschte physikalische Umsetzungsadresse (bzw. die oberen 24 Bits davon, die unteren 8 Bits werden von der logischen Adresse genommen!) und befindet sich am Ende einer Umsetzungstabelle! Als *Tabellen-Deskriptor* wird ein Eintrag

bezeichnet, der in den oberen 24 Bits die Adreßbits für den Pointer auf eine weitere Deskriptor-Tabelle enthält.

Die beiden Root-Pointer (*CPU-Root-Pointer und Supervisor-Root-Pointer*) enthalten dabei den Zeiger auf den Beginn der Tabelle im Speicher. Der CPU-Root-Pointer (*CRP*) wird dabei für Anwenderprogramme (User-Modus) und der Supervisor-Root-Pointer (*SRP*) im Supervisormodus für (na, raten Sie doch mal? Richtig!) Systemprogramme benutzt.

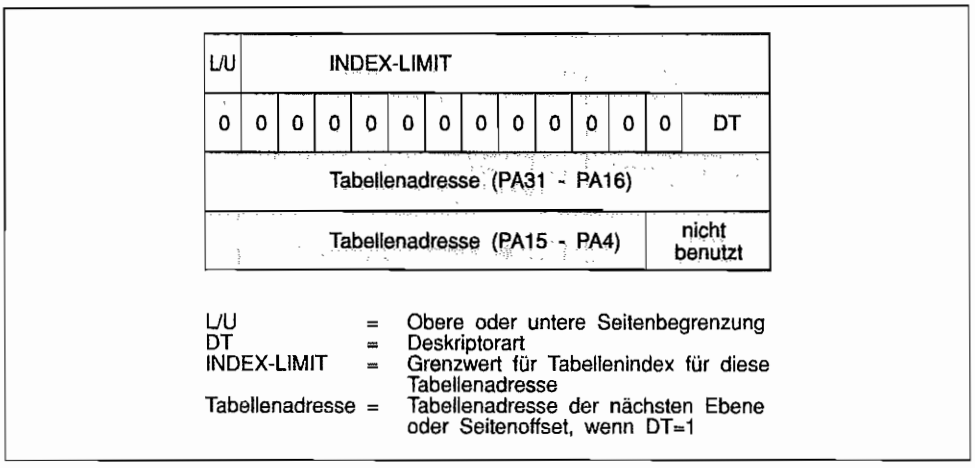


Abb. 1.4: Der Aufbau des Root-Pointers

Daß diese beiden Pointer 64 Bit groß sind, liegt daran, daß außer dem Pointer auf den Tabellenbeginn im Speicher auch noch weitere Informationen enthalten sind. Die eigentliche Tabellenadresse umfaßt dabei sogar nur 28 Bits, womit sich die Einschränkung ergibt, daß diese Tabelle an einer durch 16 teilbaren Adresse beginnen muß.

Die Bits 32..33 bilden den sogenannten *Descriptor type (DT)*. Ein Wert von $1 deutet dabei auf eine Tabelle im Speicher hin, die mit Page-Deskriptoren gefüllt ist. Der Wert von $0 ist beim Root-Pointer nicht zulässig; er markiert einen Deskriptor im Speicher als INVALID (führt dazu, daß die weitere Tabellensucherei an der Stelle abbricht!).

Die anderen möglichen Werte für DT deuten darauf hin, daß der Tabellenbereich, auf den verwiesen wird, im Short-format (vier Bytes/Eintrag bei DT = $2) oder im Long-format (acht Bytes/Eintrag bei DT = $3) vorliegt. Das höchstwertige Bit (*L/U*-Bit) im Rootpointer (wie auch bei einem Long-format-Tabelleneintrag) gibt dann an, ob durch das 15 Bit-Feld in

den Bits 48..62 das obere (L/U = 0) oder untere (L/U = 1) *LIMIT* des Indexes in der Tabelle angegeben ist. Was bedeutet, daß z. B. bei L/U = 0 der Index in die folgende Tabelle den LIMIT-Wert nicht überschreiten darf.

In den Deskriptoren, die in der Tabelle im Speicher stehen, existieren je nach Art (wie z. B. Short- oder Long-format-Tabellen- bzw. Page-Deskriptoren) noch weitere Steuer- bzw. Status-Bits am Ende des oberen Deskriptor-Langworts bzw. am Ende des Short-format-Deskriptors. Das *U*-Bit (*U*sed-Bit) wird von der MMU gesetzt, wenn der zugehörige Deskriptor für eine Adreßumrechnung gebraucht wurde.

Mit gesetztem *WP*-Bit (*W*rite *P*rotect-Bit) wird signalisiert, daß auf die letztlich gefundene physikalische Adresse nicht geschrieben werden darf. Das gesetzte *CI*-Bit (*C*ache *I*nhibit-Bit) sorgt dafür, daß bei Zugriff auf diese Page die Umsetzung nicht in den ATC übernommen wird (z. B. weil sich dahinter Hardware-Adressen verbergen, die sich ohne Beeinflussung der CPU ändern können). Mit einem gesetzten *M*-Bit (*M*odified Bit) gekennzeichnete Page-Deskriptoren signalisieren, daß durch einen Schreibzugriff Speicherstellen in dieser Page verändert wurden.

Das *Translation Control Register (TC)* besitzt mehrere Bitfelder, die z. B. für das Umschalten zwischen CRP und SRP, Größeneinstellung der Pages, Festlegung der Anzahl der Bits, die für die Adreßumsetzung benutzt werden sollen, und die Struktur der Umsetzungstabelle verwendet werden.

Die Register *TT0* und *TT1* legen "Fenster" einstellbarer Größe im Adreßraum fest, für die keinerlei Adreßumsetzung stattfinden soll. Dabei kann außerdem festgelegt werden, ob jedes dieser Fenster nur für lesende oder schreibende oder beide Betriebsweisen gelten soll.

Im *Memory Management Status Register (MMUSR)* schließlich findet man Statusinformationen über die Suche aus Anlaß einer Adreßübersetzung im ATC oder dem Umsetzungsbaum.

Auf weitere Einzelheiten zur MMU einzugehen wäre zu umfangreich. Dafür empfiehlt sich das bereits weiter oben erwähnte "MC68030 Enhanced 32 Bit Mikroprocessor User's Manual", wo auf nahezu 100 Seiten die MMU durchleuchtet wird.

Im TT-TOS beschränkt sich der Einsatz der MMU ohnehin eigentlich nur auf das Simulieren des ST-Adreßraums. Das läuft dann darauf hinaus, dem MC68030 einen mehr oder weniger zum ST kompatiblen Speicherraum vom 16 MByte vorzutäuschen. So sollen sich eben z. B. in den letzten 32 KByte des ST-Adreßraums auch beim TT die ST-Hardwareregister wiederfinden, die aber physikalisch eigentlich in den letzten 32 KByte des MC68030-Speicherraums liegen (und auch dort nochmals auftauchen!).

Kapitel 2: Damit's noch schneller geht – Der Coprozessor MC68882

Im ATARI TT verrichtet neben der MC68030-CPU noch ein weiterer Prozessorbaustein seinen Dienst. Er besitzt einen Befehlssatz von "nur" 64 Instruktionen, dafür sind diese aber sehr hilfreich, wenn es um schnelle und komplexe Berechnungen geht. Der Coprozessor wickelt mathematische Aufgabenstellungen wie z. B. sin(x), log(x) usw. mit einer sehr hohen Verarbeitungsgeschwindigkeit *und* Genauigkeit ab.

Der Vorteil bei der Verwendung des MC68882 (oder seines Vorgängers, des MC68881) in Verbindung mit dem MC68020/MC68030 liegt darin, daß man beim Programmieren in der Praxis gar nicht mehr zu unterscheiden braucht, ob der entsprechende Befehl nun durch die CPU oder den Coprozessor ausgeführt wird. Die beiden Chips spielen so elegant ineinander, daß man meint, nur einen einzigen Prozessor zu programmieren, der auch Fließkommaarithmetik beherrscht.

Motorola hat dabei eine Schnittstelle für die Coprozessor-Anbindung definiert, die ab den Prozessoren der MC68020er-Serie die Kommunikation mit dem Coprozessor regelt.

Im Gegensatz zum MEGA ST(E) mit nachgerüstetem Coprozessor, braucht der Programmierer die Coprozessor-Interface-Register nicht selbst zu bedienen, um festzustellen, wie weit der Coprozessor ist und ob z. B. noch weitere Operanden "rübergeschoben" werden müssen.

Hardwaremäßige Einbindung des MC68882 beim TT

Die Anbindung des Coprozessors an einen MC68030 ist relativ einfach zu bewerkstelligen. So sind natürlich alle 32 Datenbusleitungen zwischen beiden Chips vorhanden, um nicht durch einen zu "schmalen" Bus einen Performance-Verlust zu erzeugen. Für die Registerauswahl im Coprozessor sind außerdem die Adreßbusleitungen A1.. A4 an die *FPU* (= *Floating Point Unit*) angeschlossen. Zur Steuerung des Datenaustauschs werden dann nur noch die Quittungsleitungen _DSACK0 und _DSACK1 von der FPU bedient.

Über die _AS- und _DS-Eingänge erfährt die FPU von der CPU, wann Adreß- und Datenbusinformationen gültig sind. Die Art des Buszyklusses (Lesen oder Schreiben) wird über die R/_W-Leitung signalisiert. Und über den _CS (= Chip Select)-Anschluß teilt die TTSCU (TT-System Control Unit = Baustein für die Generierung und Überwachung von Signalen für die Ausnahmeverarbeitung) der FPU mit, wann sie denn angesprochen ist.

Weil die FPU die Buszyklen asynchron abschließt, muß diese nicht unbedingt mit dem gleichen Takt wie die CPU betrieben werden.

So sieht denn das (für die 32 MHz-CPU angepaßte) Platinenlayout im TT die Möglichkeit des Umschaltens auf 16 MHz FPU-Takt durch Umsetzen einer Lötbrücke (W101 auf 2 "–" 3 = 16 MHz-Takt; W101 auf 1 "–" 2 = 32 MHz-Takt) vor.

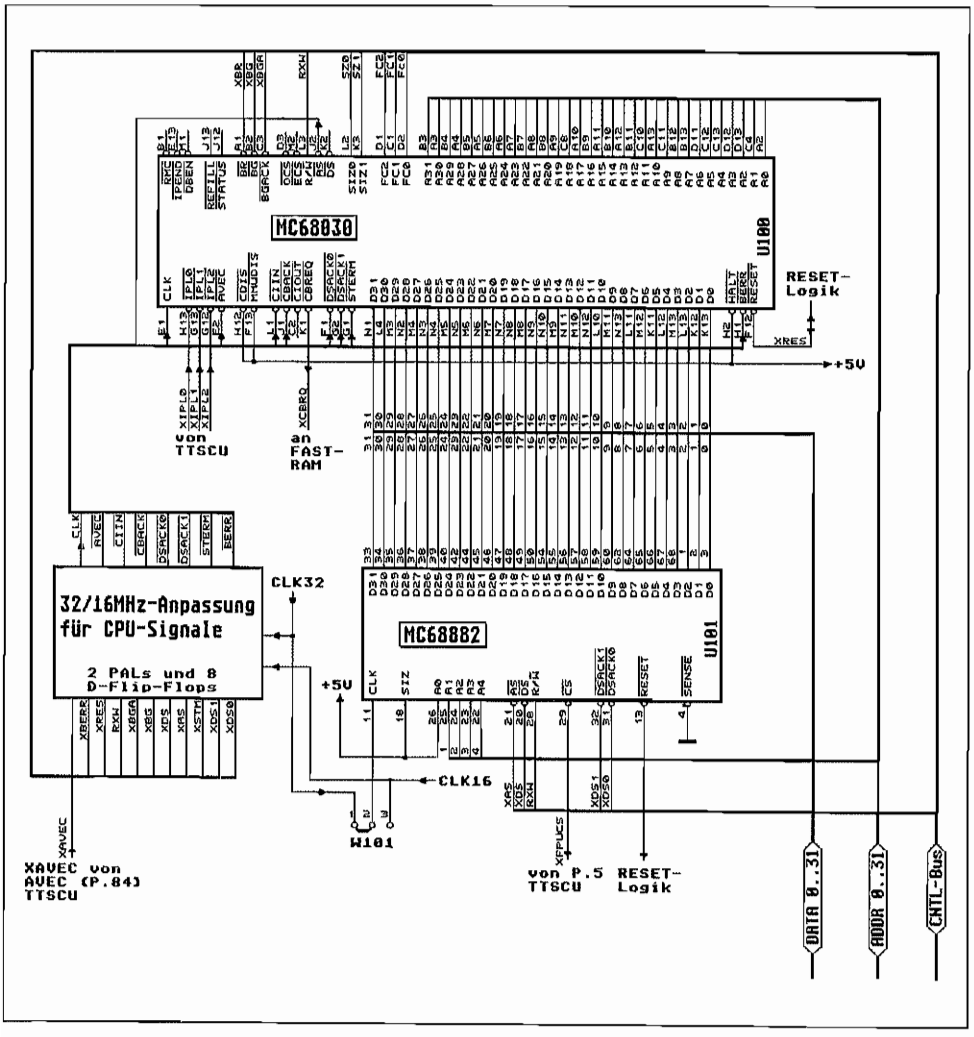


Abb. 2.1: Hardwaremäßige Einbindung des MC68882 beim TT

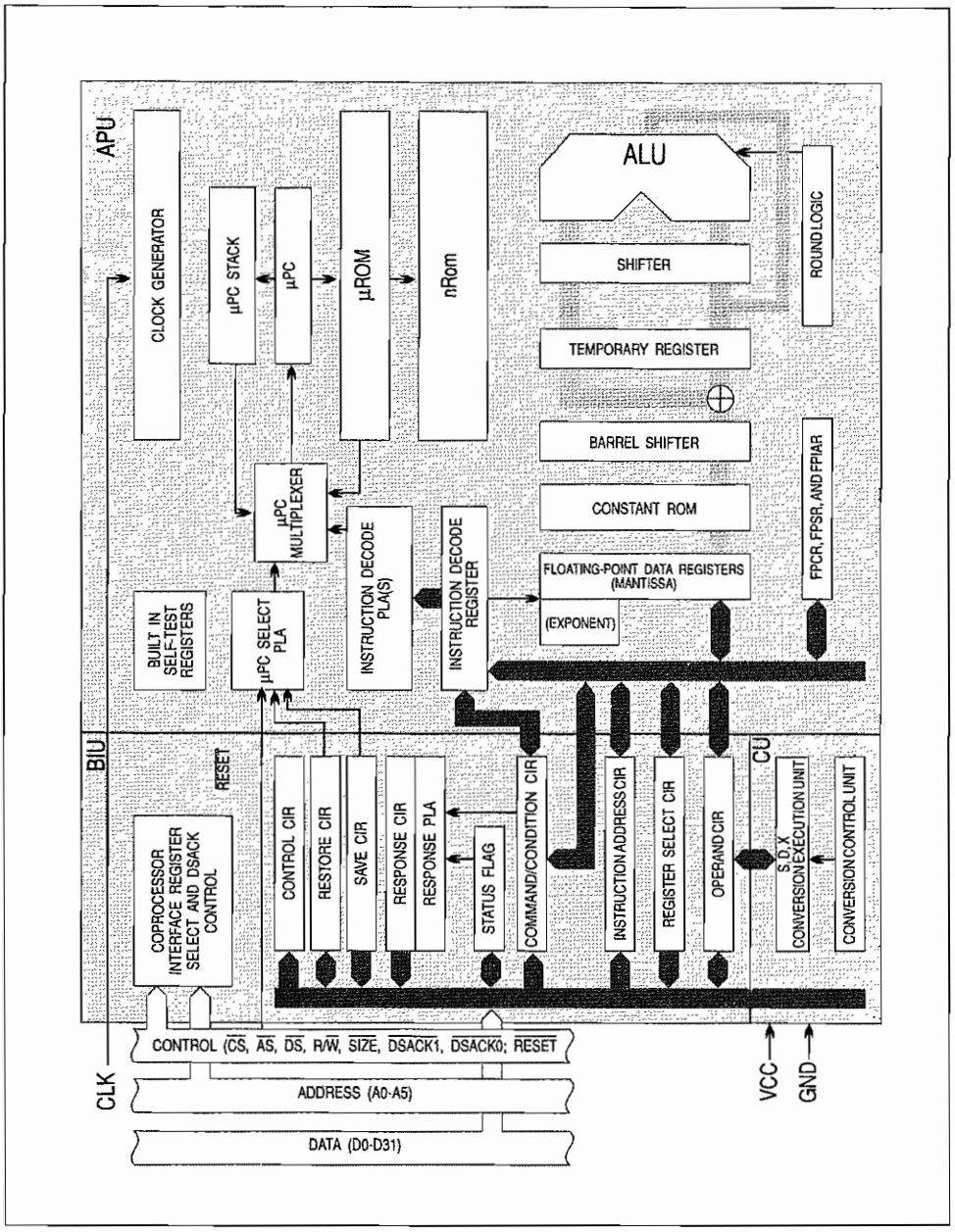


Abb. 2.2: Blockdiagramm des MC68882

Aufbau des MC68882

Der innere Aufbau des MC68882 ist ähnlich komplex wie der des MC68030. Für den Programmierer sind jedoch ohnehin nur die Register von Bedeutung und welche Funktionen darüber gesteuert werden können.

Die MC68882-FPU unterscheidet sich, außer in der Höhe der Taktfrequenz (der MC68881 ist in Ausführungen bis max. 25 MHz erhältlich, der MC68882 bis 33 MHz), der optimierten Arbeitsgeschwindigkeit, noch durch die *Conversion Unit (CU)*. Diese erlaubt bereits die Eingabe von weiteren Instruktionen, während die interne *Arithmetic Processing Unit (APU)* noch mit der vorherigen Aufgabe beschäftigt ist.

Von dieser Conversion Unit werden dann z. B. schon die weiteren Operanden angefordert (über die Bus Interface Unit der FPU) und vom platzsparenden 64 Bit-Format ins FPU-interne 80 Bit-Format konvertiert. Handelt es sich z. B. um Datentransportbefehle *in* eines der acht Floating-Point-Register, so wird der konvertierte Operand dort auch schon abgelegt (wenn die APU dieses Register nicht gerade selbst verwendet). Bei einem Datentransportbefehl *von* einem der Floating-Point-Register nach "außen" holt die CU den Operanden aus dem entsprechenden Register (wenn dieses Register nicht ausgerechnet Ziel der Rechenoperation des gerade laufenden Befehls in der APU ist), konvertiert diesen ins gewünschte externe Format und bringt ihn "ans Ziel" (natürlich mit CPU-Unterstützung!).

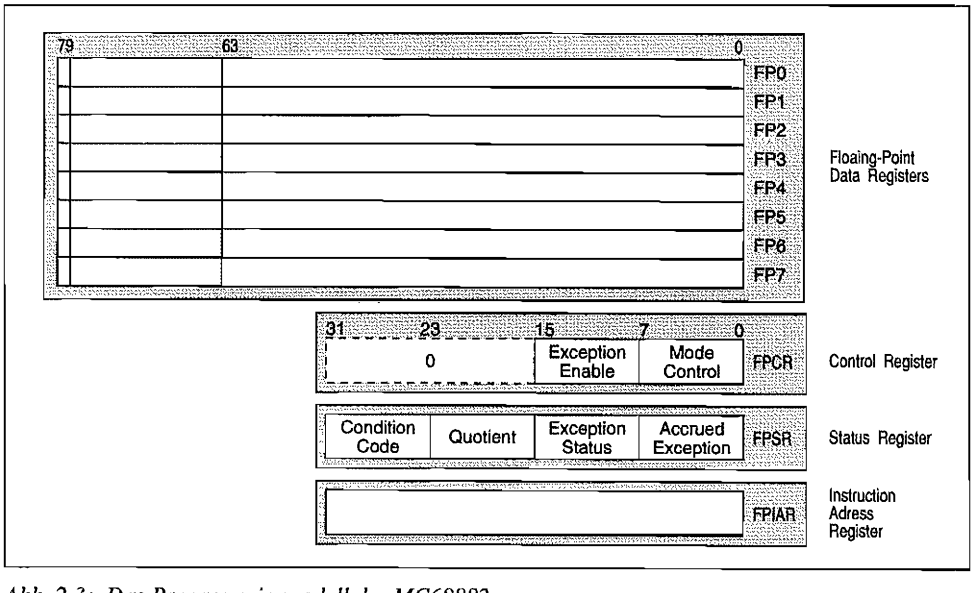


Abb. 2.3: Das Programmiermodell des MC68882

Zahlenformate beim MC68881/68882

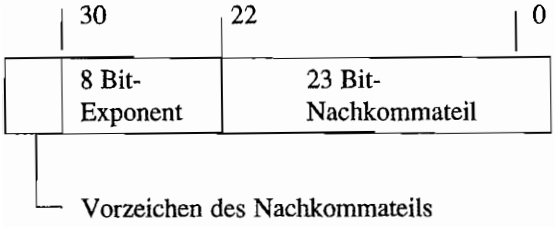
Die acht Fließkommaregister des MC68881/68882 besitzen je eine Breite von 80 Bit. Damit werden Fließkommazahlen als 64 Bit-Mantisse (Bits 0 .. 63) mit einem 15 Bit-Exponent (Bits 64..78) dargestellt. Das höchstwertige Bit dient als Vorzeichenbit für die Mantisse! Die Zahlen können jedoch auch in anderen Formaten dargestellt werden, die dann von der FPU entsprechend in das interne Format konvertiert werden. Da ja intern von der FPU ohnehin das 80-Bit-Format verwendet wird (auch als Extended Precision Format bezeichnet), können natürlich auch Operanden unterschiedlichen Formats verknüpft werden! Die MC68882-FPU unterstützt dabei die folgenden (auch im IEEE Standard for Binary Floating-Point Arithmetic vereinbarten) Formate:

Integer Data Format (Ganzzahl-Format)

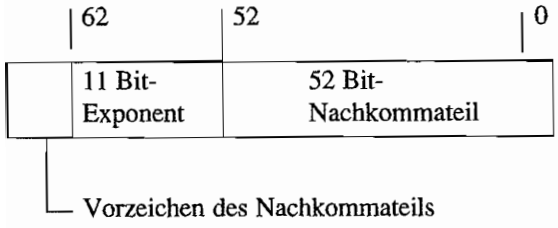
Hier sind die bereits vom MC680XX bekannten Datenformate in Zweierkomplement-Darstellung (höchstwertiges Bit ist Vorzeichenbit) von Byte-, Word- und Longword-Größe gemeint.

Binary Real Data Format (Binäres Realzahl-Format)

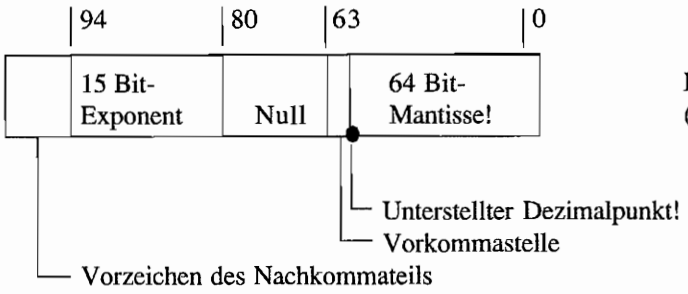
Generell wird hier mit dem folgenden Format gearbeitet: Vorzeichen x Mantisse x 2^{Exponent}



Single Real
(Einfache Genauigkeit)



Double Real
(Doppelte Genauigkeit)



Extended Real
(Doppelte Genauigkeit)

In allen drei Formaten wird die sogenannte "normalisierte" Darstellung verwendet. Das bedeutet, daß die Mantisse immer zwischen 1,0 und 2,0 liegt. Damit kann man sich die Eins vor dem Komma schenken (und muß sie sich nur denken).

Es wird also nur der Nachkommateil angegeben. Ausnahme ist hierbei das Extended Real-Format, bei dem die Vorkommastelle mit angegeben wird.

Es existieren auch sogenannte "nicht normalisierte" Zahlendarstellungen. Sie werden für Werte nahe dem untersten darstellbaren Zahlenwert verwendet. Dabei ist der Exponent = 0, und die Mantisse hat einen von Null verschiedenen Wert. Die Vorkommastelle ist dann aber ebenfalls = 0 (oder man muß sie sich beim Single Real- und Double Real-Format als Null "denken")!

Eine weitere Besonderheit ist der Zahlenwert Null. Er wird durch eine Null im Exponenten *und* in der Mantisse (bzw. Nachkommateil der Mantisse) dargestellt.

Eine Zahl, die den darstellbaren Wertebereich überschreitet, wird als Unendlich (Infinity) behandelt und durch eine Mantisse (bzw. Nachkommateil) von Null und durch den höchstmöglichen Wert für den Exponenten gekennzeichnet.

Ist der Exponent auf Maximum gesetzt *und* die Mantisse *nicht* Null, so wird damit ein Resultat angezeigt, das mathematisch unzulässig (nicht interpretierbar) ist. Z. B. das Ergebnis aus einer Division von Unendlich durch Unendlich! Solche Resultate werden als *NAN*s bezeichnet (Not-A-Number!)

Der Exponent ist bei allen drei Formaten zunächst als vorzeichenlose Ganzzahl zu betrachten, zu der allerdings bereits ein Offset addiert ist. Dieser beträgt 127 bei Single Real, 1023 bei Double Real und 16383 bei Extended Real.

Um den Zahlenwert des Exponenten zu erhalten, ist also der entsprechende Offset abzuziehen. Der so erhaltene Wert ist dann als Binärzahl im Zweierkomplement zu betrachten.

Beispiel:

Die Dezimalzahl 5 wird als Single Real-Zahl durch einen 32 Bit Long mit dem Wert $40A00000 dargestellt. Nach Zerlegung in Exponent und Nachkommateil erhält man dann:

Bits 0..22 bilden den Nachkommateil (20000_{hex}) der Mantisse. Wandelt man diese Binärzahl (als Nachkommazahl!) in Dezimal um, so erhält man $\mathbf{0{,}25}_{dez}$! Da ja die Eins vor dem Komma nicht auftaucht, aber "gedacht" werden muß (sie ist "implizit" vorhanden!), ergibt sich die komplette Mantisse dann zu $1_{dez} + 0{,}25_{dez} = \underline{\mathbf{1{,}25}}_{dez}$!

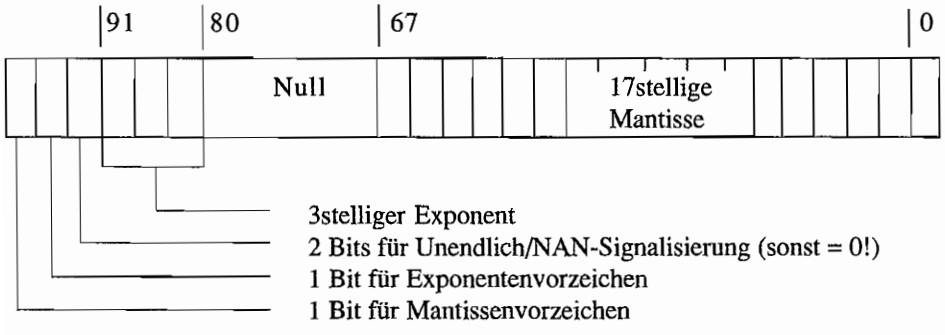
Bits 23..30 bilden den Exponenten ($10000001_{binär}$). In Dezimal sind das 129_{dez}. Da aber darin ein Offset von 127_{dez} (bei Single Real) enthalten ist, muß dieser abgezogen werden, um den Exponenten im Zweierkomplement zu erhalten. Somit ergibt sich als Resultat für den Exponenten $129_{dez} - 127_{dez} = \mathbf{2}_{dez}$.

Bit 31 ist das Vorzeichenbit des Nachkommateils und hier Null. Damit ist die Mantisse als positive Zahl deklariert.

Die Zahl läßt sich jetzt also als **Vorzeichen** x **Mantisse** x 2 ^{Exponent} schreiben. Das ergibt für das Beispiel den Wert von $+\mathbf{1{,}25} \times 2^2 = +1{,}25 \times 4 = +\underline{\mathbf{5}}$! (WZBW = **W**as **Z**u **B**eweisen **W**ar). Die gleiche Zahl als Double Real ergibt sich übrigens zu $40140000 00000000_{hex}$. Sie können ja selbst einmal probieren, ob die Umwandlung nach dem Double Real-Schema auch auf die dezimale 5 führt (wehe, wenn nicht!).

Packed Decimal Real Data Format (Dezimal Gepacktes Realzahl-Format)

Die Zahlen werden hierbei als 24stelliger, gepackter String dargestellt. In ein Byte werden jeweils zwei Ziffern "gepackt"! Somit werden zwölf Bytes für die gesamte Zahl benötigt.



Die Befehle des MC68881/68882

Die Kommunikation der CPU mit dem Coprozessor erfolgt nach einem durch Motorola festgelegten Protokoll. Dieses Protokoll ist aber im "Befehlssatz" der Prozessoren des Typs MC68000/68010 noch nicht vorhanden, sondern steht erst ab CPUs der Baureihe MC68020 und höher zur Verfügung. Wenn man die FPU in Verbindung mit einer MC68000-CPU (z. B. im ST(E)) betreiben will, so muß der Coprozessor als "normaler" Peripheriebaustein angesprochen werden. Für die Abwicklung des Protokolls, bei dem ja eine Menge von Informationen ausgetauscht werden müssen, stehen 12 FPU-interne Register zur Verfügung. Das Setzen und Auswerten dieser *Coprocessor Interface Register (CIR)* muß bei einer MC68000er-CPU durch ein Programm erfolgen, welches das Kommunikationsprotokoll einhält. Der Programmierer eines MC68020/68030 überläßt diese Arbeit seiner CPU.

Alle Coprozessorbefehle bestehen aus mindestens zwei Befehlswörtern. Beim ersten Befehlswort sind die vier höchstwertigen Bits immer gesetzt, um zu signalisieren, daß es sich um einen Coprozessorbefehl handelt. Weil alle Coprozessorbefehle also immer mit $F... beginnen, werden diese auch als Line-F-Befehle bezeichnet.

Bei MC68000/68010-CPUs lösen solche Befehle infolge des fehlenden (integrierten) Kommunikationsprotokolls einen Line-F-Trap aus. Dieser Trap soll dann dazu verwendet werden, um mittels eines entsprechenden Trap-Handlers die Coprozessorkommunikation zu steuern. Leider hat ATARI in früheren TOS-Versionen diesen Line-F-Trap für eigene Betriebssystemaufrufe benutzt! Da blieb einem dann nur noch die Möglichkeit, den nachträglich nachrüstbaren Coprozessor "von Hand" über eigene Routinen oder einen "verbogenen" Line-F-Handler zu steuern. Letztere Lösung wurde in der c't 4/1990 recht ausführlich beschrieben!

Aber nun zu den eigentlichen Coprozessor-Befehlen. Diese lassen sich in fünf Gruppen einteilen.

- Datentransportbefehle
- Befehle mit einem Operanden
- Befehle mit zwei Operanden
- Programmsteuerbefehle
- Befehle zur Systemsteuerung

Datentransportbefehle

Mit dieser Befehlsgruppe werden Daten bewegt, also Register in der FPU geladen oder Registerinhalte der FPU im Speicher oder der CPU abgelegt. Alle Transportbefehle beginnen mit dem Mnemonic *FMOVE*.

Weil ja bei Verwendung der FPU noch einige weitere Datenformate hinzukommen, müssen diese natürlich entsprechend im Befehl angegeben werden. Die bereits bekannten Datenformate Byte, Word und Long werden wie gehabt durch die Zusätze ".B", ".W" und ".L" hinter dem Befehlsmnemonic gekennzeichnet. Mit ".S" erkennt der Assembler ein Single Real-Format; ".D" steht für Double Real und ".X" für das Extended Real-Format.

Packed Dezimal wird durch ein ".P" gekennzeichnet. Außerdem ist hinter der Zieladresse noch ein Formatparameter (k-Faktor) im Zweierkomplement (zwischen –64 und +17) anzugeben. Positive Werte bestimmen dabei die Anzahl der Stellen, mit der die Mantisse ausgegeben wird. Negative k-Werte bestimmen die Anzahl der Nachkommastellen im "normalen" Zahlenformat (also wenn die Zahl nicht in Exponentialschreibweise dargestellt wird).

Beispiele:

```
FMOVE.L     d0,FP0          ; Long aus CPU-Reg. D0 ins FP0-Reg. laden.

FMOVE.X     FP0,(a0)+       ; Wert aus FP0-Reg. im Extended Real-Format im Speicher ablegen. Pointer steht in CPU-Reg. A0. Anschließend Pointer entsprechend inkrementieren.

FMOVE.P     FP7,result{#5}  ; FP7-Wert als Packed Decimal in Speicherstelle "result" ablegen. Dabei den Nachkommateil mit 5 Stellen angeben. Enthält FP7 also den Wert 12345,67898765, so ergibt sich als Ergebnisstring in "result" dann +1,2346 x 10^+4.
```

Genau wie bei der CPU gibt es auch bei der FPU die Möglichkeit, mit einem FMOVE-Befehl eine ganze Liste von Registerinhalten zu bewegen. Dieser Befehl heißt hier *FMOVEM* und transportiert entweder FP-Datenregisterinhalte im Extended Real-Format (FMOVEM.X) oder FPU-Control-Register als Long (FMOVEM.L) zwischen Speicher und FPU.

Weil in mathematischen Berechnungen bestimmte Konstanten (Kreiszahl π, Eulersche Zahl e usw.) relativ häufig benötigt werden, hat man diese in ein spezielles Konstanten-ROM der FPU "eingebaut". Mit einem speziellen *FMOVECR.X*-Befehl kann man durch Angabe des Offsets in dieses ROM die dort abgelegten Konstanten abrufen. Hier die Offsets der Konstanten in einer kleinen Tabelle:

Offset	**Konstante**	
$00	Kreiszahl π	= 3,14159...
$0B	$Log_{10}(2)$	= 0,30102...

Offset	Konstante	
$0C	Eulersche Zahl e	= 2,71828...
$0D	$Log_2(e)$	= 1,44269...
$0E	$Log_{10}(e)$	= 0,43429...
$0F	0,0	= 0,0!
$30	ln(2)	= 0,69314...
$31	ln(10)	= 2,30258...
$32	10^0	
$33	10^1	
$34	10^2	
$35	10^4	
$36	10^8	
$37	10^{16}	
$38	10^{32}	
$39	10^{64}	
$3A	10^{128}	
$3B	10^{256}	
$3C	10^{512}	
$3D	10^{1024}	
$3E	10^{2048}	
$3F	10^{4096}	

Befehle mit einem Operanden

Hierbei wird nur ein Eingabe-Operand benötigt, auf den der entsprechende Befehl angewendet wird. Dabei ist es aber durchaus möglich, das Ergebnis in einem anderen FP-Register abzulegen (Beispiel: "FSIN FP0,FP1" berechnet den Sinus des Werts in Register FP0 und setzt das Ergebnis dann in Register FP1. Register FP0 wird *nicht* verändert)!

Eine Besonderheit stellt hierbei allerdings der spezielle Befehl FSINCOS dar. Damit läßt sich von einer Zahl gleichzeitig der Sinus *und* der Kosinus berechnen.

Dazu ist allerdings erforderlich, daß *zwei* Zielregister für die beiden Rechenergebnisse angegeben werden.

Beispiel:

```
FSINCOS.X  FP0,FP1:FP2      ; Sin(FP0) -> FP2!; Cos(FP0) -> FP1!
```

Hier eine Zusammenfassung der möglichen Befehle in einer kleinen Tabelle:

Mnemonic	Funktion
FABS	Erzeugt den Absolutwert der Zahl (immer positiv!)
FACOS	Arcus Cosinus (Umkehrfunktion der Kosinusfunktion)
FASIN	Arcus Sinus (Umkehrfunktion der Sinusfunktion)
FATAN	Arcus Tangens (Umkehrfunktion der Tangensfunktion)
FATANH	Area Tangenshyperbolicus (Umkehrfkt. des Hyperbeltan.)
FCOS	Kosinus(X)
FCOSH	Hyperbelkosinus(X)
FETOX	e^X
FETOXM1	$e^X - 1$
FGETEXP	Liefert den Exponenten der Zahl
FGETMAN	Liefert die Mantisse der Zahl
FINT	Rundet auf die nächste ganze Zahl auf oder ab
FINTRZ	Rundet auf die nächste ganze Zahl ab
FLOGN	ln(x)
FLOGNP1	ln(x+1)
FLOG10	$\log_{10}(X)$
FLOG2	$\log_2(X)$
FNEG	Negation
FSIN	Sinus(X)
FSINH	Hyperbelsinus(X)
FSQRT	Quadratwurzel(X)
FTAN	Tangens(X)
FTANH	Hyperbeltangens(X)
FTENTOX	10^X
FTWOTOX	2^X

Befehle mit zwei Operanden

Bei diesem Befehlstyp werden zwei Eingabewerte benötigt, die verknüpft werden sollen.

Dabei darf der erste Operand im Speicher, CPU-Register oder in einem FPU-Datenregister stehen. Der zweite Operand *muß* immer in einem FPU-Datenregister zu finden sein! In diesem FPU-Datenregister "landet" dann auch das Ergebnis der Operation.

Auch hier alle Befehle in einer tabellarischen Übersicht:

Mnemonic	Funktion
FADD	Addiert zwei Zahlen
FCMP	FPn – Quelle. Ergebnis setzt Condition Codes entsprechend
FDIV	FPn / Quelle => FPn
FMOD	FPn mod Quelle => FPn. Division läßt nur Rest übrig
FMUL	FPn * Quelle => FPn
FREM	Diff. von FPn z. nächstliegenden Vielfachen des Quellop.
FSCALE	FPn x INT(2^{Quelle}) => FPn
FSGLDIV	Division (Ergebnis im Single Real Format)
FSGLMUL	Multiplikation (Ergebnis im Single Real Format)
FSUB	FPn – Quelle => FPn

FPn = Datenregister-#n in der FPU

Programmsteuerbefehle

Wie in der CPU läßt sich – auch abhängig von bestimmten Bedingungen in der FPU – der weitere Programmfluß steuern (verzweigen). So gibt es z. B. die von der CPU her bekannten Sprungbefehle auch in Verbindung mit der FPU. Hier wird dann abhängig von den Bedingungsbits im *Floating Point Status Register* (*FPSR*) verzweigt.

Befehl	Operand	Operation
FBcc	<label>	IF Bedindung cc wahr, THEN GOTO <label>
FDBcc	Dn,<label>	IF Bedingung cc wahr, THEN NOP ELSE Dn := Dn - 1 IF Dn <> -1, THEN GOTO <label>
FNOP		No Operation
FScc	<ea>*	IF Bedingung cc wahr, THEN <ea> := $FF ELSE <ea> := $00
FTST	<ea>* FPn	Zahl testen und FPSR-Condition-Code-Bits entsprechend setzen

<ea>*= Steht für "Effektive Adresse". Diese kann sowohl ein CPU-Datenregister als auch eine Speicherzelle sein, auf die mit den von der CPU her möglichen Adressierungsarten zugegriffen werden kann!

Für die Bedingungen cc in der obigen Tabelle können dann die folgenden Mnemonics eingesetzt werden:

Mnemo. 1	Mnemo. 2	Bedeutung	
EQ	SEQ	Gleich	(Diese Mnemonics sind nur der
NE	SNE	Ungleich	Vollständigkeit halber in die 32
F	SF	Immer falsch	möglichen Abfragen mit
T	ST	Immer wahr	aufgenommen worden!)
GE	OGE	Gleich oder größer	
GL	OGL	Größer oder kleiner (also ungleich!)	
GLE	OR	Größer, kleiner oder gleich (aber es ist kein NAN!)	
GT	OGT	Größer	
LE	OLE	Kleiner oder gleich	
LT	OLT	Kleiner	
NGE	UGE	[Das sind die gleichen Konditionen wie vorher, nur	
NGL	UEQ	daß zusätzlich das NAN-Bit gesetzt sein muß.	
NGLE	UN	Damit wird zusätzlich ermöglicht, auch Bedingungen	
NGT	UGT	zu prüfen, obwohl ein Wert verwendet wurde, der	
NLE	ULE	nicht ins gültige Datenformat paßt!]	
NLT	ULT	" " " "	

Die Mnemonics vom Typ 1 werden verwendet, wenn aufgrund der Bedingungsabfrage zusätzlich eine Exception ausgelöst werden soll, weil beim Test ein NAN beteiligt war und somit eine sogenannte "Unordered condition" aufgetreten ist. Als Folge davon wird das BSUN-Bit (Branch/Set on Unordered-Bit) im EXC-Byte des Floating-Point Status Register gesetzt. Die Mnemonics des Typs 2 setzen das BSUN-Bit nicht (auch wenn ein NAN beteiligt war)!

Befehle zur Systemsteuerung

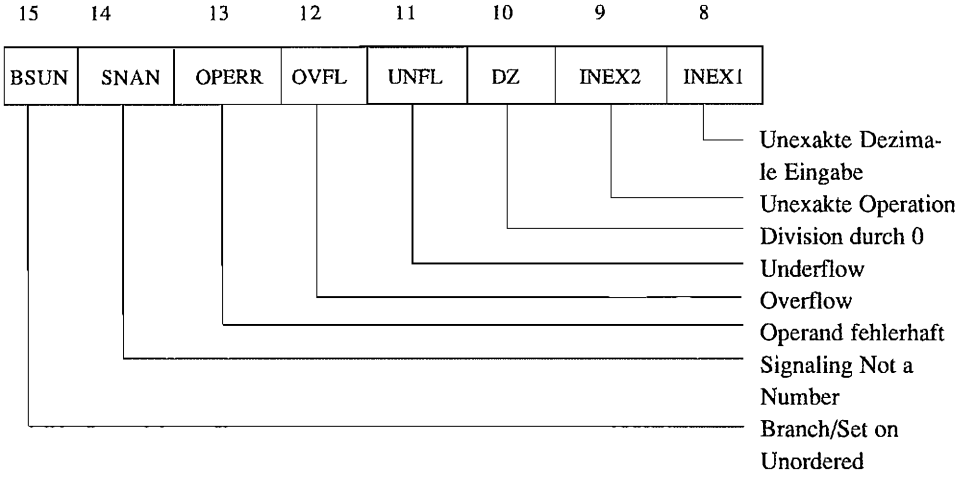
Mit diesen Befehlen kann z. B. unter bestimmten Bedingungen ein Trap (*FTRAPcc*) ausgelöst werden. Dabei sind für cc die gleichen Mnemonics zu verwenden, wie in der vorigen Tabelle bereits dargestellt.

Die ebenfalls zur Systemsteuerung gehörenden Befehle *FSAVE* und *FRESTORE* erlauben das Ablegen von internen FPU-Zuständen im Speicher (als sogenannten State Frames) und auch das spätere Wiederherstellen des alten Zustands. In Multitasking-Systemen werden solche Befehle angewendet, um zwischen verschiedenen Tasks hin- und herzuschalten. Damit ist es dann möglich, den FPU-internen Status zwischenzeitlich zu sichern, einen anderen Task zu bearbeiten und anschließend den ersten Task mit dem originalen FPU-Status wiederherzustellen und weiterzuführen!

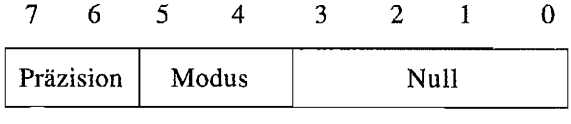
Das Floating Point Control Register (FPCR)

Dieses 32 Bit-Register verwendet zur Zeit nur die beiden niederwertigsten Bytes. Es kann vom Benutzer gelesen und beschrieben werden (z. B. durch FMOVE.L FPCR,d0).

Das *Exception Enable byte* (*ENABLE*) (Bits 8..15 des FPCR) erlaubt die Freigabe (Bit gesetzt) und das Sperren (Bit gelöscht) von verschiedenen Exceptions, die durch die FPU veranlaßt werden können.



Im *Mode Control Byte* (*MODE*) (Bits 0..7 des FPCR) kann der Anwender bestimmte Modalitäten für den Rundungsprozeß einstellen.



Durch die beiden *Modus-Bits* wird festgelegt, wie ungenaue Resultate gerundet werden sollen. (Es läßt sich nicht jede Dezimalzahl exakt als Binärzahl mit einer festen Stellenzahl darstellen).

0 0 Der nächstliegende, darstellbare Wert wird genommen. Sollten sich zwei Werte anbieten, die beide gleich "nahe" liegen, so wird der Wert mit dem niedrigstwertigen Bit=0 genommen!

0 1 Als Round-to-zero bezeichnet. Effekt ist der, daß die Bits rechts der Rundungsstelle gelöscht werden!

1 0 Es wird zum nächstliegenden Wert in Richtung $-\infty$ gerundet.

1 1 Rundung auf den nächstliegenden Wert in Richtung $+\infty$.

Die beiden Präzisionbits spezifizieren, wo der Rundungspunkt der Mantisse liegen soll.

0 0 Extended Precision. Es wird auf eine 64-Bit-Mantisse gerundet.

0 1 Single Precision. Rundung auf eine 24-Bit-Mantisse.

1 0 Double Precision. 53-Bit-Mantisse für die Rundung vorsehen.

1 1 RFFU (Soll heißen: Reserved for future use!)

Das Floating Point Status Register (FPSR)

Alle vier Bytes dieses 32-Bit-Registers werden verwendet. Lesen und Beschreiben des Registers durch den Anwender ist möglich (z.B. durch FMOVE.L d0,FPSR).

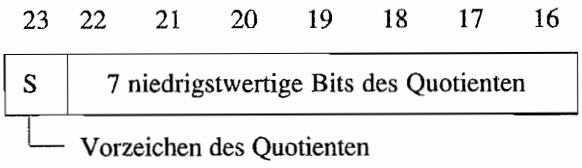
Das höchstwertige Byte (Bits 24..31) wird als *Floating Point Condition Code Byte* (*FPCC*) bezeichnet und signalisiert in 4 Bits den Bedingungscode einer mathematischen Operation, bei dem ein FP-Datenregister beteiligt war. Wichtig dabei ist, daß die Bedingungscodes *nicht* von der ausgeführten Operation abhängig sind, sondern von dem sich als Resultat ergebenden Datentyp!

31	30	29	28	27	26	25	24
Null				N	Z	I	NAN

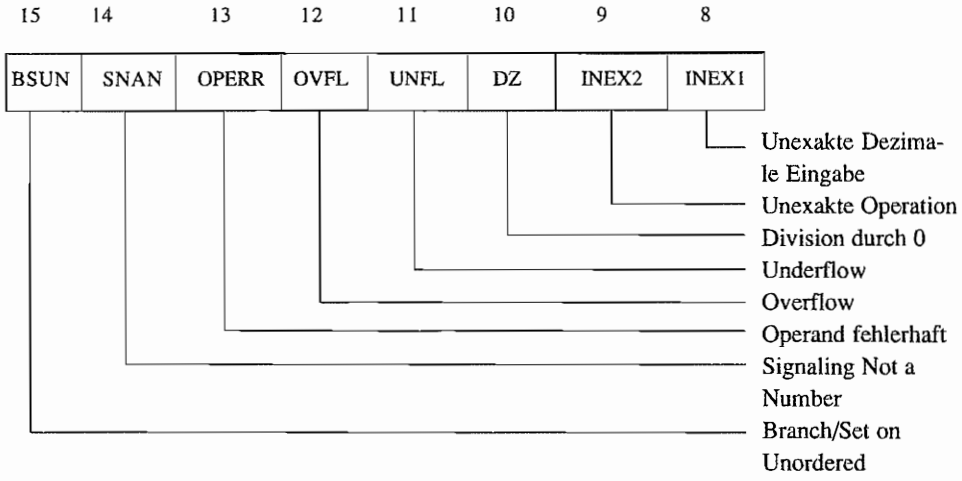
N Negatives Resultat (Negative-Bit)
Z Ergebnis ist Null (Zero-Bit)
I Resultat ist +/– Unendlich (Infinity-Bit)
NAN Ergebnis ist keine gültige Zahl oder Unordered (= bei der ausgeführten Operation war mindestens ein Eingabe-Operand ein NAN!)

Die Bits 16 .. 23 des FPSR bilden das *Quotient Byte*. Es wird am Ende einer FMOD oder FREM-Operation entsprechend gesetzt.

Dabei enthält es in den niedrigstwertigen 7 Bits die 7 untersten Bits des Quotienten und im höchstwertigen Bit dessen Vorzeichen.



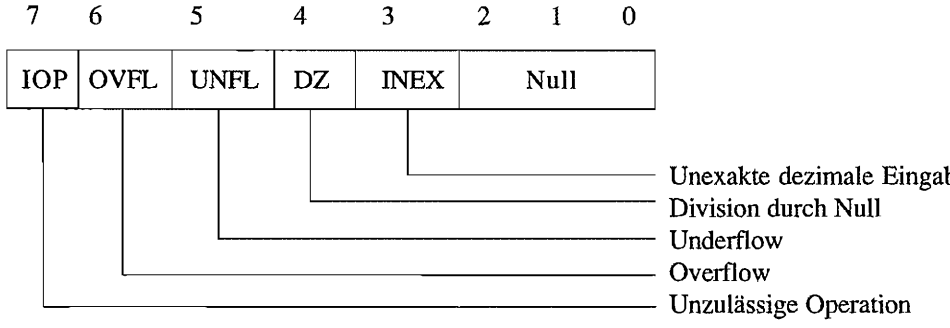
Das *Exception Status Byte* (*EXC*) ist in den Bits 8..15 des FPSR enthalten. Es enthält je ein Bit für die acht möglichen Ausnahmebedingungen (Exceptions), die durch eine der letzten ausgeführten FPU-Operationen aufgetreten sind. Vor dem Start der meisten FPU-Operationen wird dieses Bit gelöscht, so daß ein Exception-Handler erkennen kann, welche Fließkomma-Operation(en) zur Ausnahmebehandlung führten.



Die Bitanordnung stimmt dabei mit dem ENABLE-Byte des Floating Point Control Registers (FPCR) überein. Auch wenn die Exception durch das korrespondierende Bit im ENABLE-Byte gesperrt sein sollte, so wird doch das Exception-Status-Bit bei Auftreten einer entsprechenden Ausnahmebedingung gesetzt!

Für den Fall, daß ohne Ausnahmeverarbeitung gearbeitet wird, existiert das *Accrued Exception Byte* (*AEXC*) in den Bits 0..7 des FPSR. In diesem Register werden alle Floating-Point Exceptions, die nach dem letzten Löschen des Registers (muß der User allerdings selbst veranlassen!) im Laufe der letzten ausgeführten Operationen aufgetreten sind, "zusammengesammelt".

Das erspart dem User das Überprüfen des EXC-Bytes nach jeder Fließkommaoperation! Es reicht dann aus, am Ende einer Serie von Fließkommaoperationen nachzuschauen, ob zwischenzeitlich eine Ausnahmebedingung aufgetreten ist.



Das Floating Point Instruction Address Register (FPIAR)

Da ja bei der Zusammenarbeit zwischen CPU und FPU zum großen Teil Prozesse "nebeneinander" herlaufen, ist es erforderlich festzuhalten, bei welchem Befehl (Adresse) die FPU denn gerade ist. Sonst könnte bei einer FPU-Exception die CPU (und damit der Exception-Handler) nicht so einfach feststellen, bei welchem FPU-Befehl die Ausnahmebedingung auftrat.

Im FPIAR wird deshalb jeweils die Adresse des augenblicklich zu bearbeitenden FPU-Befehls vor seiner Ausführung abgelegt.

Kapitel 3: ROM und RAM beim TT

Viel Platz für das Betriebssystem – Das TT-ROM

Die im TT zusätzlich hinzugekommene Hardware braucht natürlich entsprechende Unterstützung durch Betriebssystem-Software. So kommt denn ATARI im TT (und STE) auch nicht mehr mit den im ST noch ausreichenden 192 KByte ROM aus.Verwendet werden nun 1 MBit-EPROM-Chips, wobei das Platinenlayout im TT (und STE) entsprechend flexibel gestaltet wurde. So können sowohl EPROMs des Typs 27256, 27512, 27C1000, 27C1001 und 27C1010 verwendet werden. Es lassen sich aber auch 1-MBit-ROMs verwenden. Von der Zugriffsgeschwindigkeit reichen 150 ns-Typen wohl aus.

Beim TT sind vier (beim STE sind es 2!) 32polige Sockel für die Betriebssystem-Chips vorgesehen. Damit ist es im TT also möglich, 512 KByte-Sytemsoftware unterzubringen. In der Tat sind auch alle vier Sockel mit Megabit-Eproms bestückt. Was aber noch längst nicht heißt, daß diese auch in ihrer Kapazität voll ausgereizt wurden. Es ist wohl eher so, daß man so gerade über die 256 KByte-Grenze "gerutscht" ist (TOS 3.01 und 3.05 sind ca. $41000 = 266240 Bytes lang). Es sind aber auf jeden Fall alle vier Sockel mit Chips zu bestücken, da ja im TT ein 32-Bit-Datenbus verwendet wird. Jeder Chip bedient also ein Viertel des Datenbusses.

Das Betriebssystem-ROM erscheint im TT an zwei Stellen des CPU-Adreßraums. Zum einen ist es so ziemlich am oberen Ende des "ST-Adreßraums" (die unteren 16 MByte, die von einer 68000er-CPU adressiert werden können!) von $E0 0000 .. $EF FFFF zu "finden". Dafür sorgt die MMU in der 68030-CPU. Eigentlich liegt der ROM-Bereich am hinteren Ende des 68030er-Speicherbereichs bei $FFE0 0000 .. $FFEF FFFF!

Auch beim TT findet man wieder in den ersten beiden Langwörtern des Speicherraums die Startwerte für die CPU (Adresse $0 = Startwert für den Supervisor Stackpointer; Adresse $4 = Startwert für den Programmzähler). Diese 8 Bytes werden, wie beim ST(E) aus dem Betriebssystem-ROM ($FFE0 0000. .$FFE0 0007) durch entsprechende Adreßdekodierung dort eingeblendet.

RAM ist nicht gleich RAM! – Der TT kennt mindestens zwei "Sorten"

Um dem TT zeitgemäße Leistungsfähigkeit zukommen zu lassen, wurde es erforderlich, den beim ST(E) doch eher eingeschränkten Speicherraum von max. 12 MByte (ab $C0 0000 hat

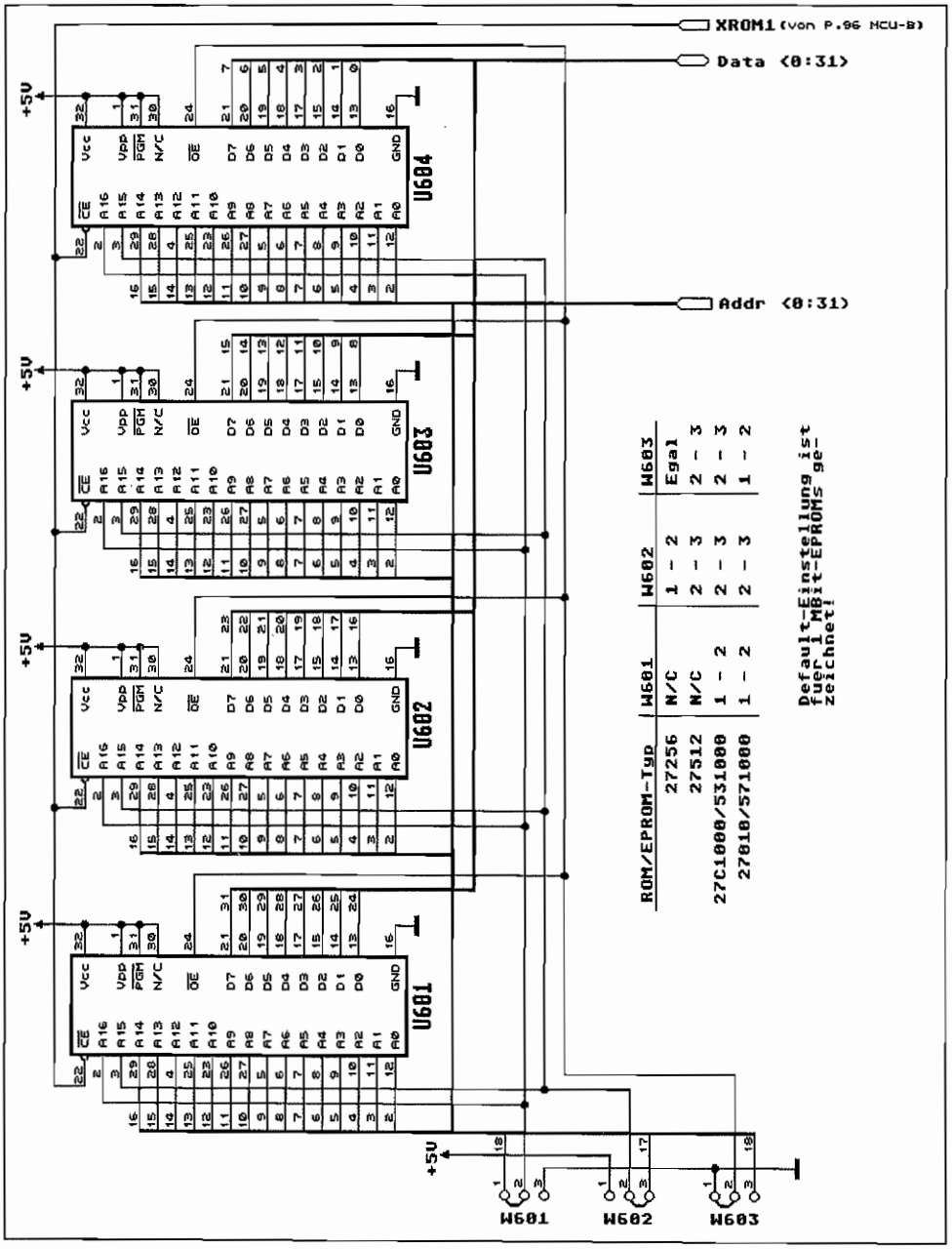


Abb. 3.1: Die Einbindung der ROMs beim TT

ATARI ja den Speicherbereich für eigene Anwendungen wie z. B. Grafikkarten reserviert!) entsprechend erweitern zu können. Vor allen Dingen, wo man doch eine CPU einsetzt, die immerhin 4 GByte (!) direkt ansprechen kann, mußte man nach Lösungsmöglichkeiten suchen, wie man diesen Adreßraum nutzbar machen konnte, ohne gleichzeitig die Kompatibilität zu den ST(E)-Geräten zu verlieren.

Die Lösung meint man mit dem sogenannten "Alternate RAM" gefunden zu haben. So existiert dann zusätzlich zum ST-kompatiblen RAM, in dem sich neben der CPU auch noch der DMA-Baustein für ACSI-Datentransfers und der Videocontroller die Zugriffe auf das vorhandene RAM teilen, noch eine zweite Sorte RAM, die nur durch die CPU angesprochen werden kann. Wobei das mit dem "nur durch die CPU ansprechbar" nicht immer ganz korrekt ist. Lediglich das Alternate-RAM, welches über den VME-Busanschluß an den TT angeschlossen werden kann, und das RAM in dem Uhrenchip des TT können nur durch die CPU angesprochen werden.

Das sogenannte Fast-RAM hingegen kann sowohl von der CPU als auch von den beiden DMA-Bausteinen für den SCSI-Bus und den schnellen seriellen Schnittstellen-Controller (SCC) benutzt werden. Allerdings kann der Videocontroller *nicht* an dieses Alternate-RAM und fällt somit als "Bremse" bei Fast-RAM-Zugriffen aus!

ST-kompatibles RAM

Im TT hat sich ATARI den Adreßraum von $8 .. $9F FFFF (mit einer weiteren "Option" für den Bereich von $A0 0000 .. $DF FFFF) explizit als "dual purpose"-RAM reserviert. Dieses RAM wird (zeitlich) abwechselnd von der CPU und dem Videocontroller (deswegen wohl "dual purpose"!) benutzt. Dabei kann der Zugriff auf dieses RAM in minimal 250ns-Abständen zwischen den beiden Konkurrenten wechseln. Während eines laufenden Display-Zyklusses wird der Videocontroller natürlich nicht einfach unterbrochen, sondern die CPU bekommt die nächstmöglichen 250ns "zugewiesen".

Auf der Hauptplatine des TTs findet sich bereits standardmäßig 2 MByte ST-RAM. Aufgebaut ist dieses aus 16 DRAMs der Organisation 256K x 4 Bit! Es werden dabei 100ns-DRAMs verwendet. Soweit gibt es bisher keine wesentlichen Unterschiede zum RAM in ST(E)s.

Aber nun zur Abweichung! Diese 16 RAM-Chips werden zu einem 64 Bit-Memory-Datenbus zusammengeschaltet! Dadurch ist es dem Videocontroller möglich, mit einem RAM-Zugriff sofort 8 Bytes an Bildschirmdaten auszulesen! Die Anpassung des 64 Bit-Memory-Datenbusses an den im System ansonsten verwendeten 32 Bit-Datenbus übernehmen zwei Spezialchips (als FUNNEL = Trichter bezeichnet).

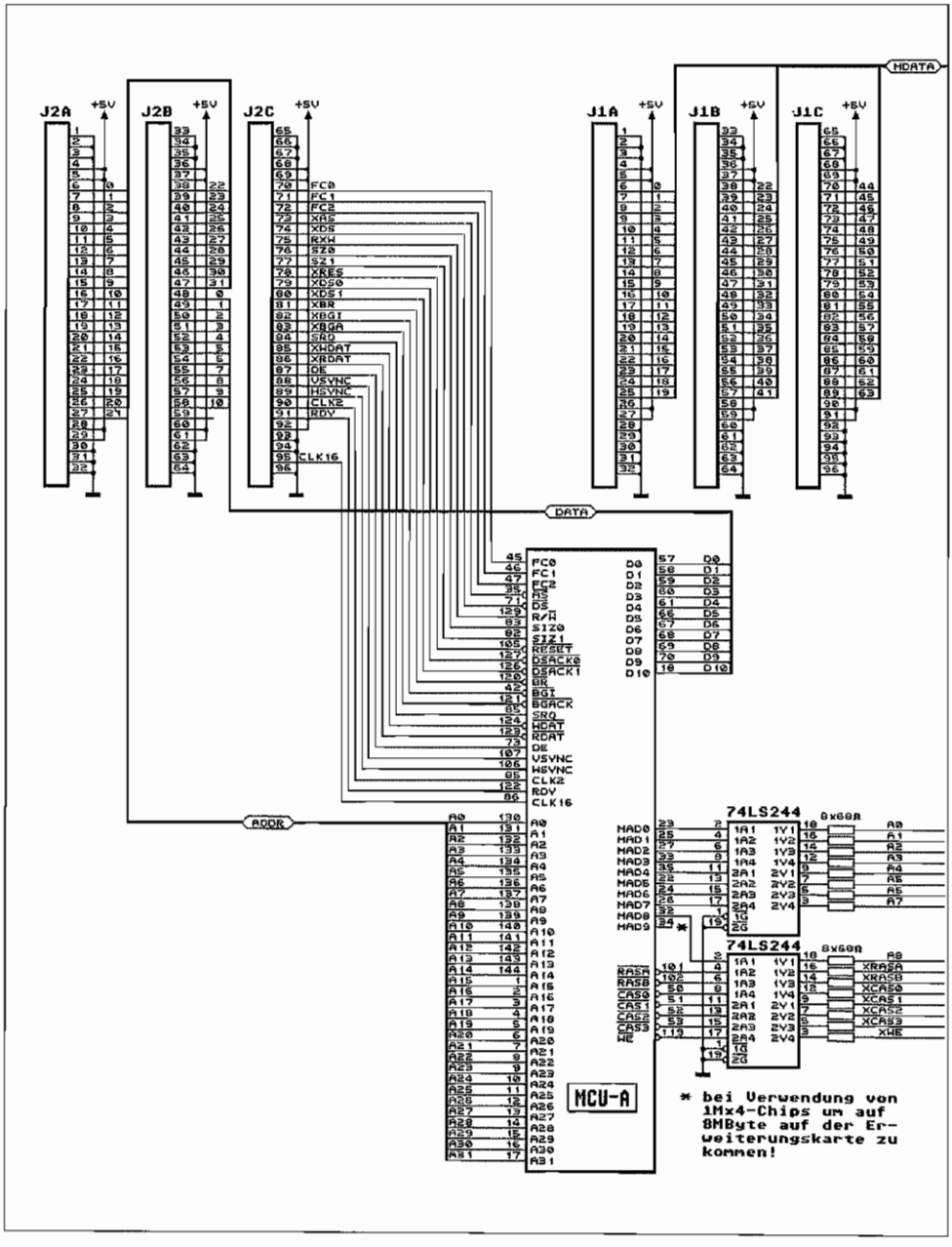


Abb. 3.2: Aufbau der ST-RAM-Erweiterungskarte (linker Teil)

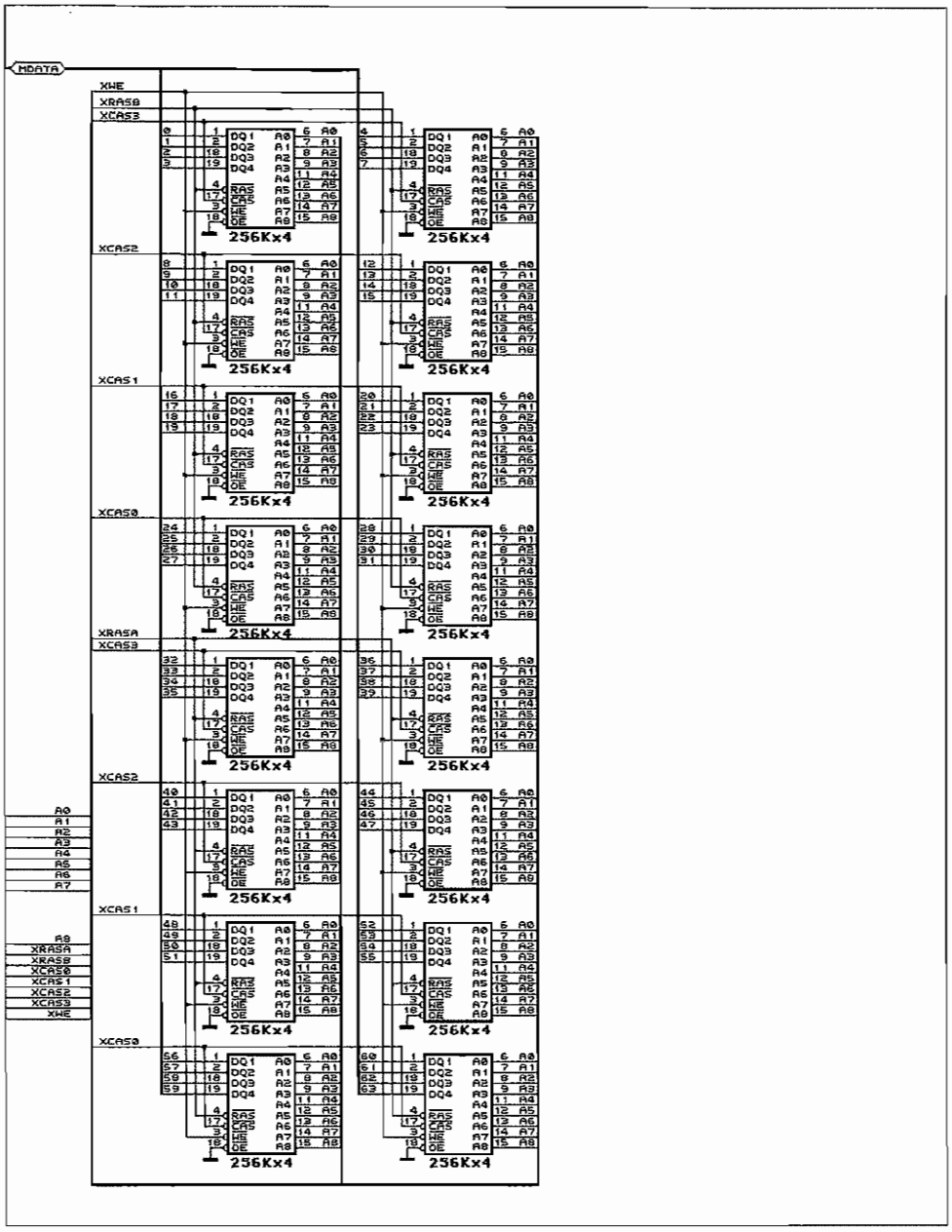


Abb. 3.2: Aufbau der ST-RAM-Erweiterungskarte (rechter Teil)

Natürlich ist es vorgesehen, dieses "dual purpose"-RAM zu erweitern. Auf der Hauptplatine (unter der Harddisk) befinden sich dazu zwei 96polige Federleisten nach DIN 41 612 (oder VG 95 324, deshalb auch häufig als VG-Leisten bezeichnet).

Es reicht aber nun nicht aus, einfach dort eine Platine mit nur den RAMs einzusetzen. Vielmehr benötigen diese RAMs eine eigene MCU (Memory Control Unit). Die MCU auf der TT-Hauptplatine hat genug mit dem dort untergebrachten RAM zu tun. Deshalb sind auch alle für eine MCU erforderlichen Signale auf diese beiden VG-Leisten geführt. ATARI hat dabei allerdings eine Signalaufteilung für diese beiden Leisten vorgenommen.

Die mit der Bezeichnung J501 versehene VG-Leiste auf der TT-Hauptplatine stellt dabei die 64 Leitungen für den Memory-Datenbus und die Versorgungsspannungsanschlüsse zur Verfügung (dieser Steckanschluß findet sein "Gegenstück" in dem Anschluß J1 der ST-RAM-Erweiterungssplatine in der Abbildung 3.2). Über die VG-Leiste J502 werden der 32 Bit-Adreßbus und die Signale für die MCU (plus Versorgungsanschlüsse) geliefert.

Außerdem werden noch die Busleitungen D0..D10 des 32 Bit-Datenbusses zur MCU der Erweiterungsplatine geführt (irgendwie müssen ja auch die Register in der MCU gesetzt werden können!). Auf der Erweiterungsplatine ist dieser Anschluß mit J2 bezeichnet!

Die MCU hat natürlich nicht nur die Aufgabe, für einen gemultiplexten Adreßbus zu den RAMs und die dazu gehörenden RAS- und CAS-Signale zu sorgen. Da ja DMA-Betrieb genauso möglich sein muß, ist in der MCU die entsprechende Logik untergebracht!

Gleiches gilt für Zugriffe des Videocontrollers. Also sind die dazu erforderlichen Signale ebenfalls zur MCU auf der Erweiterungskarte geführt.

Im Moment verwendet ATARI für seine RAM-Erweiterungskarten noch 16 DRAMs der Organisation 256K x 4 Bit und kommt so auf eine Kapazität von 2 MByte auf der Erweiterungskarte.

Sobald aber 4 MBit-Chips in entsprechenden Stückzahlen erhältlich sind, wird die Erweiterungskarte dann 8 MByte Kapazität umfassen können. Die MCU weist bereits den entsprechenden Anschluß für die gemultiplexte Adreßleitung 9 (MAD9) auf!

TT-spezifisches RAM – Das Fast-RAM

Den Spitznamen "Fast-RAM" hat diese Sorte RAM im TT wohl wegen seiner nahezu ausschließlichen Nutzung durch die CPU bekommen. ATARI spricht aus diesem Grunde in seinen Dokumentationen deshalb auch vom "single purpose"-RAM.

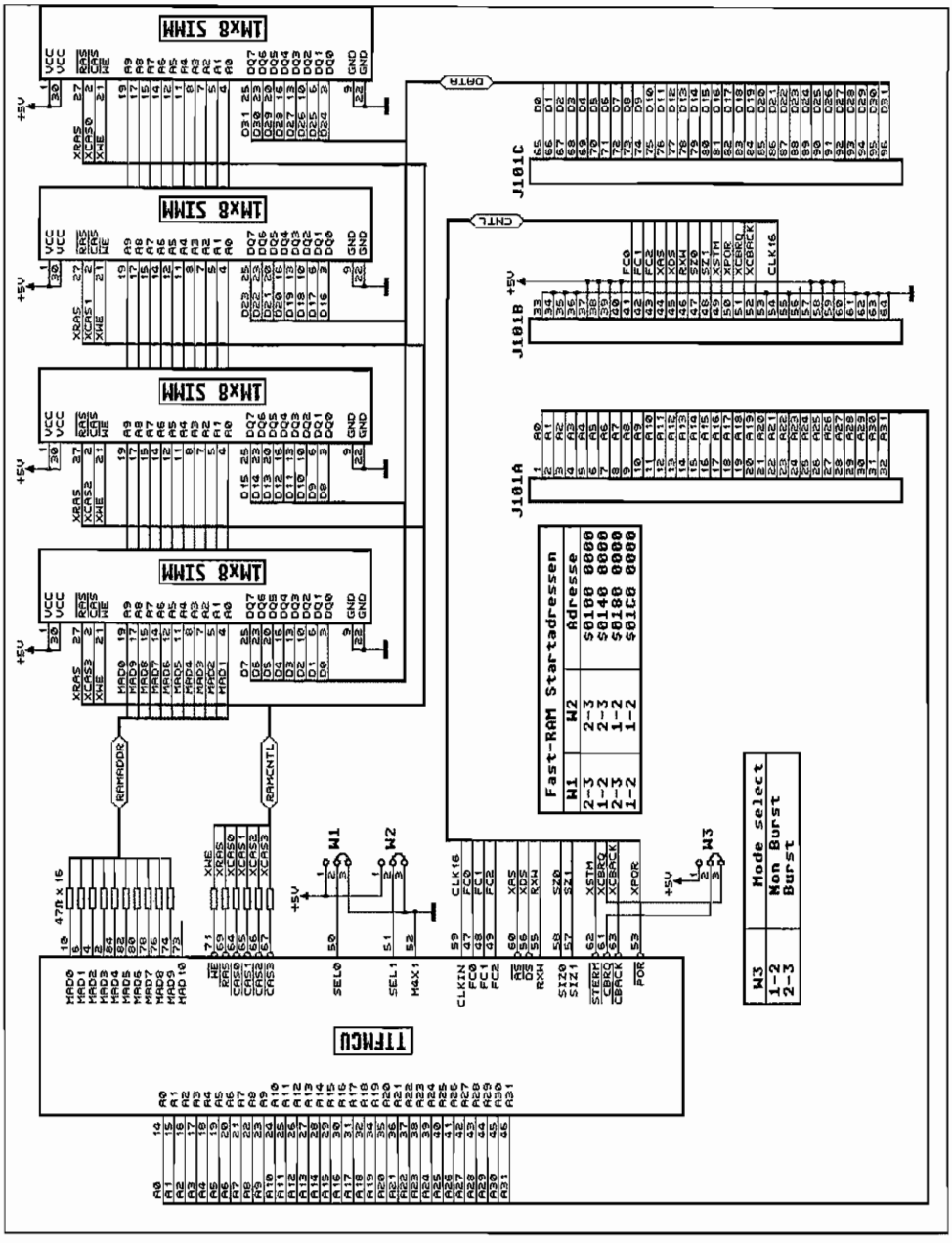


Abb. 3.3: Aufbau der FAST-RAM-Erweiterungskarte

Es ist als Erweiterung zum (ST)-RAM auf der Hauptplatine vorgesehen. Standardmäßig findet sich auf dem TT-Motherboard also kein Fast-RAM, sondern lediglich eine Anschlußmöglichkeit für eine entsprechende Erweiterungskarte.

Da der Videocontroller auf dieses RAM nicht zugreifen kann, entfällt auch die Notwendigkeit, es in Form eines 64 Bit breiten Datenbusses anzulegen. So findet man denn beim Erweiterungssteckplatz auch nur einen 32 Bit-Datenbus, mit direktem "Draht" (Busverbindung) zur CPU und FPU. Ähnlich wie bei der Erweiterung des ST-RAMs benötigt man auch für das Fast-RAM eine eigene FAST-MCU auf der Erweiterungskarte. Da aber keine Videocontrollerzugriffe in diesen Typ RAM vorgesehen sind, brauchen auch natürlich nicht so viele Steuer- und Meldeleitungen zur TTFMCU (TT-FAST-RAM-Memory Controller Unit) geführt zu werden.

ATARI verwendet im Moment auf seinen Erweiterungskarten vier SIM-Module der Organisation 1M x 8 Bit und kommt damit auf eine Kapazität von 4 MByte für diese Karte. Sobald aber die 4M x 8 Bit in größeren (und preiswerteren) Stückzahlen erhältlich sind, wird es auch eine 16 MByte-Karte für die Fast-RAM-Erweiterung geben. Ob dazu allerdings die gleiche TTFMCU verwendet werden kann, konnte von ATARI-Deutschland noch nicht bestätigt werden (die gemultiplexte Adreßbusleitung MAD10 ist zumindest schon mal vorhanden!).

Um die Cache-Speicher der MC68030 besonders effektiv zu füllen (im sogenannten Burstfill-Modus lassen sich bis zu vier Langwörter mit sehr wenigen Taktzyklen unmittelbar hintereinander einlesen!), werden für das Fast-RAM Bausteine verwendet, die im Nibble-Mode arbeiten. Damit ist es möglich, nach einem Zugriff mittels normaler RAS/CAS-Adressierung (erst Zeilenadresse, dann Spaltenadresse an die RAMs legen und jeweils durch RAS- bzw. CAS-Signal (aktiv Low) die Zelladresse im RAM-Chip auswählen) nur durch Ausgabe

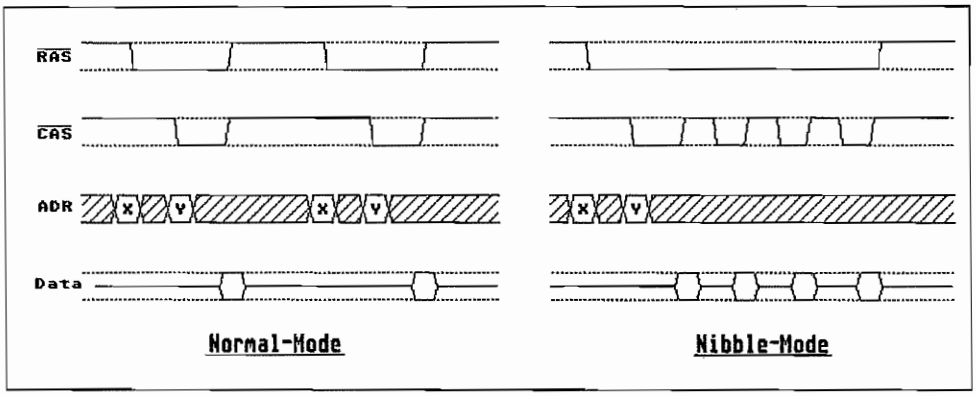


Abb. 3.4: Der Unterschied zwischen Normal- und Nibble-Mode-Zugriffen

weiterer CAS-Impulse Zugriff auf die drei folgenden Adressen zu erhalten. Das spart natürlich einiges an Zeit und erhöht somit den Datendurchsatz.

Auf der FAST-RAM-Erweiterungsplatine läßt sich durch Jumper außerdem einstellen, wo die Startadresse des FAST-RAM liegen soll. Ebenfalls läßt sich über einen Jumper der Burst-fill-Modus wahlweise abschalten (die Standardeinstellung ist natürlich mit Burst-fill-Modus!).

Der Cartridgeport beim TT

ATARI hat beim TT auch den Anschluß für ROM-Cartridges beibehalten. Dieser ist bis auf weiteres kompatibel zum ST(E)-Cartridge-Anschluß. Also keine Erweiterung des 128 KByte-Adreßraums am Cartridge-Port. Allerdings finden sich die 128 KByte des Cartridges zum einen im ST-Adreßraum an gewohnter Stelle (bei $FA 0000 .. $FB FFFF) als auch am oberen Ende des 32 Bit-Adreßraums bei $FFFA 0000..$FFFB FFFF!

Die Behandlung der Cartridge-Software funktioniert weiterhin wie bereits beim ST erläutert.

Ein Unterschied besteht jedoch in bezug auf das Timing. Da der TT-Cartridge-Anschluß sich in einer 16 MHz-Umgebung befindet, werden manche Dongles mit speziellen Zugriffsmechanismen evtl. Probleme mit dem etwas anderen Timing haben. Das liegt unter anderem daran, daß jetzt nicht mehr die _UDS- und _LDS-Signale (Upper Data Strobe und Lower Data Strobe) für den Cartridgeport von der CPU erzeugt werden, sondern von der MCU nachgebildet werden. Die MC68030-CPU mit ihren 32 Bit Busbreite benutzt für die Signalisierung, welche Datenbusbreite sie beim laufenden Buszyklus verwenden will, ja die Signale SIZ0 und SIZ1 sowie _DSACK0 und_DSACK1!

Außerdem hat ATARI einen kleinen Schritt in Richtung Sicherheit gemacht und die +5V-Leitung zum Cartridge-Port im TT über eine Sicherung (0,5 A) geführt.

Seinen ursprünglichen Zweck, nämlich das Bereitstellen von Software auf ROM-Modulen, erfüllt der TT-Cartridgeport jedoch ohne weitere Modifikationen bei bereits bestehenden Modulen. Schnellere ROMs bzw. EPROMs als am ST(E) sind am TT-Cartridgeport also nicht erforderlich!

Soviel zu ROM und RAM im TT. Das Alternate-RAM im Uhrenchip wird bei der Besprechung des MC146818 (so heißt der Uhrenchip im TT!) mit angesprochen.

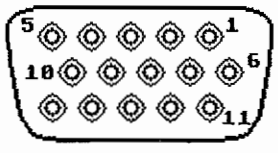
Kapitel 4: Der Videocontroller im TT

ATARI hat dem TT zusätzlich zu den schon vom ST(E) her bekannten Grafikbetriebsarten einige TT-spezifische Modi eingebaut. Insgesamt sechs verschiedene Bildschirmauflösungen sind beim TT nun möglich. Insbesondere für die TT-spezifischen Auflösungen, bei denen der Bildschirmspeicher nahezu die fünffache Größe gegenüber dem ST-Bildschirmspeicher aufweist, ist natürlich eine entsprechend schnelle Videohardware erforderlich.

In der höchsten Auflösung werden 1280 x 960 Bildpunkte in Monochrom dargestellt. Das Ganze funktioniert dank einer hohen Vertikalen Ablenkfrequenz von 71 Hz auch noch flimmerfrei! Bei so hohen Pixeltakten greift ATARI dann aber auch auf bewährte Spezialchips für den Taktoszillator und das Shiftregister zurück (National Semiconductor Chip-Set mit DP8530 und DP8516). Die Horizontale Ablenkfrequenz liegt in dieser Betriebsart dann bei ca. 74 kHz! Für die anderen Modi verwendet ATARI "hausgemachte" Videohardware, wie schon bei der ST(E)-Serie.

Der Videoanschluß

Beim TT ist ATARI von den Spezialsteckverbindern der ST(E)-Serie abgegangen und verwendet nun gleiche Steckverbinder wie beim "Industriestandard". So findet sich für den Monitoranschluß denn auch eine 15polige Subminiatur-D Buchse mit "kleinem" Kontaktabstand. Die Belegung dieser Buchse ist in Abbildung 4.1 dargestellt.



TT-Videobuchse
(Aufsicht von außen)

Pinbelegung

1	-	Rot-Signal
2	-	Grün-Signal
3	-	Bau-Signal
4	-	Monochrom Out (+)
5	-	Masse
6	-	Rot-Masse
7	-	Grün-Masse
8	-	Blau-Masse
9	-	Monochrom-Detect (Low-aktiv)
10	-	Masse
11	-	Nicht belegt
12	-	Nicht belegt
13	-	HSync (Low-aktiv)
14	-	VSync (Low-aktiv)
15	-	Monochrom out (-)

Abb. 4.1: Der Videoanschluß beim TT

Da aber die TT-Hoch-Auflösung ebenfalls über diese Buchse geführt ist, kann man bei der Belegung nicht von einer Kompatibilität zum "Industriestandard" sprechen. Allerdings lassen sich handelsübliche VGA-Monitore in den ersten fünf Auflösungen am TT betreiben. Das liegt daran, daß ATARI im TT bei einer Vertikal-Ablenkfrequenz von 60 Hz bleibt (auch bei der ST-Hoch-Auflösung! Also aus ist's mit der Flimmerfreiheit in ST-Hoch am TT!).

Die horizontale Ablenkfrequenz liegt bei ca. 32 kHz in den unteren fünf Auflösungen.

Für die Signalpegel der drei Farbsignalausgänge und die Synchronsignale gilt das gleiche, wie bereits beim ST(E) gesagt. R-, G- und B-Anschluß liefern an einer Last von 75Ω eine Signalamplitude von 1 Vss. Die V- und H-Synchronimpulse liegen im TTL-Pegel vor und sind Low-Aktiv!

Warum ATARI allerdings nur ca. 75% der horizontalen Breite einer Zeile nutzt, ist mir nicht klar. So könnte man doch immerhin statt der momentanen max. 640 Pixel pro Zeile "locker" auf über 800 bis 900 Pixel pro Zeile kommen.

Die ST-Betriebsmodi

Die Organisation des Bildspeichers in den ST-Auflösungen ist die gleiche, wie bereits für den ST(E) beschrieben. Auch hier wird wieder mit Bitplanes gearbeitet. In allen drei ST-Modi ist die Größe des Bildschirmspeichers 32000 Byte!

ST-High-Resolution

Im ST-Monochrombetrieb wird der gesamte Bildschirmspeicher als eine Bitplane aufgefaßt. Je 40 Words entsprechen einer Bildschirmzeile.

Das erste Word im Bildschirmspeicher ist dabei in der ersten Bildschirmzeile für die ganz links stehenden 16 Bits zuständig. Siehe hierzu auch Abbildung 2.6 im ST-Hardwareteil "Das Grafiksystem".

Das höchstwertige Bit jedes Words ist dem ganz links liegenden Pixel der 16 durch dieses Word gesteuerten Bildpunkte zugeordnet.

ATARI bezeichnet diesen Modus auch als Duochrom-Modus! Es können nämlich nicht nur die Zustände Schwarz und Weiß dargestellt, sondern zwei Farben aus der 256 Register umfassenden TT-Palette für Vorder- und Hintergrund benutzt werden. Verwendet werden immer die beiden Farben aus den beiden letzten TT-Farbpaletten-Registern (Nr. 254 und Nr. 255).

ST-Medium-Resolution

Hiermit können je Zeile 640 Bildpunkte angesteuert werden. In vertikaler Richtung werden 200 Zeilen geschrieben, was bei vielen grafischen Abbildungen zu "verzerrten" Proportionen führt. Die Farbe eines Pixels kann aus einem von den ersten vier Farbpalettenregistern ausgewählt werden. Jedes Palettenregister bietet die Möglichkeit, 4096 Farben einzustellen. Die Organisation der Farbbits der drei Primärfarben ist dabei die gleiche wie beim STE.

Diese Colorbetriebsart arbeitet mit zwei wortweise ineinander verzahnten Bitplanes. Je zwei "übereinander" liegende Bits dieser Planes bilden einen Indexwert für das auszuwählende Farbregister. Dieses Farbregister enthält dann je 4 Bits für die drei Primärfarben Rot, Grün und Blau. Die Pixelfarbe wird also indirekt über den Indexwert aus je 2 Bits der beiden Bitplanes abgeleitet. Siehe hierzu auch Abbildung 2.7 im ST-Hardwareteil "Das Grafiksystem". Eine Bildschirmzeile besteht in dieser Betriebsart aus zweimal 40 Wörtern zu 16 Bits!

ST-Low-Resolution

Das Darstellungformat umfaßt 320 (hor.) x 200 (vert.) Bildpunkte. Jedes Pixel kann von seiner Farbe her aus einem von 16 Farbpalettenregistern ausgewählt werden. Auch hier läßt sich in jedem Farbpalettenregister eine von 4096 Farben einstellen.

Es wird mit vier wortweise ineinander verzahnten Bitplanes gearbeitet. Je vier "übereinander" liegende Bits bilden dann den Indexwert für das zu selektierende Palettenregister mit der Pixelfarbinformation. Eine Bildschirmzeile besteht aus viermal 20 Wörtern zu je 16 Bits! Siehe auch Abbildung 2.8 im ST-Hardwareteil "Das Grafiksystem".

Die TT-Grafik-Betriebsarten

Die Video-Hardware des TT bietet noch weitere drei Auflösungen. Bis auf die TT-High-Auflösung lassen sich auch diese Darstellungsmodi auf dem gleichen Monitor wie die ST-Modi abbilden. In allen drei TT-Modi ist die Größe des Bildschirmspeichers ebenfalls wieder gleich. Sie beträgt nämlich genau 153600 Byte!

TT-Low-Resolution

Diese Betriebsart bildet 320 Pixel je Zeile bei 480 Zeilen pro Bild ab. Dabei läßt sich die Farbe eines Pixel aus einem von 256 (!) Farbpalettenregistern auswählen. Jedes Palettenregister kann eine von 4096 möglichen Farbeinstellungen aufweisen.

Um aus 256 Palette-Registern auswählen zu können, sind 8 Bits zur Indexbildung erforderlich. Deshalb wird in dieser Auflösung mit acht wortweise ineinander verschachtelten Bitplanes gearbeitet. Eine Bildschirmzeile ist damit 320 Pixel / 16 (Bits pro Wort) x acht (Planes) = 160 Words "lang". Bei 480 Zeilen pro Bild ergibt das einen Bildschirmspeicher der Größe von 153600 Bytes (WZBW = Was Zu Beweisen War).

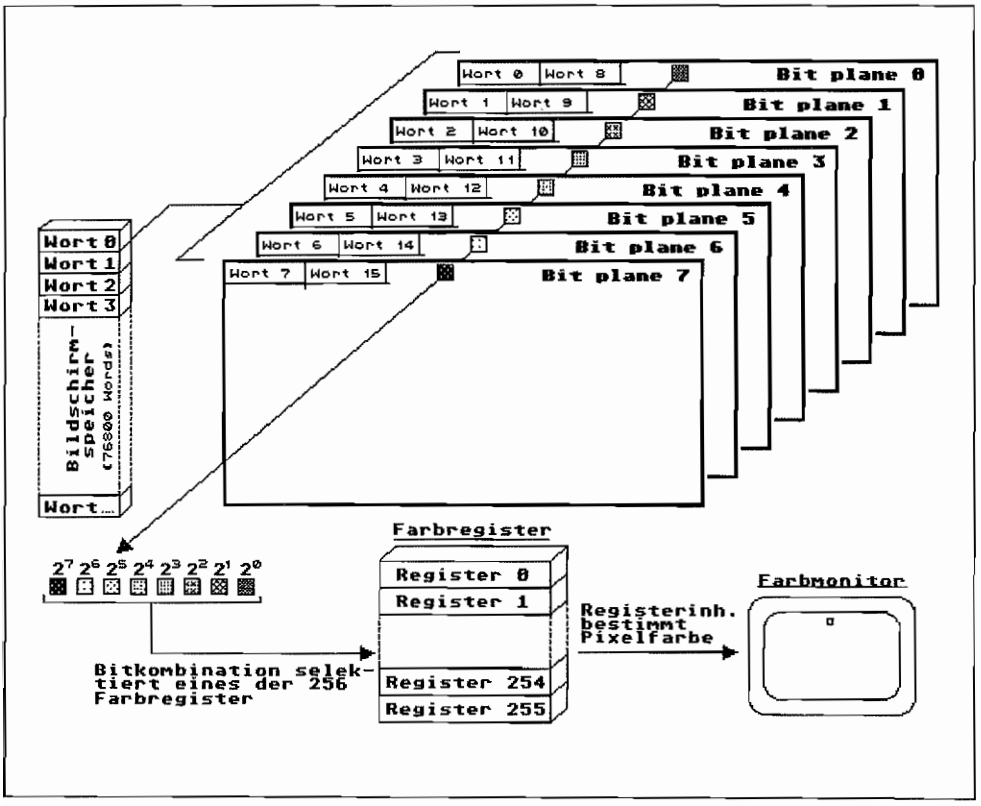


Abb. 4.2: Die Organisation des Bildschirmspeichers in TT-Low

TT-Medium-Resolution

Es werden je Zeile 640 Pixel dargestellt. In vertikaler Richtung werden 480 Zeilen geschrieben. Je Pixel kann aus einem der unteren 16 Farbpalettenregister der Wert für die Pixelfarbe (1 aus 4096 Farbeinstellungen je Palettenregister möglich) entnommen werden.

Da nur aus den ersten 16 Palettenregistern der 256 möglichen ausgewählt wird, sind auch nur vier Bits (= Planes) zur Bildung der Palettenreg.-Nummer erforderlich. Diese vier Bitplanes sind ebenfalls wortweise ineinander verzahnt, wie in der Abbildung 4.3 dargestellt.

Eine Bildschirmzeile belegt im Bildschirmspeicher 4 x 640/16 = 160 Words (= 320 Bytes).

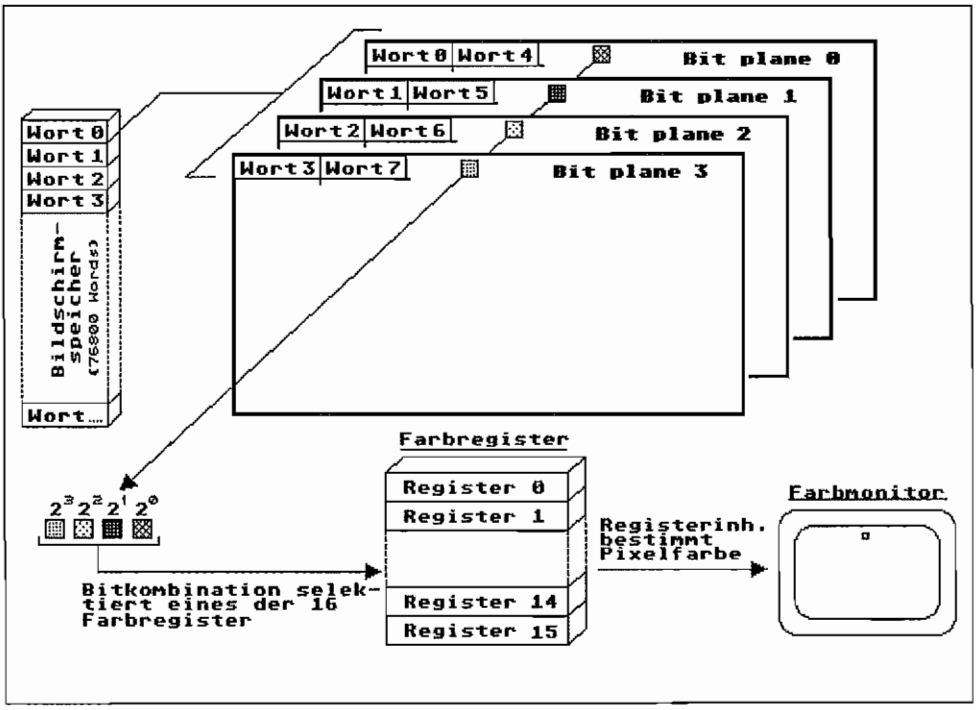


Abb. 4.3: Die Organisation des Bildschirmspeichers bei TT-Medium

TT-High-Resolution

Genau schon wie beim ST(E) die High-Resolution benötigt auch diese hochauflösende Bildschirmdarstellung beim TT einen speziellen Monitor.

Hier wird mit hohen Ablenkfrequenzen von 71 Hz für die Vertikalfrequenz und ca. 74 kHz für die Horizontalfrequenz gearbeitet. Zum einen soll ein ruhiges Bild für die Augen erzielt

werden (hohe V-Frequenz), und zum anderen soll eine große Menge an Zeilen (hohe H-Frequenz) auf dem Schirm dargestellt werden. Das führt aber ebenfalls mit den geforderten 1280 Pixeln/Zeile zu hohen Pixeltakten.

Im günstigsten Fall (ohne vordere und hintere Schwarzschulter einer Zeile = linker und rechter Rand einer Bildschirmzeile, der schwarz geschaltet wird, um Zeilenrückläufe unsichtbar zu machen, und ohne Zeilensynchronimpulse zu berücksichtigen) sind dabei Pixeltakte von 1280 x 74 kHz = 95 MHz zu erwarten!

Dafür braucht man zum einen schon hochwertige Monitore mit entsprechenden Bandbreiten im Videoteil und schnellen Strahlablenksystemen für die hohen Ablenkfrequenzen.

Zum anderen ist der Videocontroller natürlich ebenfalls auf solche Anforderungen hin auszulegen. Atari hat dabei allerdings nicht auf "Hausgemachtes" zurückgegriffen, sondern einen bewährten Chipsatz (Oszillator und Shifter) von National Semiconductor eingesetzt.

Unter anderem kann man diese Pixelinformation auch nicht mehr in TTL-Technik ausgeben, sondern verwendet spezielle differentielle Ausgangstreiberstufen in ECL-Technik (Emitter Coupled Logic) dazu.

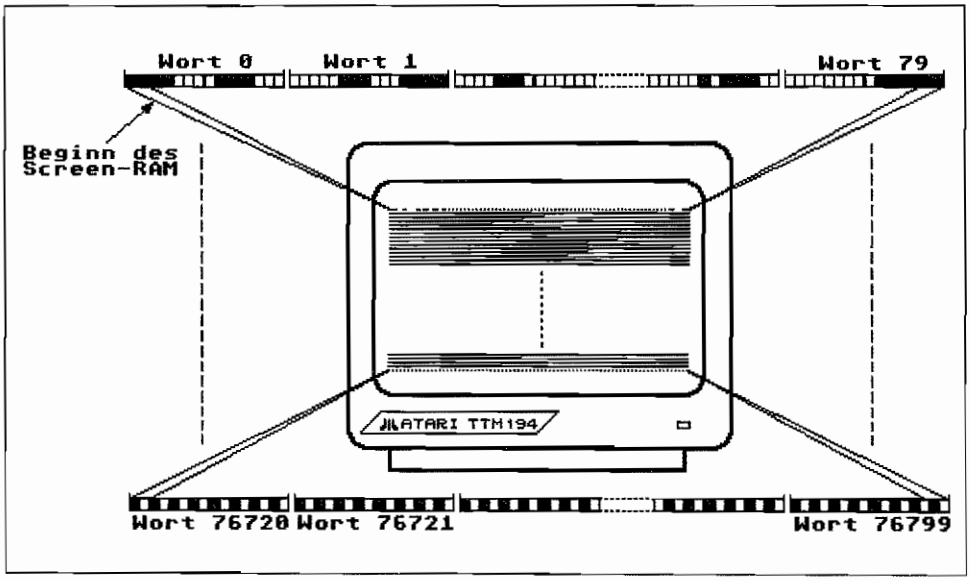


Abb. 4.4: Die Organisation des Bildschirmspeichers in TT-High

Der Bildschirmspeicher, mit seinen auch hier 153600 Bytes, wird als eine Bitplane interpretiert. Jedes Bit repräsentiert also ein Pixel. Pro Bildschirmzeile werden 80 Words zur Darstellung der Pixelinformation verwendet.

Die Programmierung der Video-Hardware

Auch bei der Video-Hardware im TT hat ATARI versucht, möglichst viele Funktionen in möglichst wenigen Bausteinen zu integrieren. So reduziert sich die Hardware für alle Betriebsarten mit 60 Hz V- und 32 kHz H-Frequenz (also ohne die TT-High-Auflösung) auf einen Chip. Allerdings sind auch im TT wieder einige Funktionen der Video-Hardware in andere Custom-Chips ausgelagert worden.

Die Umsetzung der Bildspeicherdaten in Bildinformationen übernimmt der TT-SHIFTER. Dort sind die Schieberegister untergebracht, die die Bildschirmspeicherbits nach dem parallelen Laden über den Datenbus mit MCU- und FUNNEL-Hilfe seriell als Bildinformationssignal ausgeben. (FUNNEL = Trichter; diese Spezial-Chips setzen den 64 Bit breiten ST-RAM-Datenbus für den "Rest des TTs" auf 32 Bit um und erlauben der Videohardware einen Zugriff auf immer 64 Bits auf einmal!)

Der Video-Adress-Counter und das Video-Base-Register sind allerdings in der MCU zu finden. Da bei einer ST-RAM-Erweiterung noch eine zweite MCU auf der Erweiterungsplatine vorhanden ist, findet man auch dort ebenfalls diese Video-Register!

Die Digital/Analog-Wandler für die Umsetzung der digitalen Farbintensitätsinformation in ein analoges Intensitätssignal für jede Primärfarbe sind im TT-SHIFTER untergebracht. Für das Timing bekommt der TT-SHIFTER einen 32-MHz-Takt eingespeist, aus dem er dann die erforderlichen Synchronsignale erzeugt. Außerdem liefert der TT-SHIFTER, aus dem 32-MHz-Mastertakt abgeleitet, Takte von 16 MHz für den Rest des Systems, einen 4-MHz-Takt für die beiden MFPs und einen 2-MHz-Takt für z. B. den Sound-Chip und den DMA-Sound-SHIFTER. Des weiteren kommen vom TT-SHIFTER die H- und V-Synchronsignale und das Display-Enable-Signal!

Die Einbindung der Video-Hardware beim TT zeigt die Abbildung 4.5.

Die Video-Hardware-Register im TT

Aus Kompatibilitätsgründen zum ST(E) findet man für die drei ST-Auflösungen die zugehörigen Hardware-Register an der gleichen Stelle (Adresse) wie beim ST(E)! Zusätzlich kann man aber auch am obersten Ende des MC68030er-Adreßraums darauf zugreifen.

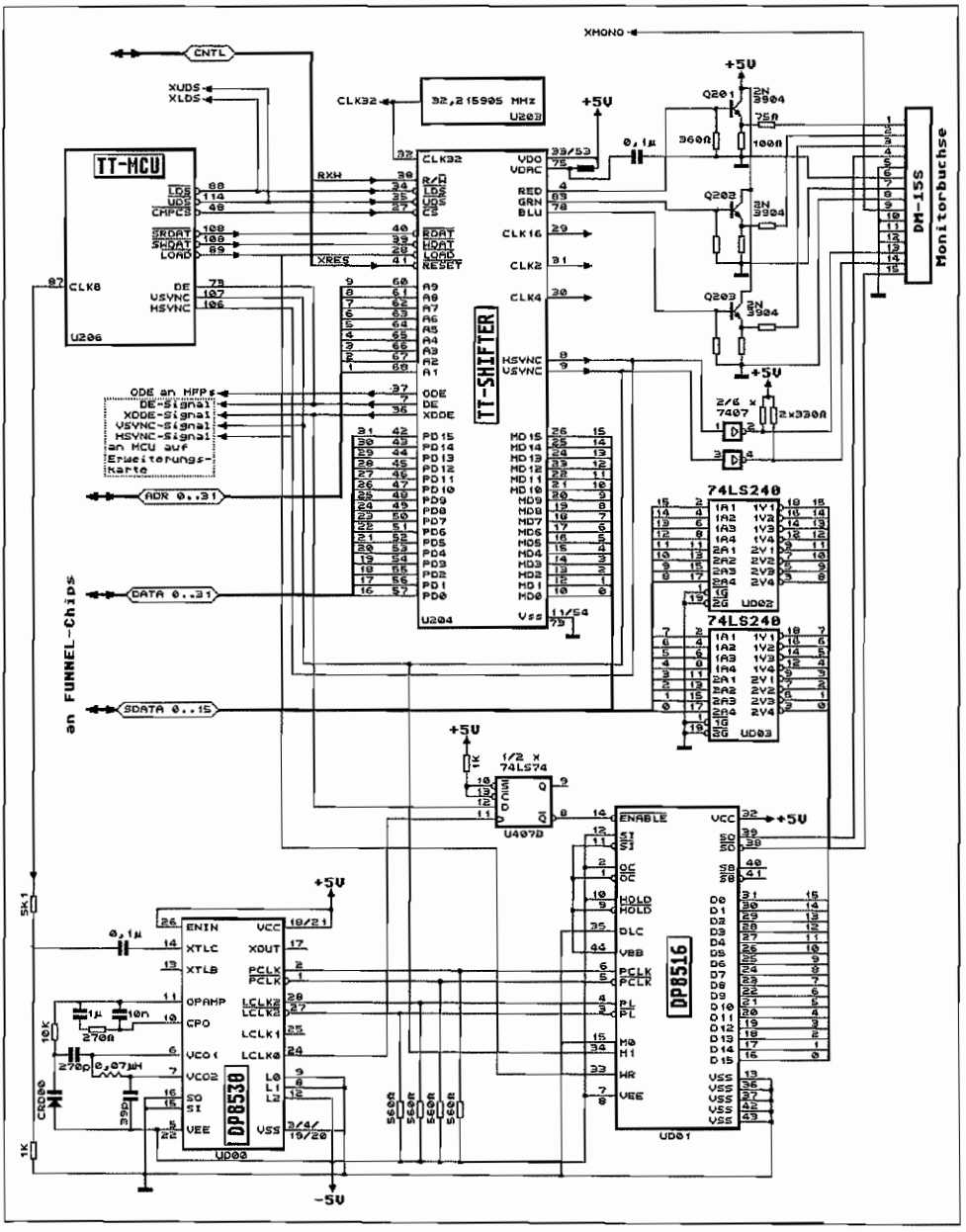


Abb. 4.5: Schaltungsauszug mit der TT-Video-Hardware

Video-Base-Register

Der Anfang des Bildschirmspeichers für alle Betriebsmodi wird der Video-Hardware über das *Video-Base-Register* mitgeteilt.

Adresse	Modus	Funktion	Label
$(FF)FF 8201	R/W	Video-Base-Reg. High-Byte	"dbaseh"
$(FF)FF 8203	R/W	Video-Base-Reg. Medium-Byte	"dbasel"
$(FF)FF 820D	R/W	Video-Base-Reg. Low-Byte	"dbaselow"

Beim Low-Byte der Video-Base-Adresse ("dbaselow" an Adr. $(FF)FF 820C) haben die Bits 0..2 keine Funktion, sprich: sie werden immer als 0 angenommen. Damit kann der Bildschirmspeicher im ST-RAM immer nur an einer 8-Byte-Schwelle liegen. Die Video-Hardware bearbeitet ja bei einem Speicherzugriff auch immer 64 Bits (8 Byte) auf einmal!

Video-Address-Counter

Die in diese drei Register einzuschreibende 24 Bit-Adresse wird bei jedem neuen Bildbeginn in den *Video-Adress-Counter* übertragen. Dort wird sie dann laufend hochgezählt, bis das Bild fertig aufgebaut ist.

Adresse	Modus	Funktion	Label
$(FF)FF 8205	R	Video-Address-Counter High-Byte	"vcounthi"
$(FF)FF 8207	R	Video-Address-Counter Medium-Byte	"vcountmid"
$(FF)FF 8209	R	Video-Address-Counter Low-Byte	"vcountlow"

Im Gegensatz zum STE können diese Register nur gelesen werden. Also keine großartigen Tricks mit "changing Video-Counter-Register on the fly!" für Display-Manipulationen während des Bildaufbaus möglich!

Sync-Mode-Register

Aus Kompatibilitätsgründen zum ST existiert dann noch das *ST-Sync-Mode-Register*, das sich aber gar nicht kompatibel zum ST(E) verhält.

Adresse	Modus	Funktion	Label
$(FF)FF 820A	R/W	ST-Sync-Mode-Register	"syncmode"

In diesem Register hat nur das Bit 0 überhaupt eine Funktion. Beim ST wurde dort durch Setzen des Bits 0 auf externe Synchronisation des STs umgeschaltet. Beim TT ist es genau umgekehrt. Wenn das Bit 0 gesetzt ist, erhält man ein Bild, sonst werden die Synchronsignale abgeschaltet!

ST-Farbregister

Aus Kompatibilitätsgründen existieren im TT an gewohnter Stelle wie im ST(E) die 16 Palettenregister zur Auswahl der Pixelfarbe. Die Bitbelegung für die drei Primärfarben ist die gleiche wie beim STE. Die unteren drei Bits jedes Nibbles weisen von rechts nach links steigende Wertigkeiten für den Intensitätswert auf. Allerdings hat dann das höchstwertige Bit eines Nibbles bei der Festlegung der Farbintensität die geringste Gewichtung (kompatibel zu STE-Palettenregistern)! Die nachfolgende Tabelle zeigt die Adreßlage der 16 STE-kompatiblen Palettenregister im TT und ihre Bitbelegung.

Adresse	Modus	Bitbelegung	Label
$(FF)FF 8240	R/W	XXXX rRRR gGGG bBBB	"color0"
$(FF)FF 8242	R/W	XXXX rRRR gGGG bBBB	"color1"
...	...	...	...
$(FF)FF 825C	R/W	XXXX rRRR gGGG bBBB	"color14"
$(FF)FF 825E	R/W	XXXX rRRR gGGG bBBB	"color15"

Anmerkung: Bitpositionen mit Kleinbuchstaben weisen auf das niedrigstwertige Farbwertbit hin!

Diese 16 ST(E)-Palettenregister sind jedoch nicht separat von den 256 TT-Farbregistern zu sehen, sondern stellen vielmehr ein Subset der TT-Farbregister dar. Durch die vier untersten Bits im TT-Shift-Mode-Register wird nämlich festgelegt, welche 16er-Registergruppe der 256 TT-Farbregister an den Adressen der ST-Farbregister auftaucht.

ST-Shift-Mode-Register

Ebenfalls aus Kompatibilitätsgründen existiert an (vom ST(E) her) gewohnter Adresse das ST-Shift-Mode-Register.

Hierüber wird die gewünschte Betriebsart für den TT-SHIFTER programmiert. Die Bitbelegung ist kompatibel zum ST(E).

Adresse	Modus	Funktion	Label
$(FF)FF 8260	R/W	ST-Shift-Mode-Register	"shiftmd"

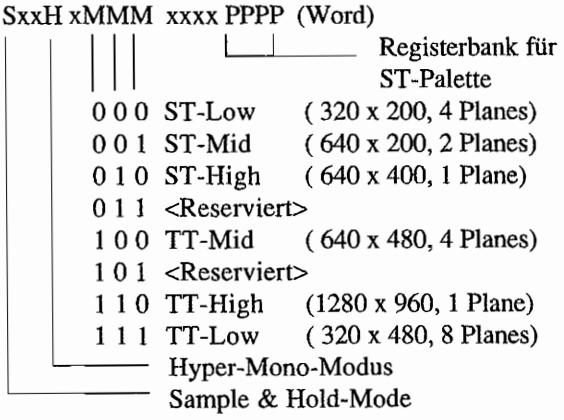
```
xxxx xxMM  xxxx xxxx  (Word)
         ||
         00  ST-Low      (320 x 200, 4 Planes)
         01  ST-Mid      (640 x 200, 2 Planes)
         10  ST-High     (640 x 400, 1 Plane)
         11  <Reserviert>
```

Allerdings tauchen bei Verwendung der TT-spezifischen Betriebsmodi deren Einstellungen ebenfalls in diesem Shift-Mode-Register auf! Das gilt sowohl für die Modus-Bits als auch für die Palettenbank. Man findet also die im TT-Shift-Mode-Register eingestellten Werte ebenfalls im ST-Shift-Mode-Register wieder!

TT-Shift-Mode-Register

Sowohl die ST-Grafikmodi als auch die speziellen TT-Betriebsarten lassen sich in diesem TT-eigenen Register einstellen. Die Bitbelegung ist die gleiche wie im ST-Shift-Mode-Register, nur daß halt noch ein paar Steuerbits hinzugekommen sind!

Adresse	Modus	Funktion	Label
$(FF)FF 8262	R/W	TT-Shift-Mode-Register	"shift_TT"



Mit den untersten vier Bits dieses Registers wird festgelegt, welcher Registersatz der 256 TT-Palette-Register für die 16 ST(E)-kompatiblen Farbpalettenregister verwendet wird. Ein Wert von Null wählt also die TT-Farbregister 0..15 für die ST(E)-Farbregister aus! Durch Ändern dieser vier Bits lassen sich alle 16 ST(E)-Farbregister "auf einen Schlag" verändern!

Mit Setzen von Bit 12 in diesem Register wird der *Hyper-Mono-Modus* eingeschaltet. Damit läßt sich der TT-SHIFTER auf einen speziellen Monochrom-Modus mit 256 Graustufen programmieren. Dabei werden im TT-SHIFTER die D/A-Wandler für den Grün- und Blau-Kanal zusammengefaßt. Dazu sind acht Intensitätsbits nötig. Die vier höchstwertigen Intensitätsbits kommen vom Grün-Kanal, das niederwertige Nibble aus den Palettenbits des Blau-Kanals.

Das *Sample & Hold-Bit* sorgt dafür, daß so eine Art Schmiereffekt auf dem Bildschirm erzielt wird. Anstelle der Hintergrundfarbe (Farbe 0) wird die zuletzt benutzte Farbe weiterverwendet. Einfach mal ausprobieren!

Die TT-Palette-Register

Diese 256 Farbregister können alle einzeln angesprochen und geändert werden. Jedoch nur die TT-Low-Betriebsart verwendet auch alle 256 Register bei der Darstellung. Die 256 Farbregister sind in 16 "Banks" zu je 16 Registern aufgeteilt, von denen dann immer eine Bank als ST-Farbregistersatz verwendet wird. Welche Bank das ist, wird mit den vier untersten Bits im TT-Shift-Mode-Register eingestellt.

Die Farbintensität kann mit jeweils vier Bits für jede der drei Primärfarben eingestellt werden. Dabei hat ATARI die Gewichtung der Bits aber wieder in eine vernünftigere Form gebracht (also nicht mehr Least significant Bit ganz links im Nibble wie im STE, sondern LSB ganz rechts und MSB ganz links im Nibble!). Da ja die TT-Palette-Register auch für die ST(E)-Palette-Register benutzt werden, muß diese unterschiedliche Behandlung der Intensitätssteuerung angepaßt werden. Das geschieht aber hardwaremäßig. Die Beziehung "Höchstwertiges Bit im STE-Farbnibble wird niedrigstwertiges Bit im TT-Farbnibble" und umgekehrt ist bereits per Hardware realisiert!

Adresse	Modus	Bitbelegung	Label
$(FF)FF 8400	R/W	XXXX RRRr GGGg BBBb	"TT_col0"
$(FF)FF 8402	R/W	XXXX RRRr GGGg BBBb	"TT_col1"
...	...	...	...
$(FF)FF 85FC	R/W	XXXX RRRr GGGg BBBb	"TT_col254"
$(FF)FF 85FE	R/W	XXXX RRRr GGGg BBBb	"TT_col255"

Betriebssystemunterstützung der Video-Hardware

Alle diese neuen Möglichkeiten der Videohardware wollen und sollen natürlich möglichst einfach und standardisiert genutzt sein. Deshalb hat ATARI eine Menge neuer XBIOS-Aufrufe im TT-TOS eingebaut, um dem Programmierer das direkte Manipulieren der Hardware-Register nach Möglichkeit zu ersparen.

Im Kapitel zum XBIOS sind diese neuen Aufrufe mit entsprechender Beschreibung zu finden.

Kapitel 5: Und weil's so schön war – Noch'n MFP im TT

Da ja im TT einiges an Hardware zusätzlich vorhanden ist, unter anderem eben auch zusätzliche serielle Schnittstellen, hat ATARI zur Unterstützung einen weiteren MFP-Baustein (MC68901) spendiert.

Der ST-MFP im TT

Der "alte" MFP im TT hat die gleichen Aufgaben wie im ST(E) übernommen und verhält sich auch kompatibel dazu. Das heißt unter anderem auch, daß durch die gleichen DMA-Sound-Fähigkeiten, wie schon im STE, eine "Doppelbelegung" des Portanschlusses IO7 vorgenommen wurde. Es werden ja darüber sowohl die Monochrom-Detect-Leitung des Monitors als auch das DMA-Sound-Aktiv-Signal abgefragt. Näheres dazu kann man im Anhang "Die Hardware des STE" nachschlagen. Auch beim TT gibt's natürlich das Monochrom-Detect-Signal, welches hier vom Pin 9 der Monitorbuchse stammt und mit dem DMA-Sound-Aktiv-Signal ver"Exklusiv Oder"t wird.

Da es beim TT allerdings keinen BLITTER gibt, ist die sonst an IO3 angeschlossene Interruptleitung des BLITTERs hier nicht vorhanden. ATARI hat dafür diesen Anschluß mit einem Signal CLKDIR (Clock Direction ???) belegt, dessen Bedeutung nicht unmittelbar aus der Schaltung ersichtlich ist. Es scheint aber in Verbindung mit dem Netzwerkinterface des TT zu stehen. Ich könnte mir vorstellen, daß man diesen Port als Ausgang benutzt, um damit (über ein PAL) dem SCC-Baustein auf Kanal A einen TT-internen Takt aufzuschalten.

Die Bezeichnung des Seriellen Ports, der mit dem ST-MFP realisiert und zum ST(E) kompatibel ist, lautet auf "Serial Port C" bzw. "MODEM 1" und ist auf eine 9polige Sub-D-Buchse zusammengeschrumpft. Auch hier ist wieder (wie beim ST(E)) der Soundchip bei den Handshakesignalen (RTS und DTR) beteiligt! Also alles beim alten geblieben.

Der "neue" MFP im TT

Durch den Einsatz von neuen DMA-fähigen Schnittstellen im TT, wie den SCSI-Port und die zwei seriellen Schnittstellen über den SCC (Serial Communications Controller vom Typ Z85C30), benötigt man natürlich entsprechende Meldeeingänge, um z. B. den Abschluß von DMA-Operationen signalisiert zu bekommen.

Also hat man einen zweiten MFP-Baustein verwendet, bei dem zusätzlich zu den interruptfähigen Eingangsports noch eine serielle Schnittstelle als "Abfallprodukt" herausschaute (wird als SERIAL 1 bezeichnet). Diese serielle Schnittstelle verfügt jedoch über keinerlei Steuer- und

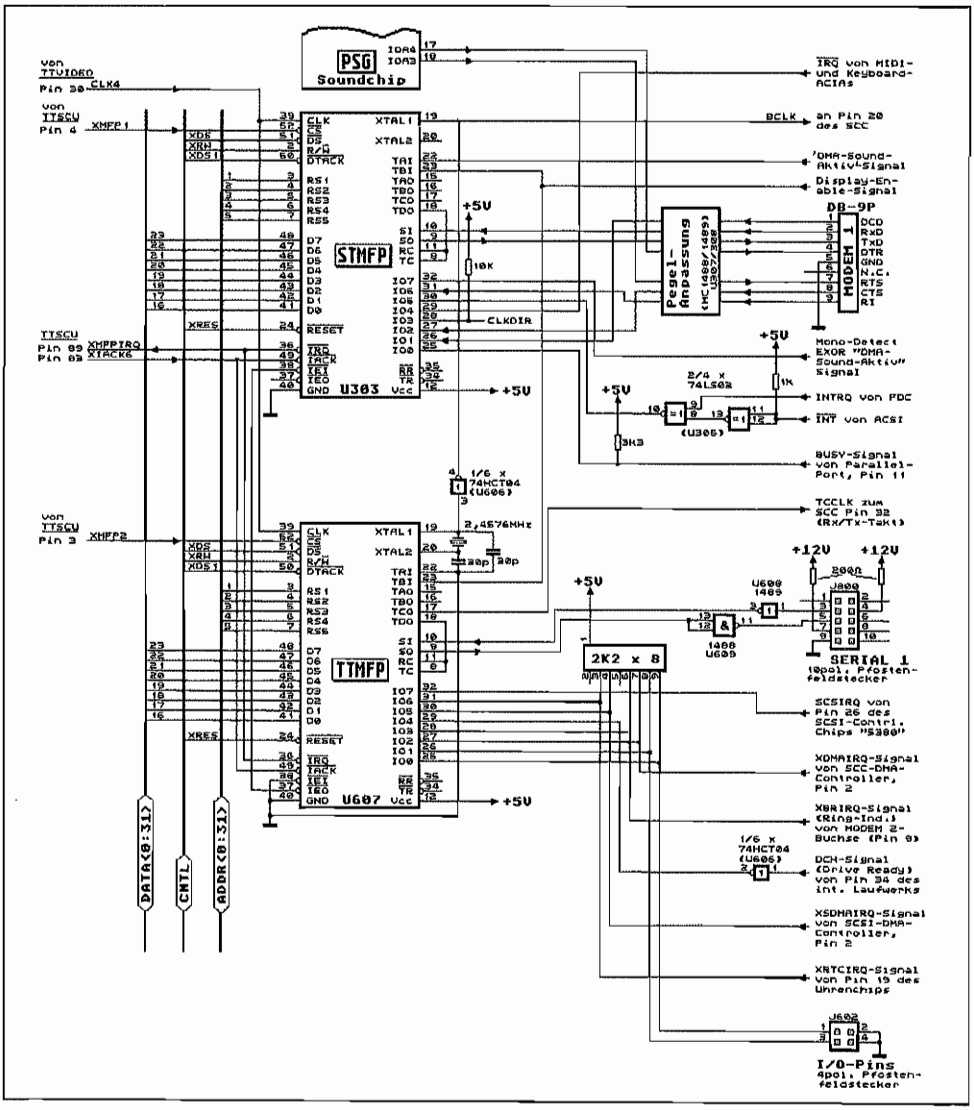


Abb. 5.1: Einbindung der beiden MFPs in die TT-Hardware

Meldeleitungen! Lediglich Empfangs- und Sendedatenleitung stehen zur Verfügung. Dieser zweite MFP (in Zukunft als TT-MFP bezeichnet) liegt schaltungsmäßig "parallel" zum bereits vorhandenen MFP. Diese Parallelschaltung bezieht sich jedoch nur auf die Anbindung zum Daten- und Adreßbus und die erforderlichen Leitungen zum Control-Bus. Allerdings gibt's schon noch kleine Abweichungen. Die Abbildung 5.1 zeigt die Einbindung der beiden MFP-Chips in das Hardwarekonzept des TT.

Die Chip-Select-Leitung ist bei den beiden MFPs natürlich nicht die gleiche, sondern der TT-MFP hat sich nur im Adreßbereich $FF FA81 .. $FF FAAF (bzw. $FFFF FA81 .. $FFFF FAAF) zu melden. Dabei liegen auch hier die Register (je 8 Bit breit!) des Bausteins auf ungeraden Adressen. Die Reihenfolge und Funktion der Register ist die gleiche wie beim ST-MFP, mit folgenden Ausnahmen:

Adresse	Zugriff	Funktion des Registers	Label
$FF FA81	**R/W**	**GPIP-Data-Register (GPIP)**	**"GPIP_TT"**

Es handelt sich zwar um ein I/O-Register, jedoch initialisiert das gegenwärtige TOS im TT alle TT-MFP-I/O-Ports als Eingänge. Man kann also über dieses Register die Zustände der MFP-Ports IO0..IO7 auslesen. Wenn man einzelne Ports auf Ausgang schalten will (insbesondere bei IO0 und IO1 interessant, da diese auf Pfostenfeldleisten (J602 auf dem Motherboard) geführt sind), muß man entsprechend die Bits 0 und 1 im Data-Direction-Register setzen! Durch Beschreiben des GPIP_TT-Registers werden die als Ausgänge programmierten Ports entsprechend gesetzt oder rückgesetzt.

Bit 0 und 1 signalisieren den Zustand der Anschlüsse IO0 und IO1. Diese sind auf dem TT-Motherboard auf Pfostensteckverbinder (J602) geführt und können für künftige Hardwareerweiterungen verwendet werden. Da die I/O-Ports des MFP sowohl als Ein- oder Ausgang programmiert werden können, läßt sich damit evtl. einiges anfangen.

Bit 2 ist das Interrupt-Request-Signal vom SCC-DMA-Baustein. Mit einem Low meldet damit der SCC-DMA-Kanal das Ende einer DMA-Operation.

Bit 3 gibt die Möglichkeit, das "Ring indicator"-Signal von der Buchse MODEM 2 auszuwerten. Mit einem Low wird hier das Setzen dieser Schnittstellenleitung signalisiert.

Adresse	Zugriff	Funktion des Registers	Label

Bit 4 kommt von Pin 34 des eingebauten Floppy-Disk-Laufwerks. Über diesen Anschluß zeigt die Floppy ihren Betriebszustand an. Manche Floppies liefern hier auch eine Information über einen Medienwechsel. Dieses Bit wird vom Betriebssystem aber nicht benutzt.

Bit 5 wird vom SCSI-DMA-Controller beeinflußt, der bei Auftreten eines Busfehlers während des DMA-Prozesses diesen Anschluß auf Low setzt.

Bit 6 ist ein Interrupt-Request (Low-aktiv) vom TT-Uhrenchip. Wann der Uhrenchip z. B. einen Interrupt auslösen kann, sollte man in dem entsprechenden Kapitel über den RTC 146818 nachlesen.

Bit 7 geht auf High (!), wenn der SCSI-Controller-Chip des TT (Typ 5380) einen Interrupt auslöst! Sowohl das TOS als auch Harddisktreiber verwenden dieses Bit (allerdings nur im Polling-Verfahren), um festzustellen, ob die gewünschte SCSI-Operation beendet wurde. Also hat es eine ähnliche Funktion wie das Bit 5 beim ST-MFP für die Fertigmeldung von FDC/ACSI-Bus!

$FF FA83 **R/W** **Active-Edge-Register (AER)** **"AER_TT"**

Für jedes Bit des I/O-Ports läßt sich hier einstellen, bei welchem Flankenwechsel an einem Port ein Interrupt ausgelöst werden soll (wenn er denn über das Interrupt-Enable-Register eingeschaltet ist!).

Ein gesetztes Bit veranlaßt eine Interrupt-Anforderung bei Low->High-Übergang am entsprechenden Portanschluß. Wann bei einem gelöschten Bit unterbrochen wird, können Sie sich jetzt, glaube ich, selbst denken!

$FF FA85 **R/W** **Data-Direction-Register (DDR)** **"DDR_TT"**

Für jeden I/O-Portanschluß existiert ein Bit. Bei gesetztem Bit ist der entsprechende Port als Ausgang programmiert. Bei gelöschtem Bit fungiert der zugehörige Port als Eingang. (Das TT-TOS setzt im Moment alle Ports als Eingänge!)

Die Timer des TT-MFP

Die Funktionsweise der Timer kann im Kapitel über den MFP im ST nachgeschlagen werden. Es sollen hier und auf der folgenden Seite nur kurz die Unterschiede aufgezeigt werden.

- Der Timer-A-Eingang liegt fest auf Masse. Damit fallen also die Betriebsart Pulsbreitenmessung (wen interessiert schon die Zeitdauer eines immer anliegenden Low-Signals?) und die Ereigniszählung (es ereignet sich halt wenig auf dem Massepotential) als Anwendung weg!

  Bliebe also nur noch der Delay Mode übrig. Vom TOS wird der Timer A jedoch nicht benutzt!

- Der Timer-B-Eingang liegt zusammen mit dem gleichen Eingang des ST-MFP am Display-Enable-Signal. Damit kann also bei Bedarf mit diesem Timer ebenfalls ein Zähler für Bildschirmzeilen realisiert werden.

Die Adressen der Timer-Register beim TT-MFP:

Adresse	Modus	Belegung	Funktion	Label
$FF FA99	**R/W**	**---RCCCC**	**Timer-A-Control-Reg.**	**('TACR_TT')**
$FF FA9B	**R/W**	**---RCCCC**	**Timer-B-Control-Reg.**	**('TBCR_TT')**

||||	**Modus**	**Vorteiler**
	0000	Stop	–
	0001	Delay	1 : 4
	0010	Delay	1 : 10
	0011	Delay	1 : 16
	0100	Delay	1 : 50
	0101	Delay	1 : 64
	0110	Delay	1 : 100
	0111	Delay	1 : 200
	1000	Ereignis	–
	1001	Pulsbreite	1 : 4
	1010	Pulsbreite	1 : 10
	1011	Pulsbreite	1 : 16
	1100	Pulsbreite	1 : 50
	1101	Pulsbreite	1 : 64
	1110	Pulsbreite	1 : 100
	1111	Pulsbreite	1 : 200
R		Timer A- bzw. B-Ausgang auf Low	

Adresse	Modus	Belegung	Funktion		Label
$FF FA9D	**R/W**	**-CCC-DDD**	**Timer C+D Control Reg.**		**('TCDCR_TT')**
		||| |||	(C: Takt für SCC-Channel 2)		
			(D: Baudrate für SERIAL 1)		
			Modus	**Vorteiler**	
		000-000	STOP	–	
		001-001	Delay	1 : 4	
		010-010	Delay	1 : 10	
		011-011	Delay	1 : 16	
		100-100	Delay	1 : 50	
		101-101	Delay	1 : 64	
		110-110	Delay	1 : 100	
		111-111	Delay	1 : 200	
	Timer:	(C) (D)			
$FF FA9F	**R/W**	**DDDDDDDD**	**Timer-A-Data-Register**		**('TADR_TT')**
$FF FAA1	**R/W**	**DDDDDDDD**	**Timer-B-Data-Register**		**('TBDR_TT')**
$FF FAA3	**R/W**	**DDDDDDDD**	**Timer-C-Data-Register**		**('TCDR_TT')**
$FF FAA5	**R/W**	**DDDDDDDD**	**Timer-D-Data-Register**		**('TDDR_TT')**

Weitere Unterschiede:

- Timer-C des TT-MFP kann als Taktgenerator für den Kanal B des neuen SCC-Chips verwendet werden. Das TOS stellt im Timer-C des TT-MFP eine Frequenz von ca. 12,5 kHz am Timerausgang ein (Timer-C mit 1:4-Vorteiler und Timer-C-Data-Wert von 25!). Der Kanal B des SCC kann allerdings mit unterschiedlichen Taktraten für Sende- und Empfangsrichtung betrieben werden. Mehr dazu bei der Beschreibung des SCC!

- Genau wie beim ST-MFP kann der Timer D des TT-MFP als Baudratengenerator für den seriellen Port des MFP verwendet werden. Da der zugrundeliegende Takt für den TT-MFP der gleiche ist wie beim ST-MFP, kann man auch hier Baudraten bis 19200 Bd einstellen.

Interrupts durch den TT-MFP

Der MFP kann ja ebenfalls als Interruptquelle in einem Mikrocomputersystem auftreten (siehe dazu auch die Erläuterungen zu den MFP-Interrupts des STs!). Diese Möglichkeit wollte sich ATARI beim TT natürlich für den TT-MFP ebenfalls offenhalten. So sind denn im TT beide MFPs, was die Interruptbehandlung angeht, "hintereinander gekettet". Für diese Verkettung

besitzt der MFP die Anschlüsse _IEI und _IEO. Eine Interruptanforderung stellt der MFP mit einem Low-Signal auf dem _IRQ-Anschluß (Interrupt Request). Diesbezüglich sind die beiden MFPs parallelgeschaltet. D. h., der MFP, der einen Interrupt anfordert, zieht die _IRQ-Leitung auf Low. Daran erkennt die TTSCU (TT-System Control Unit) eine MFP-Interruptanforderung.

Während des nun folgenden Interrupt-Bestätigungszyklusses, der durch die CPU durchgeführt wird, erhalten die _IACK-Anschlüsse der beiden MFPs dieses Bestätigungssignal zugeführt. Welcher MFP nun "seinen" Interrupt bearbeitet bekommt (also höher priorisiert ist), wird durch dessen Position in der _IEI -> _IEO-Verkettung festgelegt. Nur jener MFP darf nun auf eine Interruptbestätigung reagieren, dessen _IEI-Anschluß auf Low liegt. Als Folge davon setzt dieser MFP seinen _IEO-Anschluß auf High, und der in der Kette dahinter angeschlossene MFP hat sich aus dieser Interruptbestätigung herauszuhalten!

Der Schaltungsauszug mit den beiden MFPs im TT zeigt für diese Verkettung nun an, daß der TT-MFP infolge seines immer aktivierten _IEI-Anschlusses (fest auf Low gelegt!) höher priorisiert ist. Nur wenn vom TT-MFP keine Interruptanforderung vorliegt, wird bei einem Interruptbestätigungszyklus an dessen _IEO-Ausgang ein Low liegen und damit dem ST-MFP die Möglichkeit gegeben, "seinen" Interrupt bearbeitet zu bekommen!

Jeder MFP besitzt 16 Interruptkanäle, die untereinander unterschiedlich gewichtet sind. Alle acht I/O-Ports und die vier Timer kommen als mögliche Interruptquellen in Frage. Außerdem sind noch 4 Interruptquellen bei der seriellen Schnittstelle des MFP zu finden. Die Priorisierung der Interrupts ist bei jedem MFP gleich und kann deshalb im Kapitel "Der Multifunktionsbaustein MFP 68901" im Hardwareteil zum ST nachgeschlagen werden.

Der MFP übergibt während des Interrupt-Bestätigungszyklusses der CPU eine eigene 8 Bit breite Vektornummer (Non-Autovektor-Interrupt Prinzip). Daraus kann die CPU dann, nach Multiplikation der Vektornummer mit 4, die Adresse des zugehörigen Interruptvektors im RAM ermitteln. Damit mehrere MFPs in einem System unterschiedliche Vektornummern erzeugen können, ist das High-Nibble der MFP-Vektornummer im Interrupt-Vektor-Register eines MFPs einstellbar. Das Low-Nibble ergibt sich dann aus der Kanalnummer, welcher den Interrupt ausgelöst hat.

Im TT sorgt das TOS durch Einstellen des Interrupt-Vektor-Registers im ST-MFP dafür, daß alle Vektornummern mit dem Nibble $4 beginnen und damit die zugehörigen Interruptvektoren ab Adresse $40 x 4 = $100 im RAM zu finden sind (wie auch beim ST(E)). Für den TT-MFP beginnen alle Vektornummern mit $5, und deshalb findet man die zugehörigen 16 Interruptvektoren unmittelbar anschließend an die ST-MFP-Interruptvektoren, ab Adresse $140 im Speicher. Die nachfolgende Abbildung zeigt noch mal kurz die Lage der TT-MFP-Interruptvektoren im RAM:

Adresse	MFP-Int. Nummer:	Status	Kurzbeschreibg. der Interruptquelle	MFP-Interrupt
$140	0	maskiert	I/O-Pin 1, J602 auf Motherboard	I/O-Port 0 (I00)
$144	1	maskiert	I/O-Pin 3, J602 auf Motherboard	I/O-Port 1 (I01)
$148	2	maskiert	XDMAIRQ von SCC-DMA-Controller	I/O-Port 2 (I02)
$14C	3	maskiert	XBRIRQ von RI-Anschluß, MODEM 2	I/O-Port 3 (I03)
$150	4	maskiert	Baudrate für SERIAL 1-Port	Timer D
$154	5	maskiert	TCCLK-Takt an SCC, Channel 2	Timer C
$158	6	maskiert	DCH (Drive ready) v. int. Floppy	I/O-Port 4 (I04)
$15C	7	maskiert	XSDMAIRQ von SCSI-DMA-Controller	I/O-Port 5 (I05)
$160	8	maskiert	Display-Enable-Signal	Timer B
$164	9	aktiv.	Sendefehler ser. Port des TT-MFP	XMIT-Error
$168	10	aktiv.	Sendepuffer leer (")	XMIT-Buf. empty
$16C	11	aktiv.	Empfangsfehler (")	RCV-Error
$170	12	aktiv	Empfangspuff. voll (")	RCV-Buffer full
$174	13	maskiert	Nicht benutzt	Timer A
$178	14	maskiert	Interrupt von TT-Uhrenchip	I/O-Port 6 (I06)
$17C	15	maskiert	Interrupt von SCSI-Chip "5380"	I/O-Port 7 (I07)

Abb. 5.2: Die Lage der TT-MFP-Interruptvektoren im RAM

Die Interrupt-Steuerregister des TT-MFP

Die vier Register des TT-MFP, die zur Steuerung der Interruptbehandlung durch den MFP verwendet werden, sind natürlich vollkommen identisch zu denen des bereits vom ST her bekannten MFP. Lediglich die Interruptquellen sind bei den I/O-Ports andere Signale aus dem "Inneren" des TT. Da die Signalbelegung ja sowohl aus dem Schaltungsauszug als auch der Tabelle mit den Interruptvektoren hervorgeht, soll hier nur noch kurz auf diese MFP-Register eingegangen werden. Die Wirkungsweise der Bits dieser einzelnen Register kann (sinngemäß) in ausführlicher Form im Kapitel über den ST-MFP bei der ST-Hardware nachgelesen werden.

Adresse	Modus	Belegung	Funktion	Label
$FF FA87	**R/W**	**eeeEEEEe**	**Interrupt-Enable-Reg. A**	**('IERA-TT')**

- Timer B (Display enable Signal)
- XMIT Error (RS 232)
- XMIT Buffer empty (RS 232)
- RCV Error (RS 232)
- RCV Buffer full (RS 232)
- Timer A (nicht benutzt)
- Port I6 (TT-Uhrenchip-IRQ-Anschluß)
- Port I7 (IRQ von SCSI-Contr.-Chip)

Adresse	Modus	Belegung	Funktion	Label
$FF FA89	**R/W**	**eeEeeeee**	**Interrupt-Enable-Reg. B**	**('IERB_TT')**

Port I0 (Pin 1 des J602-Steckers)
Port I1 (Pin 3 des J602-Steckers)
Port I2 (IRQ von SCC-DMA-Baustein)
Port I3 (Ring indicator von MODEM 2)
Timer D (Baudraten-Gen. SERIAL 1)
Timer C (TCCLK an SCC-Channel 2)
Port I4 (DCH-Signal v. P.34 int. Floppy)
Port I5 (IRQ von SCSI-DMA-Baustein)

$FF FA8B	**R/W**	**PPPPPPPP**	**Int.-Pending-Register A**	**('IPRA_TT')**
			(Bitbelegung siehe 'IERA_TT')	
$FF FA8D	**R/W**	**PPPPPPPP**	**Int.-Pending-Register B**	**('IPRB_TT')**
			(Bitbelegung siehe 'IERB_TT')	
$FF FA8F	**R/W**	**SSSSSSSS**	**Int.-In-Service-Register A**	**('ISRA_TT')**
			(Bitbelegung siehe 'IERA_TT')	
$FF FA91	**R/W**	**SSSSSSSS**	**Int.-In-Service-Register B**	**('ISRB_TT')**
			(Bitbelegung siehe 'IERB_TT')	
$FF FA93	**R/W**	**mmmMMMMm**	**Interrupt-Mask-Register A**	**('IMRA_TT')**
			(Bitbelegung siehe 'IERA_TT')	
$FF FA95	**R/W**	**mmmmmmmm**	**Interrupt-Mask-Register B**	**('IMRB_TT')**
			(Bitbelegung siehe 'IERB_TT')	
$FF FA97	**R/W**	**VVVVS---**	**Interrupt-Vektor-Register**	**('VR_TT')**

End-of-Interrupt Modus
High-Nibble der Vektornummer (=5)

Anmerkung: Die vom TOS vorgenommenen Standardeinstellungen in den Interrupt-Enable- und Interrupt-Mask-Registern sind durch Klein- bzw. Großbuchstaben bei der jeweiligen Bitposition angegeben. Ein Großbuchstabe repräsentiert dabei ein gesetztes Bit!

Dieser Zusammenfassung kann man also entnehmen, daß nur die Interrupts für die serielle Schnittstelle des TT-MFP und den Timer-C "enabled" sind. Außerdem sind im Mask-Register alle Interrupts bis auf die der seriellen Schnittstelle ausmaskiert. Ähnlich ist es bei den zugehörigen Interrupt-Service-Routinen. Nur die als "Enabled" eingestellten Interrupts haben auch einen sinnvollen Interruptvektor.

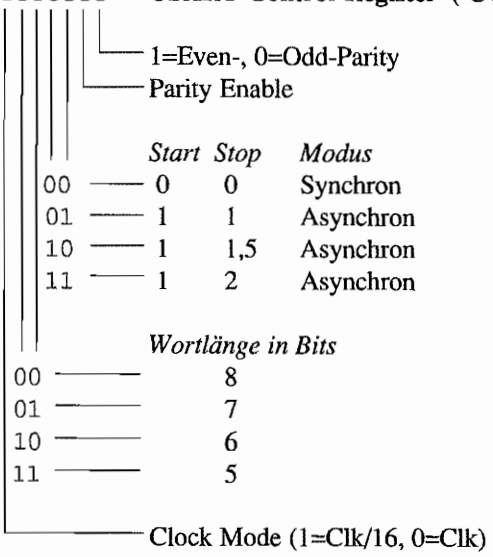
Auch der TT-MFP wird im Software-End-of-Interrupt-Modus betrieben. Das heißt, daß die zugehörige Interrupt-Service-Routine nach Abarbeitung das entsprechende Interrupt-In-Service-Bit zurücksetzen muß!

Die Steuerregister für die serielle Schnittstelle des TT-MFP

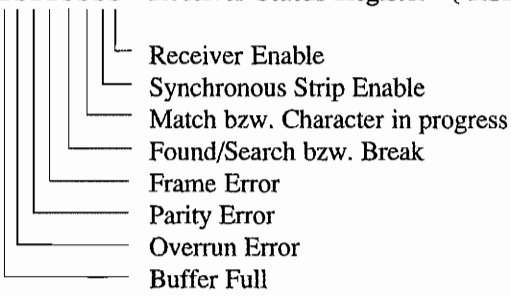
Die Funktion der Register für die serielle Schnittstelle SERIAL 1, die mit dem TT-MFP realisiert wird, ist schon beim ST-MFP erläutert worden.

Deshalb hier nur eine kurze Registerübersicht mit den entsprechenden Adressen:

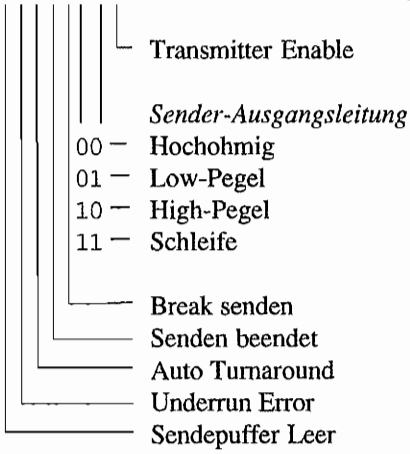
Adresse	Modus	Belegung	Funktion	Label
$FF FAA7	**R/W**	**SSSSSSSS**	**Sync.-Char. Register**	**('SCR_TT')**
$FF FAA9	**R/W**	**CCCCCCC-**	**USART-Control-Register**	**('UCR_TT')**



Adresse	Modus	Belegung	Funktion	Label
$FF FAAB	**R/W**	SSSSSSSS	**Receiver-Status-Register**	**('RSR_TT')**



$FF FAAD	**R/W**	SSSSSSSS	**Transmitter-Status-Register**	**('TSR_TT')**



Adresse	Modus	Belegung	Funktion	Label
$FF FAAF	**R/W**	DDDDDDDD	**USART-Data-Register** (Lesen holt Byte aus Empfangspuffer) (Schreiben bringt Byte in Sendepuffer)	**('UDR_TT')**

Alle Adreßangaben zu MFP-TT-Registern sind für den ST-Adreßraum gemacht. Die gleichen Register sind auch im oberen TT-Adreßraum unter $FFFF FA8X .. $FFFF FAFF erreichbar! Soviel zu den MFPs im TT!

Kapitel 6: Ein Schritt in die richtige Richtung – Der TT-SCSI-Port

Schon bei der ST(E)-Serie hat sich ATARI um eine Schnittstelle für schnelle Peripheriegeräte (Festplatten, Laserdrucker) eigene Gedanken gemacht. Herausgekommen ist dann dabei eine zu nichts (außer manchmal zu sich selbst) kompatible Schnittstelle, der ACSI-Bus. Dieser ähnelt in Ansätzen dem bei Kleincomputersystemen schon weit verbeiteten SCSI-Bus (*S*mall *C*omputer *S*ystems *I*nterface).

Beim TT hat man (= ATARI) sich dann wohl überlegt, daß man diese Performance-Bremse für Festplatten, den Plattencontroller bzw. beim Anschluß von SCSI-Platten, den als Interface erforderlichen Host-Adapter, besser weglassen sollte. Schließlich soll der TT ja auch mal ein "großes" Betriebssystem (UNIX) unterstützen, und dazu sind schnelle Massenspeicher natürlich eine sehr wichtige Voraussetzung! Also findet sich im TT nun auch ein sehr bekannter SCSI-Controller-Chip (der 5380), der von einem eigenen DMA-Controller beim Datentransfer unterstützt wird. Bevor hier auf die Interna (sprich: Register, deren Adressen und die Funktionen) des SCSI-und des DMA-Controllers eingegangen wird, erst mal ein Überblick über den SCSI-Bus.

Der SCSI-Bus

Es handelt sich hierbei um einen bidirektionalen 8-Bit-Bus, an dem bis zu acht Teilnehmer angeschlossen sein können. Von der SCSI-Bus-Spezifikation her sind diese Teilnehmer mit "Intelligenz" ausgestattet und können sowohl Computer als auch Controller zur Steuerung von Peripheriegeräten sein. Dabei können durchaus auch mehrere Computer am SCSI-Bus "hängen". Ein Peripherie-Controller kann seinerseits nochmal bis zu acht Geräte bedienen (bei SCSI-Befehlen werden diese "Untergeräte" als "Logical Units" behandelt)!

Vom Prinzip her kann jeder Busteilnehmer als Initiator oder als Target arbeiten. Die Funktion ist dabei die gleiche, wie schon beim ACSI-Bus beschrieben. Ein Initiator fordert von einem anderen Teilnehmer (dem Target) eine Reaktion an (in der Regel einen Datenaustausch). In der Praxis findet man jedoch Computer mit Host-Adaptern als Initiatoren und periphere Geräte wie Festplatten, Laserdrucker, CD-ROMs usw. als Targets.

Damit es kein Durcheinander auf dem alle verbindenden Bus gibt, dürfen zur gleichen Zeit immer nur ein Initiator und ein Target miteinander kommunizieren. Der Informationsaustausch erfolgt asynchron mittels eines Request/Acknowledge-Protokolls. Es wird immer je ein Byte

übertragen und mittels des Handshakes bestätigt. (Es existiert allerdings eine Option für synchrone Transfers, die auch von einigen SCSI-Festplatten inzwischen unterstützt werden.) Außerdem haben alle angeschlossenen Busdevices eine eigene Priorität, welche über ein eigenes Bit dargestellt wird. In einer sogenannten Arbitrations-Phase (Entscheidungsphase) präsentieren alle "Mitbewerber" um die Buszuteilung ihr Prioritätsbit (wegen der 8-Bit-Busbreite kann es also nur acht Geräte geben, die sich um die Buskontrolle streiten!). Das Device mit der höchsten Priorität "gewinnt" den Bus und kann seinerseits nun mit einem anderen Device in Verbindung treten.

Die Busverbindung

Verwendet wird zur Verbindung der einzelnen Bus-Teilnehmer (meistens) ein 50poliges Flachbandkabel mit entsprechendem Pfostenstecker. Jeweils gegenüberliegende Pins der Steckverbinder bilden dabei ein Pärchen (und damit auch die daran angeschlossenen Leitungen). Bis auf eine Ausnahme ist je eine Leitung eines solchen Adernpaares bei der "single-ended-drivers"-Version als Masseleitung ausgeführt und dient somit als abschirmendes Element für die signalführende Leitung.

Bis zu Leitungslängen von 6 m sieht der SCSI-Standard diese Ausführung als sogenannte "single-ended-drivers"-Version (eine Leitung liegt mit "einem Ende" fest auf Masse). Die "differential-drivers"-Ausführung kommt mit ebenfalls 50 Leitungen aus, jedoch werden zusammengehörige Adernpaare mit speziellen Differential-Treiberstufen angesteuert, um die Störsicherheit heraufzusetzen und längere Entfernungen (bis zu 25 m) zu überbrücken.

Damit nicht der Eindruck entsteht, Atari mangele es an eigenen Ideen für Steckverbindern, hat man am TT den herausgeführten SCSI-Portanschluß auf einer "normalen" 25poligen Sub-D-Buchse enden lassen (das spart jede Masse Masse!). Dabei weist diese Belegung eine "verblüffende Ähnlichkeit" (sprich: Übereinstimmung) mit dem SCSI-Port bei MACINTOSH-Rechnern auf!

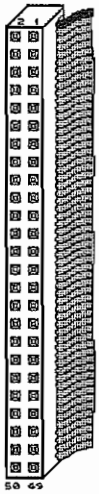
Das Ende der Busverbindung

An den beiden äußeren Enden einer Busverbindung wird der Bus durch Widerstandsnetzwerke abgeschlossen. Bei der (am häufigsten verwendeten) Single-ended-Driver-Version wird jede Signalleitung mit 330 Ω gegen Masse und 220 Ω an +5V abgeschlossen. Dafür verwendet man in der Regel Widerstandsnetzwerke in Steckfassungen, um bei Bedarf (wenn das Gerät z. B. nicht am Ende des Busses "hängt") die Terminierung entfernen zu können. Optional existiert für diese Busterminierung eine eigene Terminator-Power-Leitung (kurz: Termpwr). Damit ist es möglich, alle Abschlußnetzwerke aus einer Quelle (+5V) zu speisen, unabhängig davon, ob die am Bus angeschlossenen Geräte eingeschaltet sind!

Die Signale auf dem SCSI-Bus

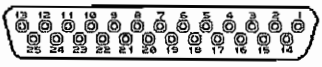
Alle Signale auf dem Bus sind Low-aktiv! Die Belegung des SCSI-Bus-Anschlusses zeigt die Abbildung 6.1. Als Signalpegel werden TTL-Pegel verwendet.

Steckverbinder am SCSI-Bus



Belegung	Pin-Nr.:		Belegung
Data 0	2	1	Masse
Data 1	4	3	Masse
Data 2	6	5	Masse
Data 3	8	7	Masse
Data 4	10	9	Masse
Data 5	12	11	Masse
Data 6	14	13	Masse
Data 7	16	15	Masse
Parity	18	17	Masse
---	20	19	Masse
---	22	21	Masse
---	24	23	Masse
(Termpwr)	26	25	N.C.
---	28	27	Masse
---	30	29	Masse
ATN	32	31	Masse
---	34	33	Masse
BSY	36	36	Masse
ACK	38	37	Masse
RESET	40	39	Masse
MSG	42	41	Masse
SEL	44	43	Masse
C/D	46	45	Masse
REQ	48	47	Masse
I/O	50	49	Masse

Außenansicht auf die SCSI-Buchse des TT



1 - REQ	14 - Masse
2 - MSG	15 - C/D
3 - I/O	16 - Masse
4 - RESET	17 - ATN
5 - ACK	18 - Masse
6 - BSY	19 - SEL
7 - Masse	20 - Parity
8 - Data 0	21 - Data 1
9 - Masse	22 - Data 2
10 - Data 3	23 - Data 4
11 - Data 5	24 - Masse
12 - Data 6	25 - N.C.
13 - Data 7	

Abb. 6.1: Die Belegung beim SCSI-Bus

Ein Low an einem Ausgangsanschluß soll zwischen 0..0,4V liegen. Dabei muß ein Ausgang bei Low-Signal einen Strom von 48mA "verkraften" können!

An einem Buseingang werden Spannungspegel bis 0,8V noch als Low "angesehen". Die Last, mit der ein Eingang den Bus bei Low belastet, soll 0,4mA nicht überschreiten.

High-Signale liegen zwischen 2,5V und 5,25V an Ausgangsanschlüssen. Bei Eingängen werden Signale zwischen 2V und 5,25V als High angesehen.

Eine Speisung für die Termpwr-Leitung soll bei einer Spannung zwischen 4V..5,25V einen Strom von 800mA liefern können.

Verwendete Leitungen

Da im Prinzip alle Geräte sowohl auf den Bus ausgeben als auch von dort Daten aufnehmen können, empfiehlt sich für Ausgänge das Open-Collector-Prinzip.

Reset (_RST) Pin 40 TT Pin 4 Bidirektional!

Wenn dieses Signal gesetzt wird (auf *Low* geht), werden die laufenden Busaktivitäten beendet. Alle Teilnehmer werden in den Grundzustand zurückversetzt.

Jeder Teilnehmer kann Reset auslösen. Die sogenannte "Reset hold time" (die Zeit, für die die _RST-Leitung mind. Low sein muß) beträgt 25ms.

Busy (_BSY) Pin 36 TT Pin 6 Bidirektional!

Der Bus ist schon abgefahren! Hiermit wird signalisiert, daß der Bus zur Zeit belegt ist. Auch während der Arbitration (Entscheidung, wer den Bus bekommt) geht Busy auf Low!

Select (_SEL) Pin 44 TT Pin 19 Initiator -> Target
Bei Reselect: Target -> Initiator

Der Initiator signalisiert damit den Targets, doch bitte mal nachzusehen, ob sich nicht vielleicht jemand durch die ebenfalls vom Initiator auf dem Datenbus ausgegebene Adresse (ID-Bit) angesprochen fühlen könnte (Targetauswahl). Ein Target benutzt dieses Signal, um sich in der Reselect-Phase wieder bei seinem Initiator zu melden (zum Beispiel eine Harddisk, die mit dem Formatieren fertig und nun bereit zu neuen Taten ist.)

Attention (_ATN) Pin 32 TT Pin 17 Initiator -> Target

Achtung! Der Initiator hat eine Message (Nachricht) für das Target. Da allerdings das Target bei Command-, Data-, Status- und Message-Transfers das "Heft in der Hand hält", also den Bus steuert, fordert der Initiator mit diesem Signal eine "Message out"-Phase beim Target an.

Message (_MSG) Pin 42 TT Pin 2 Target -> Initiator

Das Target signalisiert damit den Transfer einer Mitteilung (die Message-Richtung wird ebenfalls vom Target bestimmt!)

Control/Data (_C/D) Pin 46 TT Pin 15 Target -> Initiator

Das Target bestimmt hiermit, ob es sich bei den Signalen auf dem Datenbus um Befehle, Statusinformationen oder Messages (Control-Informations) handelt (Low-Signal).

Während der eigentlichen Datenübertragungsphase (Data-Phase) liegt dann hier ein High-Signal.

In-/Output (_I/O) Pin 50 TT Pin 3 Target -> Initiator

Das Target gibt hierüber die Transportrichtung auf dem Datenbus bekannt (High: Initiator -> Target; Low: Target -> Initiator).

Request (_REQ) Pin 48 TT Pin 1 Target -> Initiator

Für den Handshake während des Informationsaustauschs über den Datenbus fordert das Target mit diesem Signal den Initiator auf, ein Datenbyte auf den Bus zu legen (Initiator -> Target) oder daß das Target ein Byte auf den Bus gelegt hat (Target -> Initiator). Die Transferrichtung wird ja durch das _I/O-Signal bestimmt!

Acknwldg. (_ACK) Pin 38 TT Pin 5 Initiator -> Target

Das Gegenstück zum _REQ-Signal. Hiermit bestätigt der Initiator, daß er das angeforderte Datenbyte auf den Datenbus gelegt (Initiator an Target) oder daß er das vom Target auf den Bus gelegte Byte eingelesen hat (Target an Initiator)!

Datenbus (DB7..0) Pin 16, 14, 12, 10, 8, 6, 4, 2
TT Pin 13, 12, 11, 23, 10, 22, 21, 8 Bidirektional

Hierüber werden die Daten byteweise zwischen Target und Initiator verschoben. Die erreichbare Geschwindigkeit liegt laut "SCSI-Norm" bei max. 4 MByte/s.

Parität (DP) Pin 18 TT Pin 20 Bidirektional

Zusätzlich kann die Byteübertragung auf dem Bus mit einem ungeraden Paritätsbit gesichert werden. Wenn mit Parität gearbeitet wird, müssen auch alle angeschlossenen Devices diese Option verwenden. Beim TT wird diese Option z. Zt. nicht verwendet!

Die Bus-Phasen

Der Bus kann sich immer nur in einer von vier Hauptphasen befinden. Dabei werden diese Hauptphasen zum Teil noch in weitere Unterphasen unterschieden.

Alle Phasen werden durch bestimmte Zustände der Signale Busy, Message, Select, Input/Output, Control/Data, Request und Acknowledge definiert.

BUS FREE

Diese Phase ist dadurch charakterisiert, daß zur Zeit kein Gerät den Bus benutzt. Alle Bussignale sind inaktiv. Erkennen kann man diese Phase daran, daß Busy und Select deaktiviert (= High) sind!

Die angeschlossenen Geräte sollen alle Signale innerhalb des "Bus Clear Delay" (max. 800ns) deaktivieren, wenn nach Ablauf eines "Bus Settle Delay" (max. 400ns) Busy und Select auf High gegangen sind.

ARBITRATION

Diese zweite Hauptphase ist nur dann anzutreffen, wenn ein System mit mehr als einem Initiator betrieben wird. Dann muß nämlich entschieden (to arbitrate = entscheiden) werden, welcher Initiator die Buskontrolle bekommt. Da die meisten Anwendungen beim TT z. Zt. noch der Betrieb von SCSI-Platten sind und der TT der einzige Initiator im System ist, hat man die Arbitration-Phase bei der zugehörigen Treibersoftware noch nicht implementiert. Der SCSI-Controller-Chip im TT ist dazu jedoch fähig!

Wie bekommt nun ein Initiator die Kontrolle über den Bus?

- Warten, bis der Bus frei ist. Dazu müssen Busy *und* Select mindestens für die Dauer des Bus Settle Delay (400ns) deaktiviert sein. Danach ist für nochmal 800ns ("Bus Free Delay") zu warten, bevor ein Device irgendein Signal setzen soll.

- Der Initiator setzt Busy und aktiviert sein eigenes ID-Bit auf dem Datenbus (zieht die entsprechende Datenbusleitung auf Low!).

- Nach Ablauf des "Arbitration Delay" (2,2µs) "untersucht" der Initiator den Datenbus auf höher priorisierte Initiatoren (DB0 ist niedrigste, DB7 höchste Prioritätsstufe). Wenn kein höher priorisiertes Gerät gefunden wurde, setzt der "Gewinner" die Select-Leitung auf Low.

- Niedriger priorisierte Geräte haben ihr Busy-Signal und ihr ID-Bit innerhalb eines Bus Settle Delay (400ns) "vom Bus zu nehmen", nachdem Select vom "Gewinner" aktiviert wurde.

Bevor der "Gewinner" nach Setzen von Select weitere Signale auf den Bus legen darf (Eintritt in die folgende Selektions-Phase), hat er mindestens 400ns (Bus Settle Delay) plus 800ns (Bus Clear Delay) abzuwarten.

SELECTION

Die dritte Hauptphase wird vom Initiator, der die Buskontrolle übernommen hat, verwendet, um das gewünschte Target zu adressieren.

Um diese Phase von der RESELECTION-Phase (Wiederaufnahme einer Verbindung zwischen einem Target und "seinem" Initiator) zu unterscheiden, muß _I/O aktiviert sein!

Der Initiator legt nun sein eigenes ID-Bit und das des Targets, mit dem zusammengearbeitet werden soll, auf den Datenbus. Nach 2 x 45ns (2 x Deskew Delay) aktiviert der Initiator das Busy-Signal. Weitere 400ns (Bus Settle Delay) später darf der Initiator dann mal schauen, ob denn ein Target reagiert hat.

Das adressierte Target hat innerhalb der sogenannten "Selection Abort Time" (=200ns) zu reagieren, indem es nun seinerseits das Busy-Signal auf Low zieht.

Innerhalb zwei Deskew Delays (2 x 45ns) nachdem der Initiator die Aktivierung von Busy durch das Target erkannt hat, wird vom Initiator die Select-Leitung deaktiviert.

Wenn innerhalb von ca. 250ms nach Ausgabe der Targetadresse vom Initiator keine Reaktion vom Target kommt, wird eine Timeout-Prozedur eingeleitet. Dabei kann der Initiator dann z. B. ein Reset durch Setzen der _RST-Leitung starten. Die andere Möglichkeit besteht darin, nach Deaktivieren von Select wieder in die BUS FREE-Phase einzutreten.

RESELECTION

Bei dieser Phase geht die Initiative zur Herstellung einer Verbindung nicht vom Initiator, sondern vom Target aus. Diese Phase ist nur in Systemen mit ARBITRATION möglich.

Dabei tauschen Target und Initiator eigentlich ihre Rollen. Das Target versucht während der ARBITRATION, den Bus zu übernehmen, und verfährt dabei genau wie dort bereits beschrieben. Wenn die Busübernahme geklappt hat, wird das Target die Select-Leitung aktivieren und, als Unterscheidung zur SELECTION, die RESELECTION durch Aktivierung von _I/O kenntlich machen. Außerdem gibt das Target dann das eigene ID-Bit und das des gewünschten Initiators auf den Bus.

Zwei Deskew Delays (2 x 45ns) später deaktiviert das Target Busy und prüft nach weiteren 400ns (Bus Settle delay) erstmalig den Bus auf eine Reaktion vom Initiator. Wenn der Initiator sich innerhalb von ca 250ms (=Selection Abort Time) meldet (=Busy aktiviert), übernimmt das Target selbst ebenfalls wieder mit Aktivierung von _BSY den Bus. Weitere 2 x 45ns (2

Deskew Delays) später wird vom Target Select deaktiviert. Danach steht wieder die Verbindung zwischen dem Initiator und dem Target, und das Target kann das Input/Output-Signal abschalten.

Auch hierbei gibt es bei Überschreiten der Selection Abort Time (250ms) (wie bei der SELECTION-Phase) die Möglichkeit eines Bus Reset (allerdings diesmal vom Target veranlaßt) oder des Übergangs in die BUS FREE-Phase.

Systeme ohne ARBITRATION

In Systemen ohne ARBITRATION/RESELECTION und mit nur einem Initiator ("Single Initiator Systems", wie im Moment auch beim TT) steigt der Initiator nach der BUS FREE-Phase direkt in die SELECTION-Phase ein!

Nach Feststellen von BUS FREE und Ablauf von 800ns (Bus Clear Delay) wird vom Initiator deshalb nur die gewünschte Target-SCSI-Adresse auf den Bus gesetzt und _SEL aktiviert. Wenn ein Target nur seine ID auf dem Bus findet und keine weitere (die des Initiators), geht es von einem "dummen" Initiator aus, der mit Meldungen nichts anfangen kann.

Wenn die Verbindung steht – Die Transferphase

Die vorher beschriebenen Phasen sind ja nur nötig, um die Verbindung zwischen zwei SCSI-Devices über den Bus herzustellen. Wenn das erfolgreich gelungen ist, können Befehle (COMMAND-Phase), Daten (DATA IN/OUT-Phase), Meldungen (MESSAGE-Phase) oder Zustandsinformationen (STATUS-Phase) ausgetauscht werden. Die komplette Steuerung des Austauschs liegt dabei jedoch beim Target! Dazu verwendet es die Signale _C/D, _I/O und _MSG. Folgende Phasen können durch diese drei Signale unterschieden werden:

MSG	C/D	I/O	Phase	Transferrichtung
F	F	F	DATA OUT	Initiator an Target
F	F	T	DATA IN	Target an Initiator
F	T	F	COMMAND	Initiator an Target
F	T	T	STATUS	Target an Initiator
T	F	F	Reserved	
T	F	T	Reserved	
T	T	F	MESSAGE OUT	Initiator an Target
T	T	T	MESSAGE IN	Target an Initiator

F = False (High!); T = True (Low!)

Der Initiator kann lediglich über die Aktivierung von _ATN darauf aufmerksam machen, daß er eine Meldung abzusetzen hat. Das Target muß darauf dann entsprechend reagieren und in die MESSAGE OUT-Phase wechseln!

Datenaustausch per Handshake

Die Datenbytes werden per _REQ/_ACK-Handshake im Asynchronbetrieb (Standardbetriebsweise) zwischen den beiden Geräten ausgetauscht. Jedes Byte erfordert einen _REQ/_ACK-Zyklus. Die Transferrichtung wird durch das _I/O-Signal festgelegt.

Bei einem Transfer vom *Target zum Initiator* (_I/O = Low!) legt das Target zunächst das zu übertragende Datenbyte auf den Bus (evtl. mit Paritätsbit auf der Paritäts-Leitung). Ein Deskew Delay (45ns) plus ein "Cable Skew Delay" (10ns) danach wird Request aktiviert. Der Initiator liest die Daten ein, sobald _REQ aktiviert wurde, und bestätigt die Übernahme der Daten durch Aktivierung von _ACK. Sobald das Target dieses bemerkt, kann es bereits wieder Änderungen auf dem Datenbus vornehmen (z. B. nächstes Byte anlegen) und _REQ wieder deaktivieren. Das sollte dann der Initiator auch schnellstens mitkriegen und als Reaktion darauf dann _ACK ebenfalls deaktivieren. Anschließend kann das Target wieder mit Aktivierung von _REQ das nächste Datenbyte ausgeben.

Wenn *Daten vom Initiator zum Target* übertragen werden sollen (_I/O = High!), gibt das Target den Wunsch nach einem neuen Datenbyte durch Aktivierung von _REQ bekannt. Der Initiator setzt die Datenbusleitungen auf den entsprechenden Wert und gibt nach einem Deskew Delay (45ns) plus dem Cable Skew Delay (10ns) die Gültigkeit der Daten auf dem Bus mit Aktivierung von _ACK bekannt. Sobald das Target das _ACK bemerkt, hat es die Daten einzulesen und als Quittung das Request-Signal zurückzunehmen. Wenn der Initiator dieses feststellt, hat er seinerseits _ACK zu deaktivieren und kann anschließend bereits neue Daten auf den Bus legen. Das Target holt sich diese dann mit dem nächsten _REQ wieder, wie bereits zuvor geschildert, ab.

COMMAND-Phase

Zunächst einmal sollte das Target natürlich erfahren, was der Initiator denn eigentlich will. Die Übermittlung der dazu erforderlichen Informationen geschieht in der COMMAND-Phase (dabei ist dann _C/D aktiviert!). Während dieser Phase sind _I/O und _MSG deaktiviert. Das Target holt sich per _REQ/_ACK-Handshake (wie schon zuvor beschrieben) den sogenannten "Command Deskriptor Block", bestehend aus sechs, zehn oder zwölf Bytes, beim Initiator ab.

An die COMMAND-Phase kann sich dann eine DATA IN- oder DATA OUT-Phase anschließen, während der Daten zwischen Initiator und Target ausgetauscht werden (z. B. bei SCSI-Platten am TT nahezu immer der Fall!).

DATA-Phasen

Ob es sich um eine DATA IN- oder DATA OUT-Phase handelt, wird durch den Zustand der _I/O-Leitung festgelegt (Low für DATA IN). Auch hier erfolgt der Datenaustausch wieder Byte für Byte (evtl. mit Paritätsbit) nach dem bereits beschriebenen _REQ/_ACK-Verfahren. Die _MSG- und _C/D-Leitung müssen dabei auf High gesetzt sein!

STATUS-Phase

Üblicherweise schließt die STATUS-Phase eine Initiator-Target-Kommunikation ab. Dabei wird an den Initiator ein Statusbyte übermittelt (also ist _I/O = Low!), das es dem Initiator erlaubt, eine Aussage über den Zustand der beendeten Operation zu treffen.

Auch diese Übermittlung erfolgt natürlich wieder gemäß dem _REQ/_ACK-Protokoll. Da es sich hierbei auch um eine Art Kontroll-Information handelt, ist während dieser Phase _C/D aktiviert. _MSG liegt auf High!

Die Bedeutung des Statusbytes ist nachfolgend dargestellt:

Status-Byte

Bit Byte	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	Reserv.	Herstellerspez.		Status Byte Code				Herst.spez.

Hier nun die möglichen Rückmeldungen und deren Bedeutung im Statusbyte:

Bits im Statusbyte

7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**Bedeutung**
R	H	H	0	0	0	0	H	GOOD
R	H	H	0	0	0	1	H	CHECK CONDITION
R	H	H	0	0	1	0	H	CONDITION MET/GOOD
R	H	H	0	0	1	1	H	Reserviert
R	H	H	0	1	0	0	H	BUSY
R	H	H	0	1	0	1	H	Reserviert
R	H	H	0	1	1	0	H	Reserviert
R	H	H	0	1	1	1	H	Reserviert

7	6	5	4	3	2	1	0	Bedeutung
R	H	H	1	0	0	0	H	INTERMEDIATE/GOOD
R	H	H	1	0	0	1	H	Reserviert
R	H	H	1	0	1	0	H	INTERMEDIATE/CONDITION MET/GOOD
R	H	H	1	0	1	1	H	Reserviert
R	H	H	1	1	0	0	H	RESERVATION CONFLICT
R	H	H	1	1	0	1	H	Reserviert
R	H	H	1	1	1	0	H	Reserviert
R	H	H	1	1	1	1	H	Reserviert

GOOD

Wird zurückgemeldet, wenn das Target ein Kommando erfolgreich abgeschlossen hat.

CHECK CONDITION

Es ist bei der Kommandoausführung irgendetwas nicht so gelaufen, wie vorgesehen. Der Initiator sollte sich ausführlichere Informationen mit einem REQUEST SENSE-Kommando vom Target übermitteln lassen.

CONDITION MET

Wird nur in Verbindung mit den Gruppe-1-Kommandos SEARCH .. auftreten. Wenn so eine Suchoperation erfolgreich beendet wurde, wird dies über diesen Status mitgeteilt. Eine Liste von verketteten Kommandos wird durch diese Statusmeldungen nicht unterbrochen. Die gesuchten Daten stehen dann in der Log. Blockadresse, die mit REQUEST SENSE erfragt werden kann.

BUSY

"Nicht jetzt, Liebling!" Das Target ist zur Zeit beschäftigt. Dieser Status sollte immer geliefert werden, wenn zur Zeit kein Kommando vom Initiator aufgenommen werden kann. Der Initiator sollte später wieder einen neuen Versuch starten.

INTERMEDIATE

In verketteten Kommandos (Linked Commands) ist bei erfolgreicher Beendigung eines jeden Teilkommandos (außer beim letzten Befehl!) dieser Status zurückzuliefern. Wenn das nicht geschieht, wird die Kette unterbrochen und folgende, verkettete Kommandos nicht mehr ausgeführt.

RESERVATION CONFLICT — Wenn der Initiator ein Gerät am Target ansprechen will, das für einen anderen Initiator reserviert wurde (auch das gibt's bei SCSI!), wird diese Statusinformation geliefert.

MESSAGE-Phasen

Während der MESSAGE-Phasen können zwischen Target und Initiator Meldungen ausgetauscht werden. Die Informationsrichtung wird dabei wieder durch den Zustand der _I/O-Leitung bestimmt (Low = MESSAGE IN).

Das Target kann jederzeit in die MESSAGE IN-Phase umschalten, indem es die _MSG-Leitung und die _I/O-Leitung aktiviert. Der Austausch der Message-Bytes geschieht wieder nach dem bereits beschriebenen _REQ/_ACK-Protokoll.

Der Initiator kann selbst nicht in die MESSAGE OUT-Phase gelangen. Alle Kontrolle während der Transferphasen liegt ja beim Target!

So muß der Initiator dann durch Aktivierung der Attention-Leitung (jederzeit möglich, außer während BUS FREE und ARBITRATION!) dem Target signalisieren, daß er Meldungen für das Target hat. Das Target wird dann zum nächstmöglichen Zeitpunkt in die MESSAGE OUT-Phase wechseln (_MSG und _C/D aktivieren, _I/O auf High). Datenaustausch wieder gemäß dem _REQ/_ACK-Verfahren. Es werden so lange Meldungen vom Initiator angefordert, bis dieser _ATN deaktiviert!

Von der MESSAGE-Phase kann das Target wieder zu irgendeiner der Transferphasen wechseln oder in die BUS FREE-Phase gehen. Alle SCSI-Devices sollen zumindest die Meldung "COMMAND COMPLETE" beherrschen. Diese wird vom Target zum Abschluß einer Operation an den Initiator gesendet (nach der STATUS-Phase). Der Atari-Festplattentreiber (in der Version 4.02) ignoriert diese Meldung einfach und verläßt sich nur auf das während der STATUS-Phase zurückgegebene Statusbyte.

Entsprechend der SCSI-Norm sollte der Initiator sich bereits während der SELECTION-Phase mit aktiviertem _ATN beim Target vorstellen, wenn er Wert auf einen ausführlichen Meldungsaustausch legt. Dadurch wird das Target veranlaßt, nach erfolgreicher SELECTION in die MESSAGE OUT-Phase zu wechseln, um dem Initiator die Möglichkeit zu geben, sich mit seinen Möglichkeiten (Identify-Message) vorzustellen.

Auf die Einzelheiten zu den Meldungen während der MESSAGE-Phasen soll hier aus Platzgründen jedoch nicht weiter eingegangen werden.

In folgender Tabelle finden Sie die wesentlichen Messages mit ihrer Kurzbezeichnung.

Message Codes

Code	Kurzbezeichnung	MSG-Phase	
00H	COMMAND COMPLETE	In	
01H	EXTENDED MESSAGE	In	Out
02H	SAVE DATA POINTER	In	
03H	RESTORE POINTERS	In	
04H	DISCONNECT	In	
05H	INITIATOR DETECTED ERROR		Out
06H	ABORT		Out
07H	MESSAGE REJECTIn		Out
08H	NO OPERATION		Out
09H	MESSAGE PARITY ERROR		Out
0AH	LINKED COMMAND COMPLETE	In	
0BH	LINKED COMMAND COMPLETE (WITH FLAG)	In	
0CH	BUS DEVICE RESET		Out
0DH .. 7FH	Reserved Codes		
80H .. FFH	IDENTIFY	In	Out

Wer sich darüber weiter informieren will (oder muß), sollte z. B. auf die SCSI-ANSI-Unterlagen X3T9.2, Rev. 17B, zurückgreifen, die alle nötigen Angaben in knapper Form auf ca. 200 Seiten enthalten. Außerdem lassen sich viele Festplattenhersteller in ihren Dokumentationen zu SCSI-Platten zum Teil recht ausführlich über den SCSI-Standard aus.

Die SCSI-Kommandos

Alle SCSI-Kommandos gehen davon aus, daß SCSI-Geräte Daten in Form von aufeinander folgenden log. Blöcken bereitstellen bzw. entgegennehmen können. Die Übersetzung dieser log. Blöcke in physikalische Strukturen ist Aufgabe des SCSI-Devices. Bei SCSI-Festplatten werden also solche logischen Blocknummern vom SCSI-Controller der Platte in entsprechende Zylinder-, Sektor- und Kopfnummern umgesetzt.

Mit jedem Kommando können ein oder mehrere log. Blocks bearbeitet werden. Mehrere Kommandos können hintereinander gekettet werden (Linked Commands), ohne vorher über BUS FREE, ARBITRATION und SELECTION gehen zu müssen.

Nach Beendigung eines Kommandos liefert das Target ein Statusbyte (während der STATUS-Phase!) an den Initiator. Da in einem Byte nicht besonders viele Statusinformationen untergebracht werden können, kann ein Target den sogenannten CHECK CONDITION-Status zu-

rückmelden. Der Initiator sollte dann mit einem speziellen Kommando (REQUEST SENSE) die zusätzlichen Statusinformationen von dem Target einlesen.

Der Command Descriptor Block

Der Initiator sendet an das Target einen Command Descriptor Block (CDB), der aus 6, 10 oder 12 Bytes bestehen kann. Zusätzlich zu den in diesem Block enthaltenen Daten können weitere Informationen während einer DATA OUT-Phase hinterhergesendet werden. So wird zum Beispiel in einem CDB einer SCSI-Platte mitgeteilt, daß ab log. Blocknummer X jetzt eine Anzahl von Y Datenblöcken geschrieben werden soll. Die Datenblöcke selbst werden dann in einer DATA OUT-Phase hinterhergesendet. Der prinzipielle Aufbau der drei möglichen Formen eines CDB sieht folgendermaßen aus:

Command Descriptor Block – 6 Bytes –

Bit / Byte	7	6	5	4	3	2	1	0
0	Gruppen-Code			Command Code				
1	Logical Unit			Log. Blockadresse (MSB)				
2	Log. Blockadresse							
3	Log. Blockadresse (LSB)							
4	Transferlänge							
5	Control Byte							

Command Descriptor Block – 10 Bytes –

Bit / Byte	7	6	5	4	3	2	1	0
0	Gruppen-Code			Command Code				
1	Logical Unit			Reserviert				RelAdr
2	Log. Blockadresse (MSB)							
3	Log. Blockadresse							
4	Log. Blockadresse							
5	Log. Blockadresse (LSB)							
6	Reserviert							
7	Transferlänge (MSB)							
8	Transferlänge (LSB)							
9	Control Byte							

Command Descriptor Block – 12 Bytes –

Bit Byte	7	6	5	4	3	2	1	0
0	Gruppen-Code			Command Code				
1	Logical Unit			Reserviert				RelAdr
2	Log. Blockadresse (MSB)							
3	Log. Blockadresse							
4	Log. Blockadresse							
5	Log. Blockadresse (LSB)							
6	Reserviert							
7	Reserviert							
8	Reserviert							
9	Transferlänge (MSB)							
10	Transferlänge (LSB)							
11	Control Byte							

Byte 0

Jedes Kommando beginnt mit einem Byte für den Operation Code (OP-Code).

Dabei wird durch die höchstwertigen drei Bits im OP-Code die Gruppenzugehörigkeit (Gruppen-Code) des Kommandos ausgedrückt.

Gruppe 0 – Kommandos mit 6 Bytes
Gruppe 1 – Kommandos mit 10 Bytes
Gruppe 2–4 – Reserviert
Gruppe 5 – Kommandos mit 12 Bytes
Gruppe 6–7 – Herstellerspezifische Kommandos

In jeder Gruppe sind dann noch 5 Bits für den Command-Code vorgesehen, durch die dann 32 verschiedene Befehle dargestellt werden können.

Logical Unit

Da ja ein Target durchaus mehr als ein Peripheriegerät steuern kann, müssen diese natürlich voneinander unterschieden werden.

Über die drei Bits für die LUN (=Logical Unit) lassen sich so bis zu acht Geräte pro SCSI-Device unterscheiden.

Log. Blockadresse

Hierüber wird die Nummer des anzusprechenden log. Blocks mitgeteilt. Log. Blocks beginnen mit der Nummer 0 und werden in ununterbrochener Reihe fortgezählt. Bei Gruppe 0-Kommandos stehen insgesamt "nur" 21 Bits für die LBA (Log. Blockadresse) zur Verfügung. Damit kann man immerhin schon über 2 Millionen log. Blockadressen ansprechen.

Bei den Gruppe 1- und 5-Kommandos sind dann über 4,2 Milliarden (!) Blöcke ansprechbar.

Da log. Blocks eine unterschiedliche Zahl von Datenbytes beinhalten können, sollte schon eine Abstimmung über die Blockgröße zwischen Initiator und Target vorgenommen sein. Über Befehle wie MODE SENSE oder READ CAPACITY kann der Initiator zum Beispiel die Blockgröße vom Target erfragen.

Relative Address Bit (RelAdr)

Dieses Bit wird nur bei Gruppe 1 und 5-Kommandos im Zusammenhang mit verketteten Befehlen (Linked Commands) verwendet. Wenn das Bit gesetzt ist, handelt es sich bei der Log. Blockadresse nicht um eine absolute, sondern um eine relative Blocknummer.

Diese relative Blocknummer (in 2er-Komplement-Darstellung; positiver oder negativer Wert!) wird zu der letzten verwendeten *absoluten* Log. Blockadresse addiert. Das Ergebnis ist dann die gewünschte LBA!

Transferlänge

Durch diesen Wert wird üblicherweise angegeben, wie viele log. Blocks übertragen werden sollen. Manche Kommandos enthalten allerdings in dem Feld Transferlänge die Anzahl der zu übertragenden *Bytes* (z. B. REQUEST SENSE).

Wenn für die Angabe der Transferlänge nur 1 Byte verwendet wird, können maximal 256 Blocks in einem Rutsch übertragen werden. Dabei steht ein Wert von Null für 256 Blocks! Alle anderen Werte zwischen 1..255 werden durch den entsprechenden Zahlenwert ausgedrückt.

In Befehlen, die zwei Bytes für die Transferlänge zur Verfügung stellen, können maximal 65536 Blöcke pro Kommando übertragen werden. Auch hier wird die höchste Blockanzahl wieder durch den Wert von Null im Transferlängenfeld dargestellt.

Control Byte

Das jeweils letzte Byte in einem CDB ist das Control Byte. Es hat den folgenden Aufbau:

Control Byte

Bit Byte	7	6	5	4	3	2	1	0
LAST	Herstellerspez.		Reserviert				Flag	Link

Link Bei einem gesetzten Bit wird dem Target signalisiert, daß es sich um eine Verkettung von Befehlen handelt. Es werden also weitere Befehle vom Initiator ans Target geliefert.

Targets, die diese "Linked Commands" unterstützen, liefern bei erfolgreicher Kommandoausführung in der STATUS-Phase ein INTERMEDIATE-Statusbyte. Außerdem wird eine entsprechende Meldung ("LINKED COMMAND COMPLETE" bzw. "LINKED COMMAND COMPLETE (WITH FLAG)" in einer MESSAGE IN-Phase an den Initiator geliefert.

Bei Targets, die nicht mit "Linked Commands" klarkommen, wird am Ende der Operation ein CHECK CONDITION-Statusbyte geliefert. Wenn man (der Initiator) dann mit einem REQUEST SENSE-Befehl beim Target nachfragt, was denn war, erhält man in diesem Falle ein "ILLEGAL REQUEST" signalisiert.

Flag Dieses Bit wird nur in Verbindung mit dem Link-Bit benutzt und legt fest, welche Meldung (LINKED COMMAND COMPLETE bei Flag = 0, LINKED COMMAND COMPLETE (WITH FLAG) bei Flag = 1) an den Initiator gesendet wird. Typische Anwendung für dieses Bit ist die Verursachung eines Interrupts zwischen zwei Linked Commands im Initiator.

Gruppe 0-Kommandos

Im folgenden sollen die wichtigsten Gruppe 0-Kommandos für "Direct-Access Devices" laut SCSI erläutert werden. Diese Kommandos werden deshalb auch von nahezu allen SCSI-Festplatten "verstanden" und sind deshalb auch in mancher Harddisk-Software (Treiber und Tools wie AHDI 4.02, SCSITOOL 2.09 und HUSHI 2.12) zu finden.

TEST UNIT READY

Mit diesem Kommando wird getestet, ob die angesprochene LUN (Logical UNit) des angesprochenen Targets bereit ist. Damit wird nicht etwa ein Selbsttest angestoßen. Wenn die angesprochene LUN bereit für den Zugriff auf das Medium ist, liefert der TEST UNIT READY-Befehl einen GOOD-Status zurück.

TEST UNIT READY (OP-Code 00H)

Bit / Byte	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0
1	Logical Unit			Reserviert				
2	Reserviert							
3	Reserviert							
4	Reserviert							
5	Herstellerspez.			Reserviert			Flag	Link

REZERO UNIT

Das Target soll die angesprochene LUN in einen speziellen Anfangszustand versetzen (lt. SCSI). Bei Festplatten wird man mit diesem Kommando die Platte auf Zylinder 0 fahren (nicht zu verwechseln mit dem "Parken" von Festplatten! Das wird mit dem STOP-Befehl realisiert).

REZERO UNIT (OP-Code 01H)

Bit / Byte	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1
1	Logical Unit			Reserviert				
2	Reserviert							
3	Reserviert							
4	Reserviert							
5	Herstellerspez.			Reserviert			Flag	Link

REQUEST SENSE

Mit diesem Befehl lassen sich in der an die COMMAND-Phase anschließenden DATA IN-Phase Sense-Daten vom Target einlesen. Diese Sense-Daten (Sense = Bedeutung, Befinden) liefern weitere Informationen über die Ursache einer Störung, die mit einem CHECK CONDITION-Status nach Abschluß eines anderen Kommandos an den Initiator signalisiert wurde. Sense-Daten werden vom Target nach einer Störung so lange bereitgehalten, bis sie durch diesen Befehl "abgeholt" werden oder ein anderer Befehl (vom gleichen Initiator, dem CHECK CONDITION gemeldet wurde) ausgeführt wird.

REQUEST SENSE (OP-Code 03H)

Bit Byte	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	1
1	Logical Unit			Reserviert				
2	Reserviert							
3	Reserviert							
4	Transferlänge							
5	Herstellerspez.			Reserviert			Flag	Link

Im Feld *Transferlänge* übermittelt der Initiator dem Target, wie viele *Bytes* Sense-Daten er vom Target maximal aufnehmen kann! Das Target beendet dann die folgende DATA IN-Phase, wenn diese Zahl von Bytes an den Initiator übermittelt ist (auch wenn noch mehr Bytes zur Verfügung stünden!) oder wenn alle Sense-Daten übermittelt sind. Bei einer Transferlänge von Null sendet das Target übrigens trotzdem (gem. SCSI-Vereinbarung) vier Sense-Datenbytes!

Man unterscheidet zwei Formate bei den übermittelten Sense-Daten, wobei das Format durch die Art des Fehlers bestimmt wird. Genauer gesagt, in welche Klasse der Fehler eingeordnet wird:

Nonextended Sense — Für die Fehlerklassen 0..6 (Error class 0..6) kommt ein 4-Byte-Format zur Anwendung.

NONEXTENDED SENSE DATA FORMAT

Bit Byte	7	6	5	4	3	2	1	0
0	AdValid	Fehlerklasse			Fehlercode			
1	Herstellerspezifisch			Logische Blockadresse (MSB)				
2	Logische Blockadresse							
3	Logische Blockadresse (LSB)							

AdValid — Wenn dieses Bit gesetzt ist, steht die gelieferte Logische Blockadresse in direkter Beziehung zu dem Fehler (wenn bei einem Festplattenzugriff z. B. ein Lesefehler auftrat, wird man per REQUEST SENSE dann in der log. Blockadresse die Nummer des Blocks übermittelt bekommen, der nicht gelesen werden konnte!)

Fehlerklasse — Herstellerspezifische Nummer, mit der der Fehler klassifiziert wird (Fehlerwichtigkeit).

Fehlercode — Fehlernummer in einer bestimmten Fehlerklasse. Herstellerspezifisch. Einige Plattenhersteller liefern im Fehlercode den gleichen Wert wie im Sense Key-Feld beim Extended Sense-Format!

Häufiger anzutreffen ist das erweiterte Format für Sense-Daten. Vielfach liefern Targets nur bei einer Transferlänge von Null im REQUEST SENSE-Befehl das "kurze" Sense-Data-Format.

Extended Sense — Dieses Format umfaßt mindestens acht Sense-Datenbytes! Es wird nur bei der Fehlerklasse 7 verwendet

Folgendes Format wird benutzt:

EXTENDED SENSE DATA FORMAT

Bit Byte	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	Valid	Fehlerklasse (= 7)			Fehlercode (= 0)			
1	Segmentnummer							
2	FM	EOM	ILI	Reserv.	Sense Key			
3–6	(MSB)	Informationsbytes (LSB)						
7	Anzahl zusätzlicher Sense Bytes (n)							
8.. n+7	Zusätzliche Sense Bytes							

Die Fehlerklasse 7 steht für Extended Sense! Steht dann im Fehlercode eine 0, wird die oben gezeigte Datenstruktur verwendet. Die Fehlercodes 1..E_H sind reserviert. Bei einem Fehlercode von F_H kommt ein herstellerspezifisches Datenformat zur Anwendung (bei SCSI-Festplatten aber nicht anzutreffen!).

Valid — Wenn das Bit gesetzt ist, sind die Informationsbytes in den Sense-Bytes 3..6 auch gültig. Üblicherweise findet man hier dann die log. Blockadresse, die mit der Fehlermeldung im Zusammenhang steht.

Segmentnummer — Tritt nur bei Fehlern in Zusammenhang mit speziellen Befehlen wie COPY, COMPARE und COPY WITH VERIFY auf. Bei diesen Befehlen können zu kopierende/vergleichende Datenbereiche in soge-

nannten Segmentdeskriptoren ("Bereichs-Beschreibern") festgelegt werden. Die Segmentnummer liefert in diesem Fall die Nummer des Segmentdeskriptors, der mit dem Fehler in Zusammenhang steht.

In den Informationsbytes ist dann ein Zahlenwert in Zweierkomplementdarstellung enthalten, der darauf schließen läßt, an welcher Stelle (Block/Byte) in dem Segment der Fehler aufgetreten ist.

FM — File Mark-Bit. Nur bei sequentiell arbeitenden Peripheriegeräten eines Targets (z. B. Bandlaufwerke) verwendet. Filemarken werden zum Auffinden bestimmter, zusammengehöriger Datenstrukturen (ähnlich den Adreßfeldmarken in einer Spur bei Floppy-Disks) benutzt. Wenn das Bit gesetzt wird, ist eine Filemarke erkannt worden.

EOM — End-of-Medium-Bit. Wird nur bei sequentiellen Peripheriegeräten und Printer-Devices gebraucht. Bei gesetztem Bit wird auf einen End-of-Medium-Zustand aufmerksam gemacht (Band befindet sich am Ende oder beim Rückspulen ganz am Anfang! Papier ist zu Ende o. ä).

ILI — Incorrect length indicator. Wird gesetzt, wenn die im Kommando geforderte log. Blockgröße nicht mit der auf dem Medium "harmoniert"!

Sense Key — "Bedeutungs-Schlüssel". Hiermit ist ebenfalls so eine Art Fehlergruppierung möglich. Beispielsweise ließen sich damit, je nach Sense Key, unterschiedliche Fehlerbehandlungen durch den Initiator realisieren.

Nachfolgend eine Tabelle mit den Sense Keys nach SCSI-Standard:

Sense Key	**Bedeutung**
0_H	NO SENSE. Keine besonderen Vorkommnisse (die einer Sense-Information bedürfen). Diesen Key erhält man, wenn REQUEST SENSE nach einem korrekt ausgeführten Befehl ausgeführt wird (Statusbyte war GOOD!). Aber auch bei einem mit CHECK CONDITION abgeschlossenen Kommando gibt es diesen Key, wenn z. B. nur eine Filemark oder End-of-Medium erkannt wurde.

Sense Key	Bedeutung
1_H	RECOVERED ERROR. Da war wohl ein Fehler, aber er konnte durchs Target noch während der Ausführung korrigiert werden.
2_H	NOT READY. Gewünschte LUN ist nicht bereit.
3_H	MEDIUM ERROR. Fehler auf dem Medium (der nicht korrigiert werden konnte!).
4_H	HARDWARE ERROR. Nichtbehebbarer Fehler in der Hardware.
5_H	ILLEGAL REQUEST. Im Command Descriptor Block oder den zusätzlich zum Kommando gehörenden Daten waren ungültige Parameter.
6_H	UNIT ATTENTION. Target wurde zurückgesetzt oder das Medium gewechselt.
7_H	DATA PROTECT. Der per Kommando angesprochene Block ist von der gewünschten Operation ausgenommen. (Schreib/Leseschutz).
8_H	BLANK CHECK. Bei einem sequentiell arbeitenden Device oder einem "Write-once read-multiple"-Device (=WORM-Device; nur einmal beschreibbares, aber beliebig oft lesbares Medium wie z. B. Optical Disk) wurde ein "blanker" Block (nicht initialisierter Block) beim Lesezugriff erkannt. Oder bei einem WORM-Device mit sequentiellem Zugriff wurde ein bereits mit Daten gefüllter Block beim Schreibzugriff erkannt.
9_H	Herstellerspezifischer Sense Key.
A_H	COPY ABORTED. COPY-, COMPARE- oder COPY AND VERIFY-Kommando wurde abgebrochen.
B_H	ABORTED COMMAND. Der letzte Befehl wurde vom Target abgebrochen. Ein weiterer Versuch vom Initiator könnte nichts schaden.
C_H	EQUAL. Bei einem SEARCH DATA-Kommando wurde die Gleichheitsbedingung erfüllt.
D_H	VOLUME OVERFLOW. Ein mit Pufferspeicher arbeitendes Gerät hat bei einem Schreibzugriff das Ende des Mediums erreicht. Dabei konnten nicht mehr alle Daten im Puffer auf dem Medium untergebracht werden.

Sense Key	Bedeutung
E_H	MISCOMPARE. Bei einem Vergleich wurde festgestellt, daß die Quelldaten vom Initiator nicht mit vom Medium gelesenen Daten übereinstimmen.
F_H	Reserviert!

Die meisten Festplattenhersteller liefern zusätzlich zu diesen Extended Sense-Bytes noch weitere spezielle Informationen. Die Verlängerung der Extended Sense-Liste ist durch das Feld *Additional Length* um die dort eingetragene Byteanzahl möglich.

Nahezu alle SCSI-Festplatten liefern in den zusätzlichen Sense-Bytes an Byte-Position 12 einen zusätzlichen Sense-Code. Außerdem wird dieser Sense-Code bei manchen Festplatten auch im ersten Byte des Nonextended Sense-Format geliefert.

Der zusätzliche Sense Code gibt dann noch genaueren Aufschluß über Fehlerzustände der Platte oder des Controllers. Man wird aber dann nicht umhin kommen, sich mit der Hersteller-Dokumentation zu "bewaffnen", um den Fehlerursachen genau auf den Grund zu gehen. Zum Glück hatten viele Plattenhersteller "die gleichen Ideen", was diesen Sense-Code angeht. Man entdeckt so viele "Gemeinsamkeiten" beim Vergleich der Sense-Codes unterschiedlicher Plattenhersteller.

FORMAT UNIT

Mit diesem Befehl wird das Peripheriegerät angewiesen, das Medium zu formatieren.

Damit ist nach Abschluß des Kommandos gewährleistet, daß auf alle verfügbaren Datenblöcke zugegriffen werden kann.

FORMAT UNIT (OP-Code 04H)

Bit Byte	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	0
1	Logical Unit			FmtDa	CmpLst	Defektlistenformat		
2	Herstellerspezifisch							
3	Interleave (MSB)							
4	Interleave (MSB)							
5	Herstellerspez.		Reserviert				Flag	Link

Im Byte 2 des Kommandos erlauben manche Plattenhersteller die Angabe des Bitmusters, welches beim Formatiervorgang in die Sektoren geschrieben wird. Ähnlich dem Virgin-Byte bei der Floppy-Disk-Formatierroutine des XBios (Flopfmt) im XBios des ST(E)/TT.

Bei *Interleave* kann eine spezielle Zuordnung zwischen der Reihenfolge der log. Blöcke und den dazugehörigen physikalischen Blocks angegeben werden.

FmtData — Nach dem FORMAT UNIT-Kommando folgt eine DATA OUT-Phase, wenn dieses Bit gesetzt ist. Damit wird üblicherweise eine Liste mit defekten Stellen an das Target übermittelt, die beim Formatiervorgang übersprungen werden sollen.

CmpLst — Wenn das Bit gesetzt ist, muß das Target davon ausgehen, daß in der während der DATA OUT-Phase gelieferten Liste alle bekannten Defektstellen enthalten sind. Eine im Target vorhandene, ältere Defektliste ist dann nicht mehr zu verwenden! Wenn das Bit gelöscht ist, soll zu der bereits im Target existierenden Liste mit Defektstellen die vom Initiator gesendete Defektliste zusätzlich verwendet werden.

Alle beim Formatiervorgang gefundenen neuen Defektstellen werden zu den bereits bekannten Defekten (ob im Target vorhanden oder vom Initiator während einer DATA OUT-Phase geliefert) "dazugepackt"!

Defektlistenformat — Diese drei Bits bestimmen, zusammen mit den FmtData- und CmpLst-Bits, den Aufbau der resultierenden Defektliste und deren Verwendung für den Formatiervorgang. Nicht alle SCSI-Platten beherrschen alle nachfolgenden Kombinationen!

4	3	2	1	0	
Fmt	**Data**				
	Cmp	**Lst**			
		Defektlistformat			
					Bemerkungen
0	X	X	X	X	Formatieren ohne Defektliste vom Initiator! Es wird nur die im Target vorhandene Defektliste (bestehend aus vom Hersteller eingetragenen Defekten und bisher erkannten Defektstellen) verwendet. Diese Option verstehen wohl alle Platten!

4 Fmt	3 Data Cmp	2 Lst	1 Defektlistformat	0	Bemerkungen
1	0	0	X	X	Zusätzlich zu den bekannten Defektstellen im Target (Known defect list) sollen die vom Initiator gelieferten Defektstellen berücksichtigt werden. Für die Defektliste vom Initiator wird das log. Blockformat (Format A) benutzt.
1	1	0	X	X	Die vom Initiator gelieferte Defektliste soll als vollständige Liste aller Defekte benutzt werden (also keine Liste aus dem Target benutzen!). Die gelieferte Defektliste enthält die Defektstellen im log. Blockformat (Format A).
1	0	1	0	0	Zusätzlich zur Known defect list im Target sind die vom Initiator gelieferten Defektstellen zu berücksichtigen. Die Liste liegt als Liste der phys. Sektoren vor (Format B).
1	1	1	0	0	Die vom Initiator gelieferte Defektliste soll als vollständige Liste aller Defekte benutzt werden (also keine Liste aus dem Target benutzen!). Die gelieferte Defektliste enthält die Defektstellen als Liste von phys. Sektoren (Format B).
1	0	1	0	1	Zusätzlich zur Known defect list im Target sind die vom Initiator gelieferten Defektstellen zu berücksichtigen. Die Liste liegt in Form von Byte-Abständen vom Indeximpuls vor (Format C).
1	1	1	0	1	Die vom Initiator gelieferte Defektliste soll als vollständige Liste aller Defekte benutzt werden (also keine Liste aus dem Target benutzen!). Die Liste liegt in Form von Byte-Abständen vom Indeximpuls vor (Format C).

Andere Kombinationen sind herstellerspezifisch bzw. reserviert! Nachfolgend die drei möglichen Formate für die Defektliste, welche während einer DATA OUT-Phase vom Initia-

tor an das Target übermittelt werden kann. Dabei weist jede Liste zu Beginn den sogenannten Defect List Header auf, der letztlich enthält, wie viele Bytes an eigentlicher Defektinformation noch folgen.

Defektliste im log. Blockformat (Format A)

Byte	Defect List Header
0	Reserviert
1	Reserviert
2	Defektlistenlänge (MSB)
3	Defektlistenlänge (LSB)
	Defect Descriptor(s)
0	Def. Blockadr. (MSB)
1	Def. Blockadr.
2	Def. Blockadr.
3	Def. Blockadr. (LSB)

Jeder Defect Descriptor enthält eine Log. Blockadresse und belegt 4 Bytes in der Descriptorliste. Die Defect Deskriptoren sind in aufsteigender Form zu liefern.

Defektliste im phys. Sektorformat (Format B)

Byte	Defect List Header
0	Reserviert
1	Reserviert
2	Defektlistenlänge (MSB)
3	Defektlistenlänge (LSB)
	Defect Descriptor(s)
0	Zylindernummer (MSB)
1	Zylindernummer
2	Zylindernummer (LSB)
3	Kopfnummer
4	Sektornummer (MSB)
5	Sektornummer
6	Sektornummer
7	Sektornummer (LSB)

Jeder Defect Descriptor enthält die Position des Defekts durch Angabe des Zylinders, der Kopfnummer und der Sektornummer.

Jeder Defect Descriptor belegt 8 Bytes in der Liste.

Auch hier sind die Defekte in aufsteigender Folge anzugeben. Dabei wird die Zylindernummer am stärksten gewichtet, dann die Kopfnummer, und als niedrigste Gewichtung wird die Sektornummmer verwendet. Wird bei Sektornummer $FFFFFFFF angegeben, so ist der ganze Track als defekt anzusehen.

Defektliste im "Byte-from-Index"-Format (Format C)

Byte	Defect List Header
0	Reserviert
1	Reserviert
2	Defektlistenlänge (MSB)
3	Defektlistenlänge (LSB)
	Defect Descriptor(s)
0	Zylindernummer (MSB)
1	Zylindernummer
2	Zylindernummer (LSB)
3	Kopfnummer
4	Bytes-from-Index (MSB)
5	Bytes-from-Index
6	Bytes-from-Index
7	Bytes-from-Index (LSB)

Jeder Defect Descriptor enthält die Position des Defekts durch Angabe des Zylinders, der Kopfnummer und der Entfernung des Defekts in Bytes vom Indeximpuls. Dabei umfaßt eine Defektstelle lt. SCSI immer einen Bereich von 8 Bytes!

Jeder Defect Descriptor belegt 8 Bytes in der Liste.

Auch hier sind die Defekte in aufsteigender Folge anzugeben. Dabei wird die Zylindernummer am stärksten gewichtet, dann die Kopfnummer, und als niedrigste Gewichtung wird die Entfernungsangabe in Bytes vom Indeximpuls gewertet.

Bei $FFFFFFFF in "Bytes-from-Index" ist der gesamte Track als defekt zu betrachten.

Die meisten Plattenhersteller benutzen noch einige Bits in Byte 1 des Defect List Headers, um bestimmte "Anweisungen" an die Platte zu übermitteln.

Defect List Header

Bit / Byte	7	6	5	4	3	2	1	0
0	Reserviert							
1	FOV	DPRY	DCRT	STPF				
2	Defektlistenlänge (MSB)							
3	Defektlistenlänge (LSB)							

...

FOV — Format Options Valid. Wenn dieses Bit gesetzt ist, werden die Bits 4..6 in diesem Byte in ihrer Einstellung beim Formatiervorgang berücksichtigt. Bei gelöschtem FOV-Bit wird mit Standardvorgaben für die Bits 4..6 dieses Bytes gearbeitet.

DPRY — Disable Primary Defect List. *Bei gesetztem Bit* wird die bei der Fabrikation der Platte aufgebrachte Defektliste beim Formatiervorgang *nicht* berücksichtigt!

DCRT — Disable Media Certification. *Bei gesetztem Bit werden die* während eines Formatiervorgangs gefundenen *neuen Defektstellen nicht mit in die Defektliste aufgenommen*!

STPF — Stop Format on Error. Bei gesetztem Bit wird ein Formatiervorgang abgebrochen (mit Status = CHECK CONDITION), wenn die Defektliste nicht vom Medium gelesen werden konnte. Also dann, wenn die Defektliste der Platte selbst einen Defekt aufweist. Bei gelöschtem Bit wird normal weiterformatiert.

REASSIGN BLOCKS

Mit diesem Befehl lassen sich defekte log. Blocks ausklammern und statt dessen (für solche Zwecke vorgesehene) Reserveblocks verwenden. Der Peripherie-Controller wird dann bei einem Zugriff auf einen ursprünglich defekten Block diesen Zugriff auf den Reserveblock umlenken. Auch bereits schon mal neu zugeordnete Blöcke (Reassigned Blocks), die sich im Laufe der Zeit als defekt herausstellen, können wiederum durch andere Reserveblocks ersetzt werden, bis halt keine Reserven mehr da sind!

Die bei einem Reassign herbeigeführte Neu-Zuordnung eines log. Blocks ist in der Regel mit der Änderung des Blockinhalts verbunden!

Viele SCSI-Platten führen eine solche Reassign-Operation selbst durch, wenn bei einem Zugriff festgestellt wurde, daß der gewünschte Block defekt ist.

REASSIGN BLOCKS (OP-Code 07H)

Bit / Byte	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	1	1
1	Logical Unit			Reserviert				
2	Reserviert							
3	Reserviert							
4	Reserviert							
5	Herstellerspez.		Reserviert				Flag	Link

Auch bei diesem Kommando folgt eine DATA OUT-Phase, in der die Liste mit den defekten Blöcken (Defektliste) an das Target zum Reassign übergeben wird. Der Aufbau dieser Liste ist der gleiche wie bei der Defektliste im log. Blockformat (Format A) beim FORMAT UNIT-Befehl!

READ

Mit diesem Befehl wird das Target veranlaßt, Daten von der gewünschten LUN an den Initiator zu senden. An das Kommando schließt sich also eine DATA IN-Phase an. Es lassen sich dabei bis zu 256 Blöcke vom Target einlesen. Die Blockanzahl ist im Feld *Transferlänge* einzutragen (0 = 256 Blocks!). Durch die 21 Bits für die Blockadresse kann mit diesem 6-Byte-Kommando immerhin schon auf über 2 Millionen Blocks zugegriffen werden. Bei einer log. Blockgröße von üblicherweise 512 Bytes erlaubt dieser Befehl also Zugriffe auf immerhin 1 GByte Daten!

READ (OP-Code 08H)

Bit / Byte	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	0
1	Logical Unit			Log. Blocknummer (MSB)				
2	Log. Blocknummer							
3	Log. Blocknummer (LSB)							
4	Transferlänge							
5	Herstellerspez.		Reserviert				Flag	Link

WRITE

Dieses Kommando ist das Gegenstück zum READ-Befehl und zieht eine DATA OUT-Phase nach sich, während der die zu schreibenden Daten an das Target übergeben werden.

Für Transferlänge und log. Blockadresse gilt sinngemäß das gleiche, wie bereits beim READ-Befehl erwähnt.

WRITE (OP-Code 0AH)

Bit / Byte	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	1	0
1	Logical Unit			Log. Blocknummer (MSB)				
2	Log. Blocknummer							
3	Log. Blocknummer (LSB)							
4	Transferlänge							
5	Herstellerspez.		Reserviert				Flag	Link

SEEK

Die angesprochene Logical Unit (LUN) des Targets wird durch den SEEK-Befehl veranlaßt, den im Kommmando übergebenen Block anzusteuern.

Bei Platten wird dieses Kommando wohl kaum gebraucht, weil Zugriffe auf log. Blockadressen immer direkt mit READ- oder WRITE-Kommandos durchgeführt werden.

SEEK (OP-Code 0BH)

Bit / Byte	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	1	1
1	Logical Unit			Log. Blocknummer (MSB)				
2	Log. Blocknummer							
3	Log. Blocknummer (LSB)							
4	Reserviert							
5	Herstellerspez.		Reserviert				Flag	Link

INQUIRY

Mit diesem "Erkundigungs"-Kommando kann man mehr über die am Target angeschlossenen LUNs erfahren.

In einer DATA IN-Phase liefert das Target dann Informationen über die angesprochene Logical Unit.

Zum Teil sind in den Daten "Klartextinformationen" über den Typ der LUN enthalten.

INQUIRY (OP-Code 12H)

Bit Byte	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	0	0	0	1	0	0	1	0
1	Logical Unit			Reserviert				
2	Reserviert							
3	Reserviert							
4	Transferlänge							
5	Herstellerspez.			Reserviert			Flag	Link

Im Feld *Transferlänge* wird angegeben, wie viele *Bytes* der Initiator max. während der DATA IN-Phase vom Target aufnehmen kann.

Die während der DATA IN-Phase gelieferten INQUIRY-Daten weisen dabei folgende Struktur auf:

INQUIRY DATA

Bit Byte	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	Peripheriegeräte-Typ							
1	RMB	Device-Type Qualifier						
2	ISO-Version		ECMA-Version			ANSI-Version		
3	Reserviert							
4	Zusätzliche Bytes (N)							
5..N+4	Herstellerspezifische Daten							

Peripheriegeräte-Typ	00_H Direct-access Device (z. B. Platte) 01_H Sequ.-access Device (z. B. Bandlaufwerk) 02_H Drucker 03_H Prozessor-Device 04_H Write once, read-mult. (z. B. opt. Disk) 05_H Read-only, dir.-Access (z. B. CD-ROM) Die weiteren Codekombinationen sind reserviert!
RMB	Removable Medium Bit. Wenn das Medium wechselbar ist, wird dieses Bit gesetzt, sonst nicht.
Device-Type Qualifier	Diese sieben Bits stehen dem Anwender für spezielle Identifikationen der im System verwendeten LUNs zur Verfügung. Damit kann der Anwender zum Beispiel durch Vergabe von solchen Device-Type Qualifiern (meistens mit dem Befehl MODE SELECT einstellbar) mehrere gleichartige Peripheriegeräte mit Wechselmedien auseinanderhalten (Wechselplatte, Streamer, Bandlaufwerk usw.).
Version-Byte	In diesem Byte lassen sich Informationen unterbringen, welcher Normenversion (ISO, ECMA und ANSI) die SCSI-Ausstattung (Möglichkeiten) das verwendete Gerät gerecht wird.

Im Byte 4 steht dann eine Information, wie viele herstellerspezifische Bytes noch folgen werden. Wenn noch Herstellerbytes folgen, findet man in den Bytepositionen 8..15 der INQUIRY-Daten üblicherweise den Herstellernamen (im erweiterten ASCII (8 Bit)) wieder. Bytes 16..31 dienen meistens zur Produktbezeichnung und evtl. der Angabe von Seriennummern (ebenfalls in ASCII). Weitere Bytes liefern zum Teil ebenfalls Seriennummern oder Versionsnummern der Hard- und Firmware sowie das Fabrikationsdatum.

MODE SELECT

Mit diesem Kommando lassen sich in einer anschließenden DATA OUT-Phase eine Menge Parameter an das Target übermitteln. Wie viele Parameterbytes während der DATA OUT-Phase übermittelt werden, ist in dem Feld *Transferlänge* anzugeben.

In der Parameterliste sind solche Angaben enthalten wie max. Blockzahl und log. Blockgröße in Bytes! Außerdem lassen sich verschiedene Betriebsmodi einstellen. Bei Festplatten mit Cache kann z. B. die Anzahl der Cachesegmente und deren Größe, bei Platten mit Shutdown-Funktion die Zeit, nach der diese bei "Nichtgebrauch" in einen stromsparenden Standby-Modus gehen, usw. eingestellt werden.

Üblicherweise gelten die mit MODE SELECT gemachten Einstellungen nur bis zum nächsten Reset/Ausschalten. Sollen diese Einstellungen allerdings ab jetzt immer gelten (bis zu irgendeiner erneuten Änderung), so müssen diese gespeichert werden. Die Speicherung wird durch Setzen des *SP-Bits* im MODE SELECT-Befehl veranlaßt.

MODE SELECT (OP-Code 15H)

Bit Byte	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	0	0	0	1	0	1	0	1
1	Logical Unit			Reserviert				SP
2	Reserviert							
3	Reserviert							
4	Transferlänge							
5	Herstellerspez.			Reserviert			Flag	Link

Die MODE SELECT-Parameterliste weist nachfolgende Struktur auf, wobei es möglich ist, diese Liste um herstellerspezifische Angaben zu erweitern.

Dabei werden diese zusätzlichen Angaben in Funktionsgruppen zusammengefaßt. So eine Funktionsgruppe wird als Page bezeichnet.

Jede Page wird durch den Page-Code identifiziert, der jeweils im ersten Byte der Page auftaucht. Damit eine Abgrenzung zu nachfolgenden Pages möglich ist, wird im zweiten Byte jeder Page ein entsprechender Längenwert angegeben.

Dieser Wert enthält die Anzahl der Pagebytes, die noch nach dem Längenbyte kommen!

Byte	**MODE SELECT Header**
0	Reserviert
1	Medium-Typ (0)
2	Reserviert
3	Länge des Blockdescriptors
	Blockdescriptor(s)
0	Density Code
1	Blockanzahl (MSB)
2	Blockanzahl

Byte	MODE SELECT Header
3	Blockanzahl (LSB)
4	Reserviert
5	Blockgröße (MSB)
6	Blockgröße
7	Blockgröße (LSB)

Bei *Medium-Typ* sieht SCSI eine ganze Reihe von möglichen Codes vor, die sich aber hauptsächlich auf Flexible Disks und Magnetbänder beziehen. Bei Fest- und Wechselplatten steht dort 00_H!

Der *Density-Code* im Blockdescriptor ist auch wiederum nur für Flexible Disks gedacht und bei Fest-/Wechselplatten = 00_H.

Blockanzahl spezifiziert, wie viele Blocks auf dem Medium die vorgenannte Density und die nachfolgende Blockgröße aufweisen. Es ist gemäß SCSI zwar möglich, auf einer LUN mehrere Bereiche zu vereinbaren, die unterschiedliche Blockgrößen und Densities aufweisen. Dann sind natürlich entsprechend viele Blockdeskriptoren erforderlich. Bei Platten wird man solche Einstellungen aber nicht finden!

Page Descriptoren

Die bereits erwähnten Pages mit herstellerspezifischen Parametern sind optional und schließen bei Bedarf unmittelbar an die Blockdeskriptoren an. Die Reihenfolge, in denen die Pages gesendet werden, ist beliebig. Es ist auch nicht erforderlich, daß jeweils immer alle Pages mit einem MODE SELECT-Kommando übermittelt werden. So können spezielle Einstellungen jederzeit durch Übermittlung nur dieser speziellen Page bei Bedarf vorgenommen werden.

Page Code	Bedeutung
00_H	Allgemeine Betriebsparameter
01_H	Parameter für Fehlerbehebung
02_H	Parameter für Disconnect/Reconnect-Verhalten
03_H	Format-Parameter
04_H	Unveränderbare Plattenparameter (Disk Geometry) *
$05_H..1F_H$	Reserviert
20_H	Seriennummer der Platte (manchmal!) *
$21_H..3F_H$	Reserviert bzw. (sehr) herstellerspezifisch

* = *Diese Pages können nur mit MODE SENSE ausgelesen werden!*

Es handelt sich hierbei um keinerlei bindende Zuordnung gem. SCSI. Zum Teil findet man jedoch bei vielen Plattenherstellern übereinstimmende Bedeutungen bei den Parametern.

Allgemeine Betriebsparameter (Page Code 00_H)

Bit / Byte	7	6	5	4	3	2	1	0
0	Reserv.	Page Code = 00_H						
1	Page-Länge (üblicherweise = 02_H)							
2	USG	RECY	STAT	UNIT ATN	Reserviert			
3	Device-Type Qualifier							

USG — Usage Counter. SCSI-Platten führen genau Buch über Schreib-/Lesezugriffe und Seek-Operationen. Die dabei entstehenden Zählerstände können bei Überschreiten von einstellbaren Grenzwerten zu einer Fehlermeldung führen. Wenn dieses Bit nicht gesetzt ist, wird auch bei Überschreiten von Zählerständen keine Fehlerbedingung gemeldet.

RECY — Recovery. Bei gesetztem Bit werden alle Fehler gemeldet (auch solche, die korrigiert bzw. durch einen erneuten Versuch beseitigt werden könnten!), und auf eine Korrektur durch den Controller wird verzichtet. Standardeinstellung ist ein gelöschtes Bit. Damit werden nur "ernsthafte" Fehler an den Initiator gemeldet.

STAT — Status. Bei gesetztem Bit werden auch Fehler gemeldet, die vom Kontroller durch eigene Maßnahmen behoben werden konnten. Normalerweise geschieht das nicht, weshalb dieses Bit dann auch gelöscht ist!

UNIT ATN — Unit Attention. Wenn dieses Bit gesetzt ist, wird das Target beim ersten Kommando (außer REQUEST SENSE und INQUIRY) nach einem Reset oder Einschalten ein CHECK CONDITION im Statusbyte melden. Bei manchen Platten ist das die Defaulteinstellung, und somit kann man dann von diesen Platten nicht booten. Das TOS ist nämlich auf ein CHECK CONDITION beim Lesezugriff auf den Rootsektor einer Festplatte nicht vorbereitet und interpretiert diesen CHECK CONDITION-Status als "Platte nicht ansprechbar". Bei den meisten Platten kann man dieses Bit "ausknipsen", und damit ist das Problem dann wieder gelöst (wenn nicht die Firmware des Controllers

fehlerhaft ist und dadurch der Zustand dieses Bits immer als gesetzt angesehen wird. Alles schon dagewesen!). Es wird dann nach Einschalten oder Reset kein CHECK CONDITION-Status gemeldet.

Das Byte 3 wird in der Page 0 häufig verwendet, um den Device-Type Qualifier setzen zu können. Siehe dazu auch beim Befehl INQUIRY.

Parameter für Fehlerbehebung (Page Code 01_H)

Bit Byte	7	6	5	4	3	2	1	0
0	Reserv.	Page Code = 01_H						
1	Page-Länge (üblicherweise = 06_H)							
2	AWRE	ARRE	TB	RC	EEC	PER	DTE	DCR
3	Korrekturversuche							
4	Korrekturspanne							
5	Reserviert							
6	Reserviert							
7	Reserviert							

Mit Page-1-Parametern läßt sich bei Platten einstellen, wie im Fehlerfall verfahren werden soll.

AWRE — Automatic Write Reallocation. Bei gesetztem Bit führt der Controller bei einem Schreibfehler wegen eines Mediumdefekts eine Art REASSIGN durch und legt die zu schreibenden Daten in einem Reserveblock ab. Die fehlerhafte Stelle auf dem Medium wird in die Defektliste im Controller eingetragen.

ARRE — Automatic Read Reallocation. Funktioniert genau wie AWRE, nur für Fehler bei Lesezugriffen.

TB — Transfer Block. In Verbindung mit dem DTE-Bit zu sehen. Wenn beide Bits gesetzt sind, wird bei Lesezugriffen auch ein fehlerhafter Block an den Initiator übermittelt. Ist DTE gesetzt, TB aber nicht, wird der fehlerhafte Block nicht gesendet.

RC — Read Continuous. Bei gesetztem Bit wird bei einem Lesezugriff das Aussenden der Daten an den Initiator nicht durch evtl. Pausen für Fehlerkorrekturmaßnahmen unterbrochen. Das bedeutet, daß evtl. auch fehlerhafte Daten gesendet werden können.

EEC	Enable Early Correction. Wenn das Bit gesetzt ist, wird erst versucht, den Fehler durch Fehlerkorrektur nach einem Fehlerkorrekturalgorithmus "rauszurechnen". Erst wenn das nicht klappt, wird ein erneuter Zugriffsversuch aufs Medium gestartet. Wenn das Bit gelöscht ist, wird zuerst mit weiteren Zugriffen versucht (max. Anzahl im Feld "Korrekturversuche" enthalten), den Fehler zu beheben.
PER	Post Error-Bit. Gelöscht: CHECK CONDITION wird nicht erzeugt, wenn es sich um einen Fehler handelt, der im Rahmen der Einstellungen dieser Page behoben werden konnte. Gesetzt: CHECK CONDITION wird für behebbare Fehler ebenfalls erzeugt und der Sense Key für ein folgendes REQUEST SENSE-Kommando auf "Recovered Error" gesetzt.
DTE	Disable Transfer on Error. Bei gesetztem Bit (PER-Bit muß ebenfalls gesetzt sein) wird sofort der CHECK CONDITION-Status geliefert. Datentransfers zum Initiator werden gestoppt. Der fehlerhafte Block wird abhängig vom TB-Bit evtl. noch gesendet! Wenn das Bit gelöscht ist, wird der Datentransfer komplett durchgeführt (wenn evtl. Fehler behoben werden konnten) und erst danach wird ein CHECK CONDITION (falls nötig) erzeugt!
DCR	Disable Correction. Bei gesetztem Bit wird bei einem Fehler nur versucht, durch wiederholte Zugriffe den Fehler zu beheben. Ein "Herausrechnen" des Fehlers wird nicht durchgeführt. Bei gelöschtem Bit wird auch versucht, den Fehler mit Hilfe eines Error Correcting Codes (ECC) zu beseitigen.
Korrekturversuche	Anzahl der Korrekturversuche durch erneutes Lesen, bevor evtl. mit Fehlerkorrekturcode-Einsatz (ECC) gearbeitet wird.
Korrekturspanne	Max. Größe in Bits, die ein Lesefehler "lang" sein darf, bei dem es sich noch lohnt, mit Fehlerkorrekturcode eine Behebung durchzuführen (Werte um 8..10 sind gebräuchlich).

Da im TT nicht mit DISCONNECT und RECONNECT-Messages gearbeitet wird, wird auf den Aufbau der Page 2 nicht weiter eingegangen. Außerdem bieten nur wenige Plattentypen Einstellmöglichkeiten in dieser Page!

Format-Parameter (Page Code 03_H)

Bit / Byte	7	6	5	4	3	2	1	0
0	Reserv.	Page Code = 03_H						
1	Page-Länge (üblicherweise = 16_H)							
Behandlung von Defektstellen								
2–3	(MSB)	Anzahl Spuren pro Zone (Zylinder)						(LSB)
4–5	(MSB)	Anzahl Ausweichsektoren pro Zone						(LSB)
6–7	(MSB)	Anzahl Ausweichspuren pro Zone						(LSB)
8–9	(MSB)	Anzahl Ausweichspuren pro Medium						(LSB)
Angaben zum Spurformat								
10–11	(MSB)	Sektoren/Spur (bei variabler Zahl = 0!)						(LSB)
Angaben zum Sektorformat								
12–13	(MSB)	Bytes pro Sektor						(LSB)
14–15	(MSB)	Interleave						(LSB)
16–17	(MSB)	Track Skew Faktor						(LSB)
18–19	(MSB)	Zylinder Skew Faktor						(LSB)
Angaben zum Plattentyp								
20	SSEC	HSEC	RMB	Reserviert				
21–23	Reserviert							

Track Skew Faktor — Anzahl phys. Sektoren zwischen dem letzten log. Block einer Spur und dem ersten log. Block auf der direkt anschließenden Spur.

Zylinder Skew Faktor — Anzahl phys. Sektoren zwischen dem letzten log. Block eines Zylinders und dem ersten log. Block auf dem direkt anschließenden Zylinder.

SSEC — Soft Sectored. Bei gesetztem Bit ist die Platte softsektoriert. Bit HSEC *muß* dann gelöscht sein!

HSEC — Hard Sectored. Bei gesetztem Bit handelt es sich um eine hardsektorierte Platte. Dann *muß* das SSEC-Bit gelöscht sein!

RMB — Removable. Bei Wechselplatten ist dieses Bit gesetzt, sonst gelöscht.

Viele dieser Angaben sind Read-Only-Daten, d. h., sie werden wohl beim MODE SENSE-Kommando geliefert, können aber nicht geändert werden. Außerdem gilt hier die Regel, daß

in dieser Page gemachte Einstellungen erst nach einem unmittelbar folgenden FORMAT UNIT-Befehl gültig werden!

MODE SENSE

Dieser Befehl ist das Gegenstück zum MODE SELECT-Befehl. Mit ihm lassen sich die Einstellungen des Peripheriegerätes vom Initiator einlesen.

MODE SENSE (OP-Code 1AH)

Bit / Byte	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1	0
1	Logical Unit			Reserviert				
2	PCF		Page Code					
3	Reserviert							
4	Transferlänge							
5	Herstellerspez.		Reserviert				Flag	Link

PCF — Page Control Field. Mit diesen beiden Bits läßt sich festlegen, welche Art von Page-Parametern zurückgeliefert werden soll.

PCF = 00: Report Current Values. Die im Moment eingestellten Parameter sollen zurückgeliefert werden. Das sind entweder jene Parameter, die durch den letzten, erfolgreich durchgeführten MODE SELECT-Befehl eingestellt wurden, oder gespeicherte Werte (Saved Parameter), weil seit dem Einschalten des Geräts noch kein MODE SELECT-Befehl ausgeführt wurde.

PCF = 01: Report Changeable Values. Es wird in den angeforderten Pages lediglich durch gesetzte Bits signalisiert, welche Parameter änderbar sind. Nicht änderbare Bits oder Felder werden mit 0 zurückgegeben.

PCF = 10: Report Default Values. Angeforderte Pages werden mit den Standardeinstellungen ab Werk zurückgeliefert.

PCF = 11: Report Saved Values. Die angeforderten Pages werden mit den zuletzt gespeicherten Einstellungen zurückgegeben.

Page Code — In diesem Feld ist anzugeben, welche Page zurückgeliefert werden soll. Sollen alle Pages geliefert werden, ist hier $3F_H$ einzutragen.

Transferlänge Anzahl der Bytes, die der Initiator bei der auf dieses Kommando folgenden DATA IN-Phase aufnehmen kann. Das Target liefert so lange Daten, bis diese Länge erreicht ist oder bis im Target keine weiteren Daten mehr vorliegen.

MODE SENSE-Daten

Bit / Byte	7	6	5	4	3	2	1	0
0	Anzahl Sense-Daten							
1	Medium-Typ							
2	WP	Reserviert						
3	Blockdeskriptor-Länge (=8)							
Blockdeskriptor(en)								
0	Density Code							
1-3	(MSB)	Blockanzahl						(LSB)
4	Reserviert							
5-7	(MSB)	Blocklänge						(LSB)

WP steht für Write Protected und signalisiert mit einem gesetzten Bit, daß das Medium schreibgeschützt ist.

Medium-Typ, *Density Code*, *Blockanzahl* und *Blocklänge* enthalten die gleichen Angaben, wie bereits beim MODE SELECT-Kommando erläutert.

Die mit der auf dieses Kommando folgenden DATA IN-Phase eingelesenen Daten sind ebenfalls wieder Page-strukturiert.

Der Aufbau dieser Pages ist im Prinzip der gleiche, wie beim MODE SELECT-Kommando bereits beschrieben. Lediglich die beiden obersten Bits im jeweils ersten Page-Code Byte sind reserviert.

Das höchstwertige dieser beiden Bits wird manchmal benutzt, um anzuzeigen, ob die Parameter in der Page auch dauerhaft gespeichert werden können.

Da die Page 4 Parameter zur "Geometry" einer Platte enthält und somit einiges an Informationen über die Platte liefert, soll der Aufbau dieser Page (die nur gelesen werden kann!) hier noch kurz gezeigt werden.

Unveränderbare Plattenparameter (Disk Geometry) Page 04_H

Bit / Byte	7	6	5	4	3	2	1	0
0	Reserviert		Page Code (= 04_H)					
1			Page-Länge (üblicherweise 12_H)					
2–4	(MSB)		Anzahl Zylinder					(LSB)
5			Anzahl Köpfe					
6–8	(MSB)		Startzyl. mit Schreibvorkompensation					(LSB)
9–11	(MSB)		Startzyl. mit reduz. Schreibstrom					(LSB)
12–13	(MSB)		Steprate					(LSB)
14–16	(MSB)		Zylindernummer der Ladezone					(LSB)
17–19			Reserviert					

START/STOP UNIT

Mit diesem Kommando ist es möglich, eine LUN des Targets "Ein" und "Aus" zu schalten.

Bei Platten ist es so möglich, diese mit dem STOP UNIT-Befehl in eine Art Ruhezustand zu versetzen, in dem sie wenig Strom verbrauchen und auch entsprechend weniger Geräusche verursachen.

Platten ohne Autopark-Funktion (heutzutage selten zu finden) lassen sich mit diesem Befehl in eine Art Transportstellung bringen, so daß sie nicht mehr so empfindlich auf Erschütterungen reagieren.

START/STOP UNIT (OP-Code $1B_H$)

Bit / Byte	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1	1
1	Logical Unit			Reserviert				IMMED
2	Reserviert							
3	Reserviert							
4	Reserviert							START
5	Herstellerspez.		Reserviert				Flag	Link

IMMED Bei gesetztem Bit wird direkt nach Einleitung der Aktion (Start oder Stop) das Statusbyte zurückgegeben. Bei gelöschtem Bit wird der Status erst geliefert, wenn das Kommando beendet ist.

START Ein gesetztes Bit bedeutet "START". Bei gelöschtem Bit wird gestoppt!

PREVENT/ALLOW MEDIUM REMOVAL

Dieses Kommando existiert natürlich nur für Peripheriegeräte mit Wechselmedien, wie z. B. Wechselplatten. Mit diesem Kommando läßt sich eine Ver-/Entriegelung des Mediums vornehmen. Damit kann z. B. bei entsprechender Implementation der Systemsoftware ein nicht zulässiger Mediumwechsel verhindert werden.

PREVENT/ALLOW MEDIUM REMOVAL (OP-Code $1E_H$)

Bit Byte	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	0	0	0	1	1	1	1	0
1	Logical Unit			Reserviert				
2	Reserviert							
3	Reserviert							
4	Reserviert							PREVENT
5	Herstellerspez.		Reserviert				Flag	Link

PREVENT Bei gesetztem Bit ist ein Mediumwechsel gesperrt, sonst zulässig! Nach einem Reset ist eine evtl. Verriegelung jedoch wieder aufgehoben!

Doch nun zum praktischen Umgang mit den SCSI-Informationen in Verbindung mit dem TT.

Das SCSI-Interface beim TT

ATARI verwendet im TT für die Bedienung des SCSI-Ports einen alteingeführten Controller-Chip, den "5380". Alle erforderlichen Komponenten für das Betreiben eines SCSI-Ports sind in diesem Chip enthalten.

So ist der Chip durch entsprechende Ausgangsstufen in der Lage, direkt den SCSI-Bus zu treiben. Bis zu 48mA bei einem Low-Pegel von 0,5V "verkraftet" der Chip pro Ausgang ohne zusätzliche Treiber-Chips.

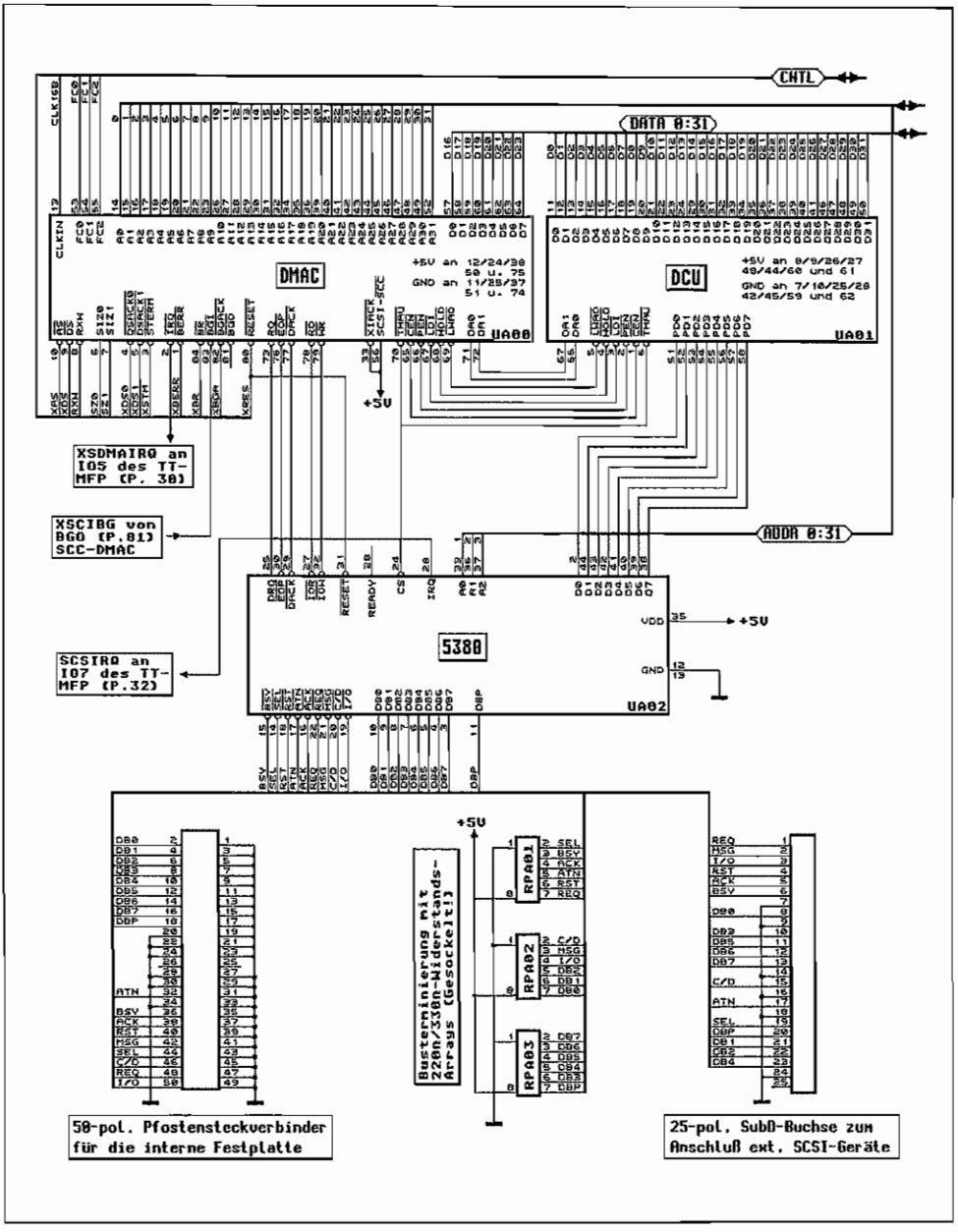


Abb. 6.2: Die SCSI-Hardware des TT

Nur durch Softwareeinstellung ist der 5380 sowohl als Initiator als auch als Target betreibbar. ARBITRATION und RESELECTION sind möglich (aber z. Zt. nicht verwendet).

Der 5380 kann in DMA-Umgebungen verwendet werden, wobei er im TT durch einen eigenen SCSI-DMA-Controller (bestehend aus zwei Custom-Chips) unterstützt wird. Eine Transferrate bis zu 1,5 MByte/s im Asynchronbetrieb (Synchronbetrieb kann der Chip nicht!) ist zu erreichen.

Natürlich kann der 5380 auch einige Interrupts erzeugen. Folgende Interrupts sind möglich: SELECTION/RESELECTION, SCSI-Bus-Reset, Parity-Fehler während eines Datentransfers, SCSI-Bus-DISCONNECTION, Ende des DMA-Transfers oder bei unzulässigen Buszuständen während einer Busphase. Ausgenutzt wird die Interruptfähigkeit des 5380 im TT jedoch nicht. Sein Interrupt-Ausgang IRQ wird lediglich bei manchen Aktionen per Polling abgefragt (Bit 7 des I/O-Ports vom TT-MFP, Adr. $FF FA81, "GPIP_TT").

Die Register des SCSI-Controllers

Der 5380 wird als Peripheriebaustein in Form von acht Registern mit einer Registerbreite von 8 Bit angesprochen. Dabei liegt im TT der 5380 mit seinen Registern auf ungeraden Adressen. Auch hier gilt wieder, daß diese Adressen sowohl im ST-Adreßraum (24 Bit) bei $FF 878X als auch im 32 Bit-Adreßraum der 68030-CPU "ganz oben" bei $FFFF 878X auftauchen. Je nachdem, ob auf den 5380 ein Lese- oder Schreibzugriff stattfindet, erfüllen die Register zum Teil verschiedene Aufgaben.

Adresse	Zugriff	Bezeichnung	Label
$FF 8781	R	Aktueller Inhalt des SCSI-Datenbusses	"s_data"

Bei einem Lesezugriff auf diese Adresse erhält man die Daten, die sich im Moment auf dem SCSI-Datenbus befinden. Wenn mit Parität gearbeitet wird, wird diese immer zu Beginn des Lesezyklusses getestet. Verwendet wird dieses Register während eines Lesevorgangs "von Hand" oder während der ARBITRATION-Phase zur Überprüfung, ob noch ein höherwertiges Device sich um den Bus bemüht.

Adresse	Zugriff	Bezeichnung	Label
$FF 8781	W	Ausgabedaten	"s_data"

Bei einem Schreibzugriff findet man an der gleichen Adresse wie zuvor das Register für die Daten, die über den Bus gesendet werden sollen.

Sowohl während eines normalen Schreibzugriffes durch die CPU auf den SCSI-Bus als auch während DMA-Betrieb wird dieses Register für die Datenausgabe verwendet. Außerdem werden während der ARBITRATION- und SELECTION-Phase über dieses Register die ID-Bits ausgegeben.

Adresse	Zugriff	Bezeichnung	Label
$FF 8783	R/W	Initiator-Befehlsregister	"s_icr"

Mit Hilfe dieses Registers lassen sich bestimmte Bussignale im Initiatorbetrieb beeinflussen und auch bei einem Lesezugriff auswerten. Der Stand des ARBITRATION-Prozesses läßt sich in diesem Register durch entsprechende Bits ebenfalls ablesen.

Obwohl die meisten dieser Bits nur bei Initiatorbetrieb interessant sind, gibt es doch auch einige, die für den Betrieb des Chips als Target verwendet werden (was im TT jedoch nicht vorkommt!).

Initiator-Befehlsregister (**WRITE**)

Bit 7	6	5	4	3	2	1	0
Assert _RST	Test Modus	0	Assert _ACK	Assert _BSY	Assert _SEL	Assert _ATN	Assert Data to Bus

Initiator-Befehlsregister (**READ**)

Bit 7	6	5	4	3	2	1	0
Assert _RST	AIP	LA	Assert _ACK	Assert _BSY	Assert _SEL	Assert _ATN	Assert Data to Bus

Bit 7

Assert _RST (R/W) Aktiviere RESET.
Solange dieses Bit gesetzt ist, wird auf dem SCSI-Bus die RESET-Leitung aktiviert! Mit Einschalten von RESET wird auch der IRQ-Ausgang des 5380 aktiviert! Bei einem Lesezugriff wird der Zustand dieses Registerbits zurückgeliefert.

Bit 6

Test Modus (W) — Nur für Testzwecke gedacht! Im Normalbetrieb muß dieses Bit gelöscht sein!

AIP (R) — Arbitration in Progress. Am Zustand dieses Bits kann abgelesen werden, ob der Chip sich gerade in einer ARBITRATION-Phase befindet. Damit der Chip sich überhaupt an der ARBITRATION-Phase beteiligt, muß das ARBITRATE-Bit (Bit 0) im Betriebsartenregister (s_mode) gesetzt sein. Wenn der Chip sich an der ARBITRATION beteiligt, wird AIP gesetzt, sobald eine BUS FREE-Phase erkannt wurde, der 5380 _BSY aktiviert hat und seine ID auf den Bus gelegt ist.

Bit 5

LA (R) — Lost Arbitration. Der 5380 hatte sich um den SCSI-Bus bemüht, jedoch die ARBITRATION verloren. Damit eine Auswertung des LA-Bits möglich ist, muß das ARBITRATE-Bit im Betriebsartenregister (s_mode) gesetzt sein.

Bit 4

Assert _ACK (R/W) — Aktiviere _ACK-Ltg. Dieses Bit wird von einem Initiator während des Datenaustauschs verwendet, um den _REQ/_ACK-Handshake abzuwickeln. Bei gesetztem Bit wird die _ACK-Leitung aktiviert. Damit der Handshake funktioniert, muß das TARGETMODE-Bit im Betriebsartenregister (s_mode) gelöscht sein.

Bei einem Lesezugriff erhält man die im Moment gültige Einstellung des Registerbits.

Bit 3

Assert _BSY (R/W) — Aktiviere _BSY-Ltg. Mit diesem Bit wird direkt die _BSY-Ltg. gesteuert.

Bei einem Lesezugriff erhält man die im Moment gültige Einstellung des Registerbits.

Bit 2

Assert _SEL (R/W) — Aktiviere _SEL-Ltg. Direkte Beeinflussung der _SEL-Ltg.

Bei einem Lesezugriff erhält man die im Moment gültige Einstellung des Registerbits.

Bit 1

Assert _ATN (R/W) — Aktiviere _ATN-Ltg. Nur wirksam, wenn der 5380 als Initiator arbeitet. Dazu muß das TARGETMODE-Bit im Betriebsartenregister (s_mode) gelöscht sein.

Bei einem Lesezugriff erhält man die im Moment gültige Einstellung des Registerbits.

Bit 0

Assert Data to Bus (R/W) — Bei gesetztem Bit können die Daten im Ausgabedatenregister (s_dat) auf den SCSI-Datenbus gelangen.

Außerdem wird die zugehörige Parität erzeugt und ausgegeben.

Die Datenbusausgänge des 5380 werden bei Initiatorbetrieb jedoch nur dann freigegeben, wenn das TARGETMODE-Bit im Betriebsartenregister (s_mode) gelöscht ist, _I/O-Signal vom Bus deaktiviert ist und die Signale _C/D und _MSG mit den entsprechenden Einstellungen im Target-Befehlsregister (s_tcr) übereinstimmen, also "in Phase liegen".

Dieses Bit sollte bei DMA-Sendebetrieb natürlich ebenfalls gesetzt sein!

Bei einem Lesezugriff erhält man die im Moment gültige Einstellung des Registerbits.

Adresse	Zugriff	Bezeichnung	Label
$FF 8785	R/W	Betriebsartenregister	"s_mode"

Mit einem Schreibzugriff lassen sich die einzelnen Betriebsarten des 5380 einstellen.

Ein Lesezugriff liefert dann die im Moment eingestellte Betriebsart.

Bit 7	6	5	4	3	2	1	0
Block Mode DMA	Target-mode	Enable Parity Check	Enable Parity Interr.	Enable EOP-Inter.	Monitor Busy	DMA Mode	Arbitrate

Blockmode-DMA	Mit diesem Bit wird die Art des Handshakes zwischen DMA-Baustein und 5380-Chip festgelegt. Im TT ist kein Blockmode-DMA möglich. Folglich muß bei allen DMA-Operationen das Bit gelöscht sein!
TARGETMODE	Mit diesem Bit wird festgelegt, ob der 5380 als Initiator (Bit = 0) oder als Target (Bit = 1) arbeiten soll!
Enable Parity Checking	Bei gesetztem Bit werden Paritätsfehler signalisiert. Wenn Paritätsfehler aufgetreten sind, wird dies in Form eines gesetzten Bit 5 im Statusregister angezeigt.
Enable Parity Interrupt	Bei gesetztem Bit wird bei einem Paritätsfehler ein entsprechender Interrupt ausgelöst. Dazu muß natürlich auch das "Enable Parity Checking"-Bit gesetzt sein!
Enable EOP-Interrupt	EOP steht für End of Process. Bei gesetztem Bit reagiert der 5380 damit auf das EOP-Signal vom DMA-Baustein, der damit das Ende eines DMA-Prozesses signalisiert. Es wird bei gesetztem Bit dann ein entsprechender Interrupt ausgelöst.
Monitor Busy	Bei gesetztem Bit wird die _BSY-Leitung des SCSI-Busses überwacht und bei unerwartetem Deaktivieren von Busy ein Interrupt ausgelöst. Wenn das geschieht, werden die untersten sechs Bits des Initiator Befehlsregisters (s_icr) gelöscht, und der 5380 nimmt alle Signale vom SCSI-Bus herunter!
DMA-Mode	Dieses Bit wird benutzt, um einen DMA-Transfer in die Wege zu leiten. Es ist dazu vor dem Beschreiben der DMA-Register des 5380 zu setzen. Ein DMA-Transfer kann durch Rücksetzen dieses Bits abgebrochen werden. Bei einem EOP-Signal vom DMA-Baustein wird dieses Bit allerdings nicht zurückgesetzt!
ARBITRATE	Durch Setzen dieses Bits wird der 5380 veranlaßt, in die ARBITRATION-Phase einzutreten. Dazu muß das Ausgabedatenregister (s_data) die eigene SCSI-Device-ID enthalten. Sobald der 5380 eine BUS FREE-Phase erkennt, wird der Chip versuchen, den Bus "für sich zu gewinnen"! Erfolg oder Mißerfolg und auch der Zustand der ARBITRATION kann ja über die AIP- und LA-Bits im Initiator-Befehlsregister (s_icr) ermittelt werden.

Adresse	Zugriff	Bezeichnung	Label
$FF 8787	R/W	Target Befehlsregister	"s_tcr"

Wenn der 5380 als Target (TARGETMODE-Bit im Betriebsartenregister (s_mode) gesetzt!) betrieben wird, erlaubt dieses Register die Kontrolle über die SCSI-Bus-Transferphasen durch direktes Steuern der _MSG-, _C/D- und _I/O-Leitungen.

Außerdem wird über dieses Register die _REQ-Handshakeleitung direkt gesteuert.

Bei DMA-Betrieb wird durch ein Bit signalisiert, wann das letzte Datenbyte über den SCSI-Bus gesendet wurde (nur 53*C*80-Chip!).

Bit 7	6	5	4	3	2	1	0
Last Byte Sent	nicht be-nutzt	nicht be-nutzt	nicht be-nutzt	Assert _REQ-	Assert _MSG	Assert _C/D	Assert _I/O

Last Byte sent — Nur lesbar! Bei gesetztem Bit ist das letzte Byte auf den SCSI-Bus gegeben worden. Dieses Flag wird gebraucht, weil das "End of DMA"-Bit im Statusregister (s_idstat) lediglich darüber informiert, wann das letzte Byte vom DMA-Baustein übergeben wurde! ATARI verwendet im TT jedoch noch den 5380-Chip, der dieses Bit noch nicht kennt!

Wenn der 5380 als Initiator betrieben wird, müssen während eines Datentransfers die Zustände der Bits 0..2 mit den Zuständen auf den SCSI-Bus-Leitungen übereinstimmen. Sonst entsteht ein Phasen-Mischmasch, äh.., "Phase-mismatch", der, wenn eingeschaltet, zu einem Interrupt führt. Bei Initiatorbetrieb hat das Assert _REQ-Bit keine Funktion!

Adresse	Zugriff	Bezeichnung	Label
$FF 8789	R	Busstatusregister	"s_idstat"

Aus diesem Register können die jeweils aktuellen Zustände der sieben wichtigsten SCSI-Steuer-Leitungen abgelesen werden.

Man erhält dabei jeweils einen "Schnappschuß" der aktuellen Busphase. So kann bei Initiatorbetrieb z. B. ein _REQ-Signal vom Target an dem entsprechenden Bit erkannt werden. Anschließend wird das angeforderte Byte ausgegeben und mit _ACK quittiert.

Bit 7	6	5	4	3	2	1	0
_RST	_BSY	_REQ	_MSG	_C/D	_I/O	_SEL	Parität

Adresse	Zugriff	Bezeichnung	Label
$FF 8789	W	Adreßregister	"s_idstat"

Dieses Register kann während der SELECTION-Phase benutzt werden. Es ist als Maskenregister zu verwenden, in dem das ID-Bit gesetzt wird, auf dessen "Auftauchen" während der SELECTION-Phase gewartet wird. Wenn dieses ID-Bit auf dem Bus erkannt wird und zudem _BSY deaktiviert und _SEL aktiviert ist, wird ein Interrupt veranlaßt. Wenn auf diesen Interrupt verzichtet werden soll, sind alle Bits in diesem Register auf 0 zu setzen!

Adresse	Zugriff	Bezeichnung	Label
$FF 878B	R	Statusregister	"s_dmastat"

In diesem Statusregister findet man zum einen die SCSI-Bussignale wieder, die im Busstatusregister (s_idstat) nicht zu finden sind, und zum anderen sechs weitere Statusbits.

Bit 7	6	5	4	3	2	1	0
End of DMA	DMA Request	Parity Error	IRQ aktiv	Phase match	Busy Error	_ATN	_ACK

End of DMA — Ende eines DMA-Prozesses. Zurückgesetzt wird dieses Bit, wenn das "DMA-Mode"-Bit im Betriebsartregister (s_mode) gelöscht wird.

DMA Request — Damit kann die CPU den Zustand des DRQ-Anschlusses des 5380 überwachen. Wird normalerweise nicht benötigt.

Parity Error — Hierüber wird ein Paritätsfehler während eines Datenempfangs oder während der SELECTION-Phase mitgeteilt. Natürlich nur dann, wenn das "Enable Parity Checking"-Bit im Betriebsartregister (s_mode) gesetzt ist. Gelöscht wird diese Fehlermeldung durch Auslesen des Resetregisters für Interrupts und Parityfehler (s_inircv).

IRQ aktiv	Der Zustand des IRQ-Pins des 5380 kann hier zusätzlich abgelesen werden. Interrupts werden durch Auslesen des Resetregisters für Interrupts und Parityfehler (s_inircv) gelöscht.
Phase match	Dieses Bit ist nur für Initiatorbetrieb interessant und enthält eine ständige Aussage darüber, ob die unteren 3 Bits des Target-Befehlsregisters (s_tcr) mit den SCSI-Signalen _MSG, _C/D und _I/O korrespondieren. Nur bei einem Phase match ist Datentransfer zulässig.
Busy Error	Dieses Bit wird aktiviert, wenn _BSY unerwartet inaktiv wird.
_ATN	Der Zustand der Attention-Leitung.
_ACK	Der Zustand der Acknowledge-Leitung.

Adresse	Zugriff	Bezeichnung	Label
$FF 878B	W	Start DMA-Ausgabe	"s_dmastat"

Bei einem Schreibzugriff auf dieses Register (Wert ist dabei egal!) wird die DMA-Ausgabe gestartet. Vorher muß das "DMA-Mode"-Bit im Betriebsartregister (s_mode) gesetzt sein!

Außerdem sind alle nötigen Einstellungen wie Initiator/Targetbetrieb und "Block-Mode-DMA" oder "Normal DMA-Mode" natürlich ebenfalls vorher vorzunehmen.

Im TT steuert eine eigene SCSI-DMA-Einheit (zwei Custom Chips) den DMA-Betrieb mit dem 5380!

Adresse	Zugriff	Bezeichnung	Label
$FF 878D	W	Start Target-DMA-Eingabe	"s_targrcv"

Wenn der 5380 als Target arbeitet, wird für lesenden DMA-Betrieb (vom SCSI-Bus lesen!) die Aktion durch einen Schreibzugriff auf dieses Register (Wert egal) gestartet.

Adresse	Zugriff	Bezeichnung	Label
$FF 878D	R	Eingaberegister	"s_targrcv"

Über einen Lesezugriff auf dieses Register erhält man die Daten vom SCSI-Datenbus. Jeweils bei Aktivierung von _ACK (bei Target-DMA-Eingabe) oder bei Aktivierung von _REQ (bei

Initiator-DMA-Eingabe) wird das Datebyte auf dem SCSI-Bus in diesem Register zwischengespeichert (latched).

Der DMA-Baustein liest die Daten Byte für Byte aus diesem Register aus. Dabei werden die _IOR- und _DACK-Anschlüsse entsprechend durch den DMA-Baustein bedient.

Adresse	Zugriff	Bezeichnung	Label
$FF 878F	W	Start Initiator-DMA-Eingabe	"s_inircv"

Ein Schreibzugriff (Wert ist egal) startet den lesenden DMA-Betrieb bei Programmierung des 5380 als Initiator. Für DMA-Betrieb muß natürlich wieder das "DMA-Mode"-Bit im Betriebsartregister (s_mode) und der 5380 auf Initiatorbetrieb eingestellt sein.

Adresse	Zugriff	Bezeichnung	Label
$FF 878F	R	Resetreg. für Interrupts + Parityfehler	"s_inircv"

Interrupts und Parity-Fehleranzeigen werden durch Lesen dieses Registers gelöscht. Dabei erhält man keinen sinnvollen Wert zurück. Es kommt nur auf den Lesezugriff an!

Damit's einfacher wird – SCSI-Datentransfer mit DMA-Unterstützung

Für den Datenaustausch während der Datentransferphasen hat ATARI dem 5380 eine eigene DMA-Einheit zur Seite gestellt. Gleiches gilt für den im TT vorhandenen Serial Communications Controller SCC.

Da der 5380 Daten nicht in der auf dem Systembus verwendeten Long-Form (32 Bit-Zugriff) verarbeiten kann, ist die DMA-Einheit für die Umsetzung zuständig. Dabei arbeitet der DMA-Baustein immer mit zwei Longs. Während also die DMA-Einheit darauf wartet, das eine Long auf den Systembus zu geben, wird schon das nächste Long im DMA-Baustein zusammengesetzt. Wenn das zweite Long bereits gefüllt sein sollte, bevor der DMA-Baustein die Kontrolle über den Systembus wieder abgegeben hat, wird das zweite Long direkt hinterher geschrieben.

SCSI-DMA-Betrieb ist dabei auf jede RAM-Adresse möglich. Also im Gegensatz zum DMA-Betrieb mit ACSI-Geräten gibt es keinerlei Beschränkung auf den Bereich des ST-RAMs.

Für den Programmierer stellt sich der DMA-Kanal folgendermaßen dar:

- Ein Status- und Control-Register in Word-Breite, auf das schreibend und lesend zugegriffen werden kann.
- Ein 32-Bit-Adreßregister (Pointer) für die DMA-Adresse (liegt in Form von vier Registern mit Bytebreite vor).
- Ein Restdatenregister, welches benutzt wird, um mit CPU-Hilfe Transfers auf nicht im Long-Raster (durch 4 teilbare) liegende Adressen zu ermöglichen. Gleiches gilt für DMA-Transfers mit Blockgrößen, die nicht im Long-Raster (4-Byte-Raster) liegen.
- Ein 32-Bit-Bytezähler für DMA-Daten. Dieses Zählregister wird durch Kombination von vier Bytes gebildet.

Die SCSI-DMA-Register

Die Registeradressen des SCSI-DMA-Chips liegen sowohl am oberen Ende des ST-Adreßraums ($FF 87XX) als auch ganz oben im TT-Adreßraum ($FFFF 87XX). Angegeben ist hier jeweils die Lage im ST-Adreßraum!

Adresse	Zugriff	Bezeichnung	Label
$FF 8701	R/W	DMA-Address-Pointer (Highest Byte)	"tt_dmabas"
$FF 8703	R/W	DMA-Address-Pointer (High Byte)	"tt_dmabas"+2
$FF 8705	R/W	DMA-Address-Pointer (Low Byte)	"tt_dmabas"+4
$FF 8707	R/W	DMA-Address-Pointer (Lowest Byte)	"tt_dmabas"+6
$FF 8709	R/W	DMA-Bytezähler (Highest Byte)	"tt_dmacnt"
$FF 870B	R/W	DMA-Bytezähler (High Byte)	"tt_dmacnt"+2
$FF 870D	R/W	DMA-Bytezähler (Low Byte)	"tt_dmacnt"+4
$FF 870F	R/W	DMA-Bytezähler (Lowest Byte)	"tt_dmacnt"+6

Da diese vier Byte breiten Register alle so schön hintereinanderliegen, kann für den Long-Zugriff darauf in Maschinensprache ein "movep.l"-Befehl benutzt werden!

Adresse	Zugriff	Bezeichnung	Label
$FF 8710	R	Restdatenregister (High-Word)	"tt_dmarsd"
$FF 8712	R	Restdatenregister (Low-Word)	"tt_dmarsd"+2

Wenn am Schluß einer DMA-Operation kein voller Long mehr in den Speicher übertragen wurde, muß hier der Rest an Datenbytes "per Hand" ausgelesen und durch die CPU an die

richtigen Adressen gebracht werden! Wie viele Restbytes das sind, erfährt man aus den letzten beiden Adreßbits des DMA-Address-Pointers.

Die Startadresse n für die Restbytes ergibt sich aus dem Inhalt des DMA-Address-Pointers minus der Anzahl an Restbytes. Die Restbytes sind "linksbündig" im Restdatenregister angeordnet. Also gehört das Highbyte im High-Word an die Startadresse n. Das Lowbyte im Highword kommt nach Startadresse n+1 usw.!

Adresse	Zugriff	Bezeichnung	Label
$FF 8714	R/W	Kontrollregister (Word)	"tt_dmactl"

Obwohl dieses Register mit Wordzugriffen bearbeitet wird, sind nur die untersten 8 Bits benutzt. Man kann deshalb auch mit Bytezugriffen auf die Adresse $(FF)FF 8715 arbeiten, um diese Kontrollbits zu beeinflussen bzw. auszuwerten. Hier deshalb nur die interessanten unteren 8 Bits des Registers:

Bit 7	6	5	4	3	2	1	0
Bus Error	Byte count zero	Reser-viert	Reser-viert	Reser-viert	Reser-viert	DMA Enable	DMA-Direc-tion

Bus Error — Ein gesetztes Bit weist auf einen Busfehler während des DMA-Betriebs hin. Durch Auslesen dieses Registers wird der Busfehler-Status wieder zurückgesetzt. Diese Fehlersignalisierung geht auch an den TT-MFP, I/O-Port #5 und kann bei entsprechender Programmierung einen Interrupt auslösen.

Byte count zero — Ein gesetztes Bit zeigt an, daß der DMA-Bytezähler auf Null gezählt hat, also alle Bytes transferiert sind.

DMA-Enable — Bei gesetztem Bit ist der DMA-Baustein "eingeschaltet".

DMA-Direction — Bit gesetzt: Schreiben an SCSI-Controller. *Bit gelöscht*: Lesen von SCSI-Controller.

Um einen DMA-Betrieb durchzuführen, ist zunächst die Transferrichtung im DMA-Baustein (Bit 0 in "tt_dmactl") einzustellen. Anschließend wird die Basisadresse im DMA-Address-Pointer ("tt_dmabas") gesetzt und die Zahl der zu übertragenden Bytes im DMA-Bytezähler ("tt_dmacnt") eingestellt.

Wenn dann auch der SCSI-Controller entsprechend programmiert ist, wird durch Setzen des "DMA-Enable"-Bits in "tt_dmactl" der DMA-Prozeß gestartet.

Die Programmierung des TT-SCSI-Ports

Das nachfolgende Listing zeigt in Form einer kommentierten Beispielroutine den Umgang mit der SCSI-Hardware im TT. Der TT arbeitet dabei als Initiator in einer Single-Initiator-Umgebung (also ohne ARBITRATION-Phase) und liest mit dem INQUIRY-Befehl die Daten von dem SCSI-Device-#0 in einen Buffer ein.

Bei diesem Code-Beispiel ist also sowohl die SELECTIONS-Phase, die COMMAND-Phase und anschließend eine DATA IN-Phase mit abschließender STATUS- und MESSAGE-Phase enthalten.

```
;-------------------------------------------------------------
; Beispielroutine für die Programmierung des TT-SCSI-Ports
; von Hans-Dieter Jankowski für ATARI ST/STE/TT Profibuch
;-------------------------------------------------------------

device          equ  2          ; Target-ID (0 = int. Platte)
trlength        equ  $A0        ; 160 Bytes Einlesen reicht!

kurz            equ  200        ; Timeout-Wert für 1 Sekunde
lang            equ  400        ; Timeout-Wert für 2 Sekunden
bsy_da          equ  13         ; Timeout-Wert f. SELECTION
(~250ms)

_hz_200         equ  $4BA       ; Adr. 200 Hz-Systemtimer

GPIP_TT         equ  $FFFFFA81  ; TT-MFP Portregister

tt_dmabas       equ  $FFFF8701  ; SCSI-DMA-Addreß-Pointer
tt_dmacnt       equ  $FFFF8709  ; SCSI-DMA-Bytezähler
tt_dmarsd       equ  $FFFF8710  ; SCSI-DMA-Restdatenbytes
tt_dmactl       equ  $FFFF8714  ; SCSI-DMA-Controlregister

s_data          equ  $FFFF8781  ; Erstes Register im 5380
s_icr           equ  s_data+2
s_mode          equ  s_data+4
s_tcr           equ  s_data+6
```

```
s_idstat   equ   s_data+8
s_dmastat  equ   s_data+$A
s_targrcv  equ   s_data+$C
s_inircv   equ   s_data+$E

;-------------------------------------------------------------------
; Für das SCSI-Device-Nr. "device" werden die INQUIRY-Daten ein-
; gelesen
; Die per DMA gelesenen Daten stehen dann im Speicher ab Adr.
; "buffer"
; Bei korrekter Ausführung liefert d0 im Low-Word das Statusbyte
; und im High-Word das MESSAGE-Byte zum Abschluß der Operation.
; Ein neg. Wert in d0 weist auf eine Zeitüberschreitung in be-
; stimmten Phasen hin.
;-------------------------------------------------------------------
inquiry:
     lea       s_data,a5       ; Ptr. auf 5380-Register setzen
     lea       tt_dmabas,a6    ; Ptr. auf SCSI-DMA-Baustein setzen
     lea       _hz_200,a4      ; In a4 Ptr. auf Systemtimer halten

;-  Erstmal testen, ob der SCSI-Bus zur Zeit frei ist
;
     move.l    (a4),d7         ; Aktuellen Systemtimerwert holen
     addi.l    #kurz,d7        ; Bis zu diesem Timerwert versu-
                               ; chen.
chk_bsy0:
     btst      #6,8(a5)        ; _BSY-Leitung testen
     beq.s     doselect        ; _BSY aktiviert?
     cmp.l     (a4),d7         ; Timerwert mit Endwert vergleichen
     bcc.s     chk_bsy0
     moveq     #-2,d0          ; Dauert zu lange, also raus hier.
     bra       error
;
;-  Nix mehr los auf'm SCSI-Bus! Also dann in die SELECTION-Phase!
;
doselect:
     moveq.l   #0,d0           ; 4 Bytes mit Null in Reg. D0!
     movep.l   d0,2(a5)        ; s_icr/s_mode/s_tcr/s_idstat
                               ; löschen
     bset      #device,(a5)    ; Target-ID-Bit in s_data einstel-
                               ; len.
```

```
        move.b      #1,2(a5)        ; ID-Bit auf SCSI-Bus ausgeben.
        move.b      #5,2(a5)        ; _SEL-Ltg. aktivieren
        move.l      (a4),d7         ; Aktuellen Systemtimerwert holen
        addi.l      #bsy_da,d7      ; In 250ms muß Targ. sich melden
chk_bsy1:
        btst        #6,8(a5)        ; _BSY-Leitung testen
        bne.s       docmnd          ; _BSY aktiviert?
        cmp.l       (a4),d7         ; Timerwert mit Endwert vergleichen
        bcc.s       chk_bsy1        ; Dauert zu lange, also raus hier.
        moveq       #-3,d0
        bra         error
;
;-      Target hat sich gemeldet (BUSY ist aktiviert worden). Also ab
;       in die COMMAND-Phase.
;
docmnd:
        move.b      #0,2(a5)        ; Initiator deaktiviert alle Ltg.
        move.w      #0,tt_dmactl    ; DMA-Baust. auf Lesen einstellen

        move.l      #buffer,d0      ; Anfangsadr. des DMA-Puffers in
        movep.l     d0,(a6)         ; DMA-Address-Ptr. ablegen

        move.l      #trlength,d0    ; Transferlänge in
        movep.l     d0,8(a6)        ; DMA-Bytezähler ablegen
;
;-      DMA-Baustein ist vorbereitet! Jetzt "per Hand" die Command-
;       Bytes ans Target übermitteln. Dabei wird der _REQ/_ACK-
;       Handshake unter CPU-Kontrolle erledigt!
;       Die Aktion sollte nicht länger als "kurz" x 5ms dauern, sonst
;       Abbruch und Sprung in die Fehlerroutine.
;
        lea         cmdtable,a0     ; Ptr. auf 1. Byte vom SCSI-
                                    ; Kommando
        moveq       #5,d1           ; (Anzahl-1) der zu sendenden Bytes

        move.l      (a4),d7         ; Akt. Sys-Timerwert holen und End-
        addi.l      #kurz,d7        ; wert berechnen (steht in d7)

        move.b      #2,6(a5)        ; "Assert _C/D" in s_tcr (CMD-
                                    ; Phase!)
        move.b      #1,2(a5)        ; "Assert Data to Bus" in s_icr
```

```
sndcmd:
     bsr       w4reqa          ; Auf _REQ-Aktivierung warten
     bpl.s     gotreq          ; _REQ aktiviert?
     moveq     #-4,d0          ; Nein!
     bra       error           ; Dann raus hier!
gotreq:
     move.b    (a0)+,(a5)      ; Command-Byte auf SCSI-Bus legen.
     bset      #4,2(a5)        ; _ACK aktivieren (Bit 4 in s_icr)
     bsr       w4reqd          ; Auf _REQ-Deaktivierung warten
     bpl.s     nxtbyte         ; _REQ wieder deaktiviert?
     moveq     #-5,d0          ; Nein!
     bra       error           ; Dann raus hier!

nxtbyte:
     bclr.b    #4,2(a5)        ; Target hat Byte! _ACK
                               ; deaktivieren!
     dbra      d1,sndcmd       ; Schleife, bis alle Bytes raus
                               ; sind!
     move.b    #0,2(a5)        ; Initiator deaktiviert alle Ltg.
;
;-   Command-Descriptor-Block wurde erfolgreich übertragen!
;    Jetzt wird auf DMA-Betrieb geschaltet, und das Target
;    übernimmt die Steuerung der DATA IN-Phase.
;
     move.b    #1,6(a5)        ; "Assert _I/O" in s_tcr (DATA IN!)
     tst.b     14(a5)          ; RESET Int.+Parity-Fehler
                               ; (s_inircv)
     move.b    #2,4(a5)        ; "DMA-Mode" in s_mode einstellen
     move.w    #2,tt_dmactl    ; DMA-Baustein freigeben (DMA
                               ; Enable)
     move.b    #0,14(a5)       ; Start Initiator-DMA-Eingabe!
;
;-   Nun müssen wir uns in Geduld fassen, bis die DMA-Phase
;    beendet ist. Dazu wird gewartet, bis der 5380 einen IRQ an
;    den TT-MFP gibt. Da alle anderen Interrupts des 5380 ausge-
;    schaltet sind, kann es sich bei einem 5380-Interrupt nur um
;    die "Fertigmeldung" des 5380 handeln!
;
     move.l    (a4),d7         ; Akt. Sys-Timerwert holen und End-
     addi.l    #lang,d7        ; wert berechnen (steht in d7)
```

```
wait4int:
     btst       #7,GPIP_TT      ; Auf Int. vom 5380 oder DMA-
                                ; Baustein
     bne.s      scsint          ; warten. Dabei Zeitlimit im Auge
     btst       #5,GPIP_TT      ; behalten!
     beq.s      dmaint
     cmp.l      (a4),d7
     bcc.s      wait4int
     moveq      #-6,d0
     bra        error           ; Zeitüberschreitung! Raus hier!
dmaint:
     btst       #7,tt_dmactl+1  ; Bus error-Bit testen!
     beq.s      wait4int        ; Bus error bei DMA aufgetreten?
     moveq      #-7,d0
     bra        error           ; Ja! Dann aber raus hier!
scsint:
     tst.b      14(a5)          ; RESET Int.+Parity-Fehler
                                ; (s_inircv)
     moveq.l    #0,d0
     move.w     d0,tt_dmactl    ; DMA-Baustein ausschalten
     move.b     d0,4(a5)        ; 5380 aus DMA-Betrieb holen
                                ; (s_mode)
     move.b     d0,2(a5)        ; Alle SCSI-Ltg. deaktivieren
                                ; (s_icr)
;
;-   Jetzt wird das Statusbyte vom Target geholt.
;    Da evtl. ein MESSAGE-Byte folgt, wird anschließend auf die
;    MESSAGE IN-Phase gewartet und das MESSAGE-Byte eingelesen!
;
     move.l     (a4),d7         ; Akt. Sys-Timerwert holen und End-
     addi.l     #kurz,d7        ; wert berechnen (steht in d7)
     move.b     #3,6(a5)        ; _C/D + _I/O aktivieren (s_tcr)
     bsr        w4reqa          ; _REQ vom Target abwarten
     bpl.s      getstat         ; _REQ aktiviert?
     moveq      #-8,d0          ; Nein!
     bra        error           ; Dann raus hier!

getstat:
     move.b     (a5),d0         ; Statusbyte (s_data) nach d0
                                ; holen.
     bset       #4,2(a5)        ; _ACK aktivieren (s_icr).
```

```
        bsr     w4reqd          ; _REQ-Deaktivierung abwarten
        bpl.s   getmsg          ; _REQ deaktiviert?
        moveq   #-9,d0          ; Nein!
        bra     error           ; Dann raus hier!
;
;-      Jetzt wird auf die MESSAGE IN-Phase gewartet und das
;       MESSAGE-Byte eingelesen!
;
getmsg:
        move.b  #0,2(a5)        ; Alle Initiatorltg. deaktivieren
        move.b  #7,6(a5)        ; _MSG, _C/D und _I/O sind bei
w4msg:
        btst    #3,8(a5)        ; der MESSAGE IN-Phase aktiviert!
        bne.s   msg_in          ; MESSAGE IN-Phase erreicht?
        cmp.l   (a4),d7         ; Noch nicht! Zeitüberwachung!
        bcc.s   w4msg           ; Zeitüberschreitung?
        moveq   #-10,d0         ; Ja!
        bra     error           ; Dann raus hier!

msg_in:
        bsr     w4reqa          ; _REQ-Aktivierung testen
        bpl.s   hole_msg        ; _REQ aktiviert?
        moveq   #-11,d0         ; Nein!
        bra     error           ; Dann raus hier!

hole_msg:
        swap    d0              ; Statusbyte ins High-Word swappen
        move.b  (a5),d0         ; MESSAGE-Byte ins Low-Word holen
        swap    d0              ; Statusbyte wieder ins Low-Word
        bset    #4,2(a5)        ; _ACK aktivieren (s_icr).
        bsr     w4reqd          ; Warten, bis _REQ deaktiviert ist.
        bpl.s   das_wars        ; _REQ deaktiviert?
        moveq   #-12,d0         ; Nein!
        bra     error           ; Dann raus hier!

das_wars:
        move.b  #0,2(a5)        ; Initiator deaktiviert alle Ltg.
;
;---- Wenn der DMA-Transfer nicht auf einer Long-Grenze endete,
;       befinden sich im DMA-Restdatenreg. noch ein paar Bytes, die
;       "von Hand" in den Bufferspeicher gebracht werden müssen!
```

```
;
     movep.l    (a6),d1          ; Aktuell. DMA-Adr.-Ptr. holen
     move.l     d1,d2            ; Kopie davon in d2
     andi.w     #3,d2            ; Anzahl Bytes im Restdatenregister
     beq.s      ende             ; Keine (Transf. auf Long-Grenze)?
     subq.w     #1,d2            ; Doch! Einen weniger wg. Schleife!
     andi.b     #$FC,d1          ; Letzte Long-Grenze ansteuern.
     move.l     d1,a0            ; Zielpointer in a0
     lea        tt_dmarsd,a1     ; Quellptr. (Restreg.) nach a1
restloop:
     move.b     (a1)+,(a0)+      ; Restliche Bytes aus Restdatenreg.
     dbra       d2,restloop      ; in den Buffer übertragen.
     bra        ende

;--- Fehlerausstieg! ----------------------------------------------
;    Bei irgendeinem Fehler wird zur Sicherheit ein RESET auf
;    dem SCSI-Bus ausgeführt!
;
error:
     move.l     (a4),d7          ; Akt. Sys-Timerwert holen und End-
     addi.l     #kurz,d7         ; wert berechnen (steht in d7)
     move.b     #$80,2(a5)       ; _RST-Ltg. aktivieren
wait1:
     cmp.l      (a4),d7          ; Timerende testen
     bcc.s      wait1            ; Timer abgelaufen?
     move.b     #0,2(a5)         ; Ja! Dann _RST-Ltg. deaktivieren
     move.l     (a4),d7          ; Akt. Sys-Timerwert holen und End-
     addi.l     #kurz,d7         ; wert berechnen (steht in d7)
wait2:
     cmp.l      (a4),d7          ; Timerende testen
     bcc.s      wait2            ; Timer abgelaufen?
ende:
     rts

;--- Unterprogramm:  Warten auf die Aktivierung von _REQ ----
;
;    In:   d7.l =    Timerendwert
;          a4.l =    Pointer auf _hz_200
;    Out:  d0   =    0  : Alles klar! _REQ ist aktiviert
;                    -1 : Zeitüberschreitung
```

```
w4reqa:
     moveq     #0,d0          ; Default ist kein Timeout!
     btst      #5,8(a5)       ; _REQ-Ltg. überwachen.
     bne.s     gtrqend        ; _REQ vom Target aktiviert?
     cmp.l     (a4),d7        ; Nein! Zeitüberwachung!
     bcc.s     w4reqa         ; Zeitüberschreitung?
     bra.s     gtrqerr

;—— Unterprogramm:  Warten auf die Deaktivierung von _REQ —
;
;    In:  d7.l =    Timerendwert
;         a4.l =    Pointer auf _hz_200
;    Out: d0   =    0  : Alles klar! _REQ ist aktiviert
;                   -1 : Zeitüberschreitung
w4reqd:
     moveq     #0,d0          ; Default ist kein Timeout!
     btst      #5,8(a5)       ; _REQ-Ltg. überwachen.
     beq.s     gtrqend        ; _REQ vom Target deaktiviert?
     cmp.l     (a4),d7        ; Nein! Zeitüberwachung!
     bcc.s     w4reqd         ; Zeitüberschreitung?
gtrqerr:
     moveq     #-1,d0         ; Ja! Mit -1 in d0 raus
gtrqend:
     rts

     .data
;
;—— Hier folgt der komplette Command-Descriptor-Block (6-Byte-CDB)
cmdtable:
     dc.b      $12            ; Inqiry-Befehl
     dc.b      $00            ; LUN = 0
     dc.b      $00
     dc.b      $00            ; Reserved Bytes
     dc.b      trlength       ; max. Transferlänge
     dc.b      $00            ; Kein Linked Command!

     .bss
;
;—— In den folgenden Puffer werden die INQUIRY-Daten eingelesen
buffer:
     ds.w      $100           ; 256-Bytes Puffer reicht aus!
```

Kapitel 7: Seriell, aber schnell! – Der SCC macht's möglich

Im Zeitalter der schnellen Kommunikation wollte wohl auch ATARI seine neuen Maschinen (TT und MEGA STE) bei den Kommunikationsschnittstellen etwas aufwerten. So findet sich dann auch im TT (und MEGA STE) ein eigener Schnittstellen-Steuerbaustein vom Typ 85C30 wieder. Dieser Serial Communications Controller (kurz SCC) übernimmt eine ganze Menge Steuerungsaufgaben, wenn es um den Austausch von seriellen Daten geht.

Der SCC besitzt zwei voneinander unabhängige duplexfähige Kanäle, die im TT sogar durch einen eigenen DMA-Chipsatz (gleiche Bausteine wie beim SCSI-DMA-Chipsatz) für schnellen Datentransfer unterstützt werden. Die beiden Kanäle erlauben die Realisierung von Multiprotokoll-Schnittstellen für die gängigsten Datenübertragungsteuerungsverfahren.

Dabei besitzt der SCC für jeden Kanal einen eigenen Baudratengenerator, mit dem im asynchronen Start/Stop-Betrieb bis zu 250 KBit/s möglich sind (im TT nur 125 KBit/s!). Im Start/Stop-Verfahren können 5..8 Bit-Zeichen mit 1/1,5 oder zwei Stop-Bits und optionaler Erzeugung/Auswertung eines Paritätsbits verwendet werden.

Im Synchronbetrieb (verbundene Computer sind durch Mitübertragung der Taktsignale immer bitsynchron!) erlaubt der SCC DÜ-Raten bis zu 4 MBit/s, wobei aber durch einen Primärtakt PCLK im TT von 8 MHZ "nur" maximal 2 MBit/s erreichbar sind. Wenn im Synchronbetrieb die Taktsignale nicht mitübertragen werden, kann der SCC durch eine Digital Phase Locked Loop-Einheit (DPLL) aus den Datenflanken der empfangenen Bits den Takt rekonstruieren und damit die einlaufenden Daten abtasten als auch diesen Takt für das Aussenden der eigenen Sendedaten verwenden. Dazu braucht der DPLL einen Takt, der bei der 16fachen bzw. 32fachen Baudrate der Empfangsdaten liegt. Im TT liegt die realisierbare DÜ-Rate wegen der höchstmöglichen Taktfrequenz für den SCC von 8 MHz bei max 125 KBit/s.

Vom SCC unterstützte Datenübermittlungs-Steuerungsverfahren

Diese Steuerungsverfahren regeln unter anderem die Organisation des Datenflusses während der Datenaustausch-Phase. Da es hierbei unter anderem zur Blockbildung der zu übermittelnden Daten kommt und Sicherungsinformationen zur Fehlererkennung durch die Gegenstation eingefügt werden, kann ein Teil dieser Aktionen durch den Schnittstellenbaustein des Computers erledigt werden.

Asynchrone Steuerungsverfahren

Das am häufigsten verwendete asynchrone Steuerungsverfahren ist das Start/Stop-Verfahren. Dabei wird jedes Zeichen (beim SCC 5..8 Bit plus evtl. Paritätsbit) als eine Einheit (Block) betrachtet. Zwischen den einzelnen übermittelten Zeichen können unterschiedlich lange Ruhepausen auftreten.

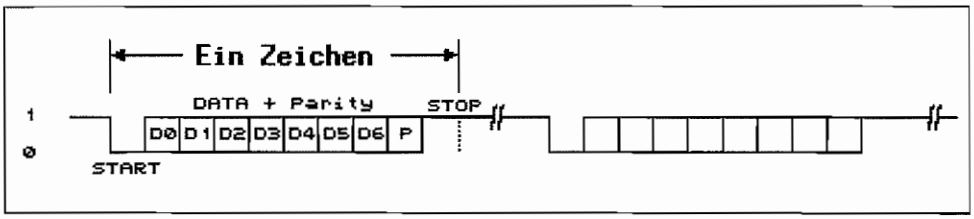


Abb. 7.1: Das Start-/Stop-Verfahren

Im Ruhezustand befindet sich die Sendedatenleitung auf log. 1. Sobald ein Zeichen gesendet werden soll, wird für eine vereinbarte Zeit (Einheitsschrittdauer. Anzahl der Einheitsschritte je Sekunde ergibt die Baudrate!) eine log. 0 ausgegeben. Auf diesen Startschritt folgen dann die eigentlichen Informationsschritte eines Zeichens plus evtl. einer Paritätsinformation (da hier ausschließlich Signale mit zwei Kennzuständen, log. 1 und log. 0, betrachtet werden, kann man auch für Schritt gleich Bit setzen!). Die Abtastung der Empfangsdatenleitung erfolgt sinnvollerweise jeweils in Bitmitte. Dazu erzeugt sich der Empfänger Abtastsignale aus einer eigenen Taktquelle, die ein Vielfaches der Datenübertragungsrate beträgt (meistens 16fache Datenrate). Die Abtastung wird jeweils durch den Startschritt neu gestartet!

Nach Übermittlung aller Datenbits eines Zeichens geht die Sendedatenleitungen für eine vereinbarte Mindestdauer wieder in den Ruhezustand log. 1, bevor das nächste Zeichen (wieder mit einem Startbit beginnend) ausgesendet werden darf. Diese Mindestdauer für den Ruhezustand zwischen zwei Zeichen (in Bitlängen ausgedrückt) beträgt mindestens 1 Stopbit. Der SCC kann auf 1, 1,5 oder 2 Stopbits eingestellt werden.

Synchrone Steuerungsverfahren

Bei der Verwendung von Steuerungsverfahren für synchrone Übertragung ist gewährleistet, daß Taktinformationen zusammen mit den Daten übermittelt werden oder daß bei Verwendung von Modemverfahren zur Datenübermittlung der Takt im Modulationsprozeß mit "eingepackt" wird. So kann also davon ausgegangen werden, daß Bitsynchronismus zwischen

den beteiligten Endstellen besteht. Der Empfänger kann anhand des mitgelieferten Takts die sichere Abtastung der Bitinformation vornehmen.

Wenn die Bitsynchronität besteht, muß in der nächsten Phase der Zeichensynchronismus hergestellt werden. Das heißt, daß der Empfänger herausfinden muß, wie die einzelnen empfangenen Bits zu Gruppen (Zeichen) zusammenzufassen sind. Dazu hat der Empfänger den einlaufenden Bitstrom auf charakteristische Synchronisierzeichen zu untersuchen.

Zeichenorientierte Steuerungsverfahren

Hierbei werden Daten in Zeichenblöcken (üblicherweise Textinformation) zusammengefaßt und übertragen. Jeder Textblock wird durch ein oder mehrere 8 oder 16 Bit-Synchronzeichen eingeleitet. "Umrahmt" sind Textblöcke durch Steuerzeichen, welche den Beginn (STX = Start of Text) und das Ende (ETX = End of Text) des Informationsblocks kenntlich machen.

Meistens werden die übermittelten Blöcke noch um Prüfzeichen ergänzt, die durch spezielle Sicherungsverfahren gebildet werden. Anhand dieser Prüfzeichen und einer auf gleiche Weise im Empfänger durchgeführten Prüfung können Fehler in der Übertragung erkannt und eine Blockwiederholung veranlaßt werden. Dabei wird sowohl das Längsparitätsverfahren (über

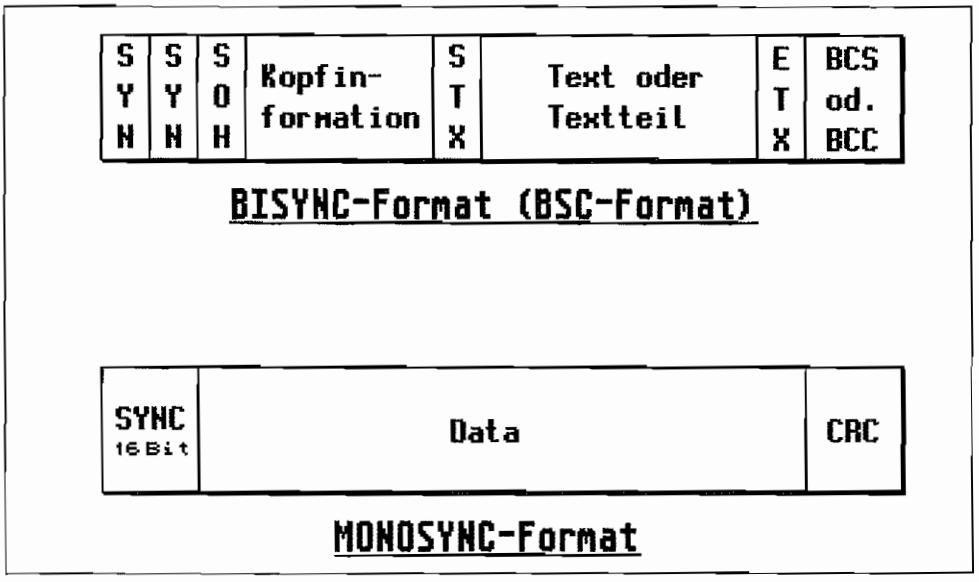


Abb. 7.2: Zeichenorientierte Steuerungsverfahren

alle Bits gleicher Wertigkeit eines Textblocks wird jeweils ein Paritätsbit gebildet und die Zusammenfassung aller so gewonnenen Paritätsbits als Blockprüfzeichen mitübertragen) als auch das CRC-Verfahren verwendet ("Berechnung" einer 16-Bit-Prüfbitfolge aus allen Bits des Textblocks und Anhängen an den Datenblock in Form von zwei 8 Bit-Zeichen). Der SCC kann die Prüfzeichen nach dem CRC-Verfahren nach zwei verschiedenen "Berechnungsregeln" (CRC-16 und CRC-CCITT) ermitteln und auswerten.

Zeichenorientierte Steuerungsverfahren arbeiten vom Prinzip her nur halbduplex, auch wenn die Übertragungsstrecke und die beteiligten Datenstationen duplexfähig sind. Es ist nämlich jeweils nach Aussenden eines Textblocks eine Quittung von der Gegenstelle erforderlich, bevor der nächste Block gesendet wird. Eine weitere Einschränkung besteht darin, daß üblicherweise eine Codebindung besteht. Außerdem werden sowohl die Textinformationen als auch die Steuerzeichen aus dem gleichen Code benutzt, so daß normalerweise im Text keine Steuerzeichen vorkommen dürfen! Es gibt allerdings Ausnahmen, indem durch ein spezielles Zeichen (als Data Link Escape = DLE bezeichnet) die Steuerzeichen als solche kenntlich gemacht werden.

Bitorientierte Steuerungsverfahren

Auch hier werden die Daten zu Blocks zusammengefaßt. Dabei ist aber keinerlei Bindung an einen bestimmten Code erforderlich, sondern es wird mit Bitblöcken gearbeitet. Die weitverbreiteten Steuerungsverfahren SDLC/HDLC (SDLC = Synchronous Data Link Control von IBM / HDLC = High-Level Data Link Control, vom CCITT empfohlen) verwenden zum Informationsaustausch einen standardisierten Blockaufbau, den Frame. Die Unterschiede zwischen SDLC und HDLC sind von geringer Natur und bestehen aus Abweichungen in der Behandlung einiger Bits im Adress- und Control-Feld.

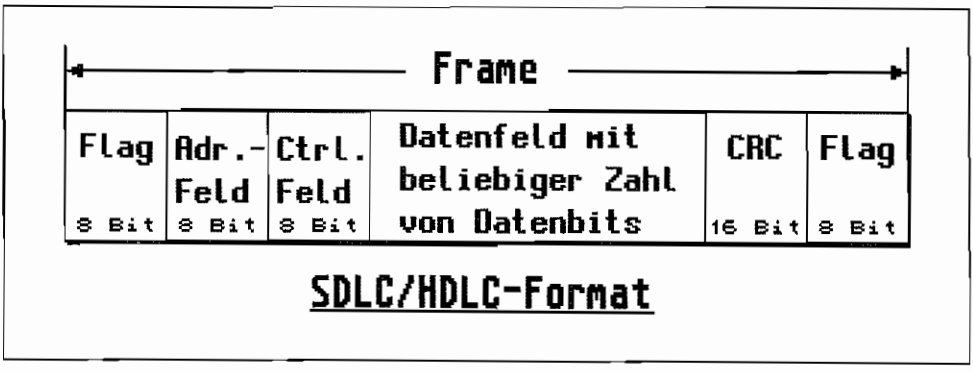


Abb. 7.3: Bitorientierte Steuerungsverfahren

Der Unterschied eines Steuer-/Meldeframes zu einem Informationsframe besteht lediglich darin, daß Steuer- und Meldeframes kein Datenfeld beinhalten.

Jeder Frame beginnt mit einem FLAG. Dafür wird die 8-Bitfolge $01111110_{bin} = 7E_{hex}$ verwendet. Der Empfänger "rastet" seine Synchronisierung auf diese Bitfolge ein. Da ja alle Bitmuster bei diesem Verfahren z. B. auch im Datenfeld verwendet werden dürfen, muß der Sender dafür sorgen, daß die FLAG-Sequenz nicht im eigentlichen Datenstrom auftaucht. Vom Sender wird daher immer nach fünf "1"-Bits automatisch ein "0"-Bit in den Datenstrom eingefügt (außer beim FLAG!). Der Empfänger muß dann zwischen dem Start-FLAG und dem End-FLAG eines Frames diese "0"-Bits wieder entfernen.

Bei SDLC/HDLC wird zur Blocksicherung das CRC-Verfahren angewendet und die "berechnete" CRC-Bitfolge in Form von 16-Bits unmittelbar vor dem End-FLAG mitübertragen.

Mit SDLC/HDLC-Steuerungsverfahren ist Duplexbetrieb möglich. Informationsframes enthalten im Control-Feld Zählerwerte, mit denen die Gegenstelle A ihrem Gegenüber B mitteilt, welche Nummer der gerade übermittelte Frame besitzt. Des weiteren wird von Station A durch eine Framenummer im Control-Feld der Station B übermittelt, bis zu welcher Nummer die Station A die von B gesendeten Frames bereits empfangen und verarbeitet hat. Das bedeutet, daß mit dem Aussenden eines Informationsframes gleichzeitig eine Bestätigung der bereits von der Gegenstelle empfangenen Frames erfolgen kann.

Das Adreßfeld bei SDLC/HDLC kann benutzt werden, um in Mehrpunktverbindungen, die gewünschte Zielstation zu adressieren. Aber auch bei Punkt-zu-Punkt-Verbindungen werden Adressen verwendet. Jede Station versendet Informations- und Steuerframes mit der Zieladresse im Adreßfeld. Meldungen/Quittungen werden mit der eigenen (Absende-)Adresse versehen.

Codierungsverfahren für Datensignale

Wenn Datensignale nicht mittels MODEM-Verfahren auf Übertragungswegen übermittelt werden sollen, sondern im sogenannten Basisbandverfahren (Basisband deshalb, weil keine Umsetzung der Datensignale in einen anderen Frequenzbereich erfolgt!) auf die Leitung gelangen, werden unterschiedliche Methoden verwendet, um das Signal z. B. möglichst frei von Gleichstromkomponenten (sonst bekommt man die Signale so schlecht durch Übertrager!) zu halten. Einige dieser Verfahren codieren die Daten so um, daß sich möglichst viele Zustandswechsel ergeben. Damit läßt sich dann einfacher aus diesem Datenstrom wieder die Taktinformation zurückgewinnen!

Die Umcodierung kann sowohl in Software realisiert als auch (wie beim SCC) durch Hardware erledigt werden.

NRZ-Verfahren

*N*on *R*eturn to *Z*ero. Hier erfolgt eigentlich keine weitere Umcodierung.

Alle Daten werden durch ihre entsprechenden log. Zustände dargestellt. Um einen möglichst geringen Gleichanteil zu haben, wird z. B. die log. 1 durch eine positive Spannung und die log. 0 durch eine negative Spannung bezogen auf Masse dargestellt.

Bei längeren Null- oder Eins-Folgen ist eine Taktrückgewinnung aus dem Datenstrom kaum sicher möglich.

NRZI-Verfahren

*N*on *R*eturn to *Z*ero - *I*nverted. Bei jedem Null-Bit wird ein Zustandswechsel durchgeführt! Damit ist auch gewährleistet, das bei längeren Null-Folgen Zustandswechsel auftauchen. Normalerweise kann auch aus diesem umcodierten Signal nicht besonders effizient die Taktinformation zurückgewonnen werden (z. B. bei längeren Eins-Folgen!).

Da aber bei SDLC/HDLC ja durch die "Zero bit insertion" nach spätestens fünf Eins-Bits eine Null eingefügt wird, hat man doch noch eine einigermaßen "wechselhafte" Bitfolge vor sich.

Biphase Mark-Verfahren oder FM1-Verfahren

Bei dieser Umcodierung wird an jeder Datenbitgrenze ein Zustandswechsel durchgeführt. Hat das darzustellende Datenbit den Wert Eins, so wird in Bitmitte ein zusätzlicher Zustandswechsel durchgeführt. Dieses Umcodierungsverfahren stellt ausreichend Zustandswechsel zur Taktrückgewinnung bereit.

Biphase Space-Verfahren oder FM0-Verfahren

Hierbei handelt es sich lediglich um eine Umkehrung des vorgenannten Verfahrens. Also bei Null-Bits wird in der Bitmitte ein zusätzlicher Zustandswechsel ausgeführt.

Manchester-Codierung

Hierbei wird in Bitmitte immer ein Zustandswechsel durchgeführt. Die Richtung des Zustandswechsels gibt dabei den Datenbit-Wert an!

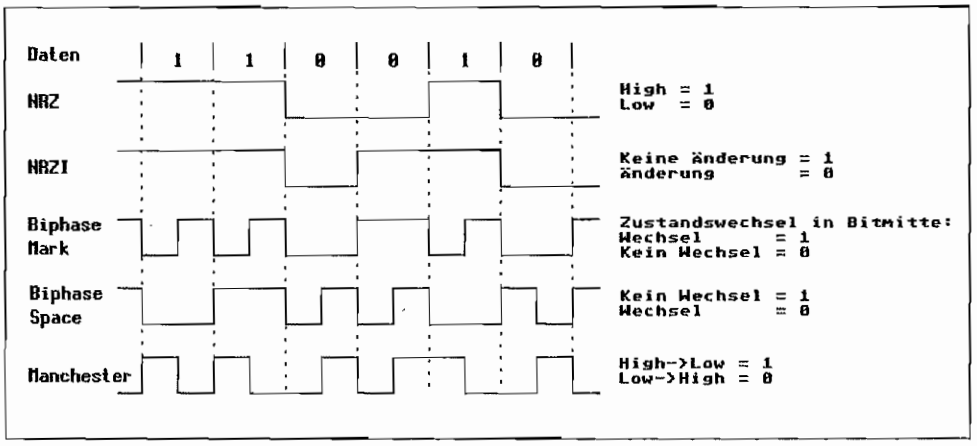


Abb. 7.4: Daten-Umcodierungsverfahren

Der SCC

Um all die geschilderten Gleichlaufverfahren (Asynchron/Synchronbetrieb), Umcodierungs- und DÜ-Steuerungsverfahren abwickeln zu können, besitzt der SCC einen komplexen Aufbau.

Zwei voneinander unabhängige Kommunikationskanäle existieren im SCC. Im TT kann dabei der SCC-Kanal A zwischen einem 8pol. Miniatur-DIN-Netzwerkanschluß mit elektrischen Werten entsprechend RS422 (als LAN bezeichnet) und einem 9pol. SubD-Anschluß (als SERIAL 2 bezeichnet) mit elektrischen Werten nach RS232C umgeschaltet werden.

Die Umschaltung erfolgt durch den IOA7-Port (Pin 14) des Soundchips (Low = LAN / High = SERIAL 2). Als Taktquelle für Kanal A des SCC kann sowohl der PCLK mit 8 MHz verwendet werden (maximale Datenrate 125 KBit/s) als auch ein 3,672 MHz-Takt von einem eigenen Oszillator (maximale Datenrate ca. 230 KBit/s). Außerdem läßt sich bei LAN-Betrieb ein ext. Takt aus dem LAN verwenden.

Der Kanal B kann im TT/MEGA STE als RS232C-Schnittstelle für unterschiedliche, max. Datenraten programmiert werden. Wird der PCLK-Takt von 8 MHz als Ausgangstakt verwendet, können bis zu 125 KBit/s erreicht werden. Wenn als Taktquelle der 2,4576 MHz-Takt der MFPs benutzt wird, geht die maximale Datenrate bis maximal 153600 Bit/s. Auch der Ausgang des Timers C des ST-MFP kann als Taktquelle verwendet werden. Dann reduziert sich die maximale Datenrate auf 19200 Bit/s.

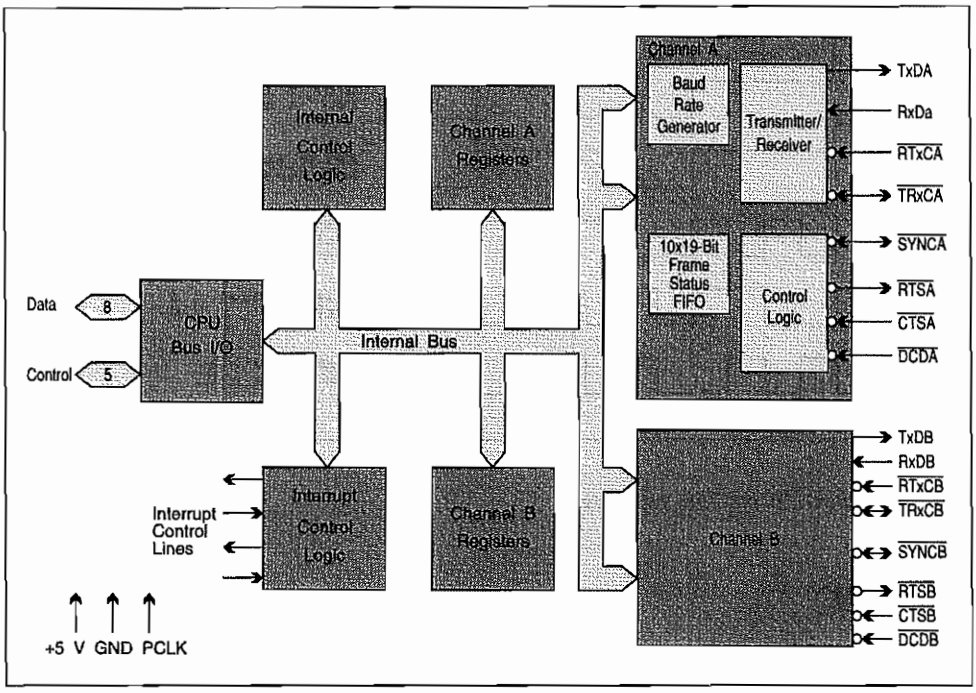


Abb. 7.5: Block-Diagramm des SCC

Die zuvor genannten Datenraten gelten für eine Betriebsweise mit einer Art Vorteiler durch 16, um bei Start/Stop-Betrieb eine stabile Abtastung in Bitmitte erreichen zu können.

Im Synchronbetrieb, wenn ein zu den Daten passendes Taktsignal vorliegt, lassen sich natürlich Geschwindigkeiten erreichen, die dann um den Faktor 16 (Vorteiler nicht erforderlich!) höher liegen!

Ein Blick ins Innenleben des SCC

Um die verschiedenen Gleichlauf- und DÜ-Steuerungsverfahren unterstützen zu können, ist der SCC "ein wenig" komplexer aufgebaut als z. B. ein MFP oder 6850! Auf alle Details des SCC hier einzugehen würde den Rahmen des Buches sprengen (allein das kurzgefaßte "Datenblatt" zum Z85C30 umfaßt schon mehr als 130 Seiten komprimierte Informationen zu dem Baustein!).

Der Empfangsteil des SCC

Deshalb hier erstmal eine Kurzbeschreibung des Empfängerteils im SCC (jeder Kanal besitzt natürlich einen eigenen Empfangsteil!).

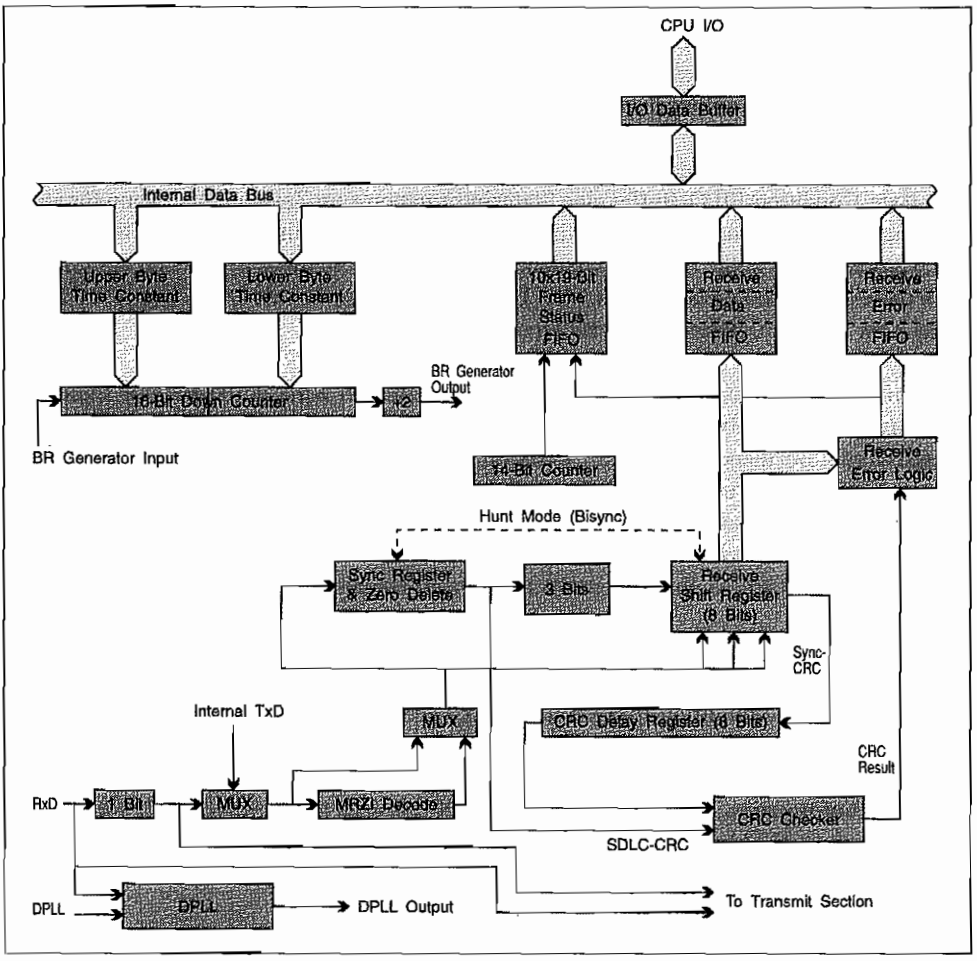


Abb. 7.6: Block-Diagramm des SCC-Empfängers

Die Daten von der Empfangsdatenleitung gelangen nach einer 1 Bit-Verzögerungsstufe (wird im SDLC-Loop-Mode benutzt; dazu später mehr), abhängig von der eingestellten Betriebsart,

in die NRZI-Decoderlogik, das SYNC-Empfangsregister oder das Empfangsschieberegister. Das SYNC-Empfangsregister hat im SDLC/HDLC-Betrieb zudem die Aufgabe, das vom Sender nach jeweils fünf Eins-Bits eingefügte Null-Bit wieder herauszunehmen.

Sobald ein Zeichen im Empfangsschieberegister "zusammengebaut" ist, gelangt es in ein 3-Byte-Empfangs-FIFO-Buffer. Damit lassen sich also bis zu drei Bytes zwischenspeichern, um der CPU oder dem DMA-Baustein etwas "Luft" für das Auslesen der Empfangsdaten zu lassen. Jedes Auslesen liefert immer das "oberste" Zeichen des FIFO-Buffers und bewirkt nach dem Lesezugriff ein Nachrücken der "darunterliegenden" Zeichen!

Jedes Zeichen, welches in den 3-Byte-FIFO übertragen wird, enthält in einem parallel geführten 3-Byte-Condition-FIFO-Buffer eine Statusinformation zum Zeichen. Also kann für jedes Empfangszeichen die zugehörige Statusinformation aus dem Condition-FIFO-Buffer entnommen werden! Dabei sollte beachtet werden, immer erst den Status zum Zeichen auszulesen, bevor das eigentliche Zeichen geholt wird. Umgekehrt bekommt man nämlich sonst evtl. den Status vom nächsten, bereits "aufgerückten" Zeichen!

Wenn mit "Special Receive Condition"-Interrupts gearbeitet wird, geschieht eine Interruptanforderung für das "oberste" Zeichen des FIFO-Buffers erst dann, wenn das Zeichen ausgelesen wurde! Bei manchen Special-Condition-Interrupt-Betriebsweisen wird zusätzlich der gesamte FIFO-Buffer "eingefroren". So kann dann in "aller Ruhe" die Special Condition ausgewertet werden. Erst nach Ausführung eines entsprechenden ERROR-RESET-Kommandos wird der FIFO-Buffer wieder "aufgetaut". Dieses Verhalten erscheint bei DMA-Betrieb recht sinnvoll, weil die CPU ja ohne Interrupts nichts vom Datentransfer mitbekommen würde.

Zusätzlich zum 3-Byte-Condition- und Daten-FIFO-Buffer wird für SDLC/HDLC-Betrieb noch ein 10fach-Frame-Status-FIFO-Buffer mit 19 Bit Breite verwendet. Hierin werden, ähnlich dem 3 Byte-Condition-FIFO-Buffer für Zeichen, die Statusinformationen der letzten zehn Frames aufbewahrt. Ein 14-Bit-Zähler zählt zusätzlich die Datenbytes je Frame (bei 14 Bit also maximal 16 KByte pro Frame!) und legt diese Zählerstände zusammen mit einer 5-Bit-Statusinformation in diesem Frame-Status-FIFO-Buffer ab. Werden so z. B. Frames per DMA-Betrieb im RAM abgelegt, kann anhand der Zählerstände nachträglich jeder einzelne Framebeginn im RAM ermittelt und "defekte" Frames ausge"mapped" werden!

Ebenfalls im Empfangsteil untergebracht ist die CRC-Prüflogik (ein 16-Bit-Schieberegister mit entsprechenden Rückkopplungspunkten). Je nach Betriebsart laufen die Daten zur CRC-"Berechnung" über das SYNC-Register (SDLC-CRC) oder das Empfangsschieberegister und eine 8-Bit-Pufferstufe (SYNC-CRC) in die Prüflogik ein.

Das Ergebnis der Blocksicherung wird ebenfalls durch die "Receive-Error-Logic" ausgewertet und als Statusinformation zugänglich gemacht.

Im Empfangsteil findet sich auch die DPLL-Stufe, die aus den Flanken im Empfangsdatensignal einen Takt abzuleiten versucht. Dabei kann der Quelltakt für die DPLL-Stufe sowohl vom eingebauten Baudraten-Generator (durch einen 16-Bit-Wert in der Ausgangsfrequenz programmierbar!) als auch vom RTxC-Eingang gespeist werden (Takt z. B. durch externen Oszillator erzeugt).

Der Sendeteil des SCC

übernimmt die Umsetzung der parallel eingegebenen Daten in einen seriellen Datenstrom. Dazu verfügt der Sender über ein 8-Bit-Sendedatenregister und ein Sendeschieberegister. Wenn synchron zeichenorientiert gearbeitet wird (z. B. Bisync- oder Monosync-Verfahren), werden die entsprechenden SYNC-Register bei der Bedienung des Sendeschieberegisters natürlich mit berücksichtigt.

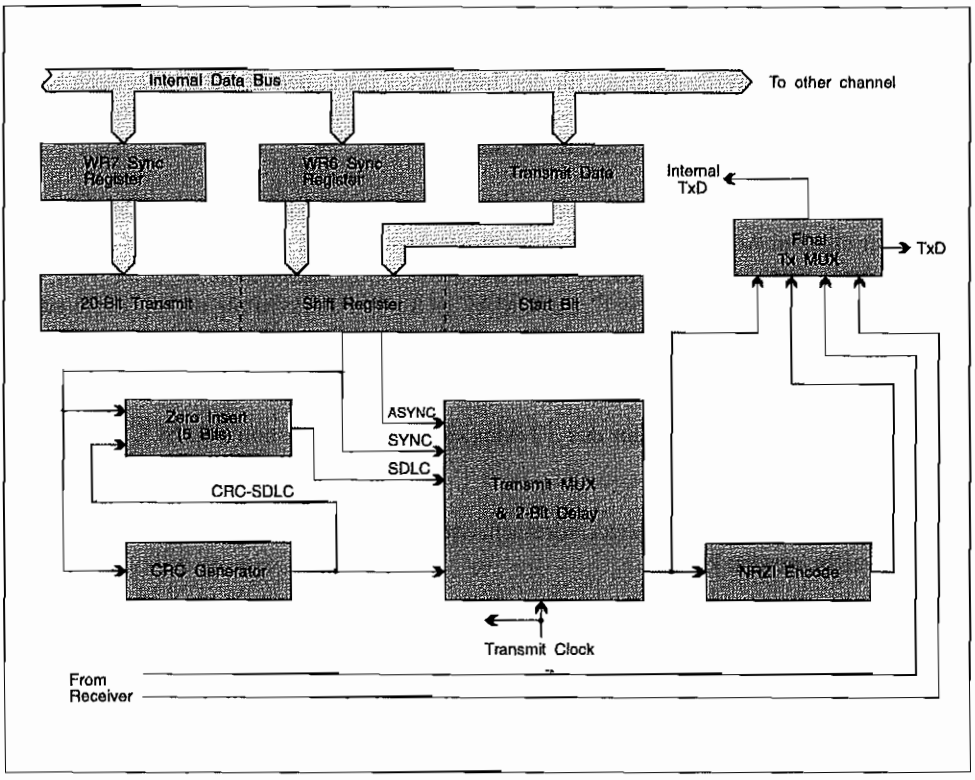


Abb. 7.7: Blockdiagramm des SCC-Senders

Immer wenn ein Zeichen aus dem Sendedatenregister ins Sendeschieberegister übergeben wurde, wird durch ein entsprechendes Statusbit (oder einen Interrupt) signalisiert, daß das Sendedatenregister nachgeladen werden kann. Auf diese Weise erfolgt im Prinzip die Pufferung eines ganzen Zeichens, so daß für die CPU genug Zeit zum "Nachladen" bleibt.

Je nach Steuerungsverfahren gelangen die Sendedaten noch evtl. über eine Stufe für die "Zero Bit Insertion" (bei SDLC/HDLC-Betrieb) und eine Umcodierstufe. Außerdem wird bei Verwendung von Blocksicherungsverfahren nach der CRC-Methode der Datenstrom noch durch die CRC-Stufe "geschleust", um die CRC-Prüfsumme zu "berechnen".

Die Register des SCC

Für die Steuerung des Z85C30 und die Abfrage von Statusinformationen sind eine Menge von Registern vorgesehen. Zur Steuerung existieren für jeden Kommunikationskanal 14 Write-Register (können nur beschrieben werden). Für Statusinformationen und zum Auslesen der Daten existieren je Kanal noch einmal zehn Read-Register! Außerdem haben beide Kanäle noch gemeinsam ein Write-Register für den Interrupt-Vektor, den der SCC erzeugen kann, und ein Write-Register für die Interrupt-Steuerung. Ähnliches gilt für zwei Read-Register, die von beiden Kanälen gemeinsam benutzt werden.

Im TT (MEGA STE) findet man für jeden der beiden SCC-Kanäle nun nicht jedes Register unter einer eigenen Adresse. Vielmehr werden zur Ansteuerung des SCC immer die beiden DMA-Bausteine "zwischengeschaltet". Der DCU übernimmt die Umsetzung des 32-Bit-Datenbusses auf den 8-Bit-Bus des SCC. Der DMAC ist für die Adressierung und die Erzeugung/Auswertung der Control-Signale zum SCC zuständig. Beide Bausteine "spielen zusammen", wenn der SCC im DMA-Betrieb verwendet wird.

An welcher Stelle im Adreßraum des TT/MegaSTE findet man nun den SCC bzw. seine Register?

Für jeden SCC-Kanal existieren zwei Adressen, unter denen alle Register des jeweiligen SCC-Kanals erreicht werden können.

Adr. $(FF)FF 8C81	**Control-Reg. SCC**	**Kanal A**	**Label: "sccctl_a"**
Adr. $(FF)FF 8C83	**Data-Reg. SCC**	**Kanal A**	**Label: "sccdat_a"**
Adr. $(FF)FF 8C85	**Control-Reg. SCC**	**Kanal B**	**Label: "sccctl_b"**
Adr. $(FF)FF 8C87	**Data-Reg. SCC**	**Kanal A**	**Label: "sccdat_b"**

Die Data-Register erlauben mit einem Schreibzugriff direktes Einschreiben eines Bytes in den Sendebuffer (Write Register 8). Bei einem Lesezugriff erhält man das Byte aus dem Emp-

fangsbuffer (Read Register 8). Also ist ein Zugriff auf die "zeitkritischen Daten" ohne große Umwege möglich. Die Control-Register werden auf folgende Weise in zwei Stufen "bedient":

- Der erste Schreibzugriff auf die Control-Register "landet" in Write Register 0 des entsprechenden SCC-Kanals und wird zur Einstellung der gewünschten Registernummer verwendet.

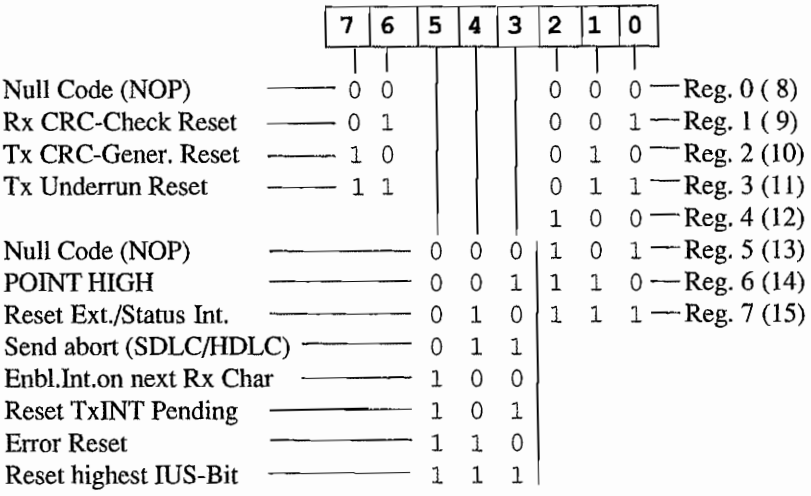
- Der nächste Zugriff (Schreiben oder Lesen) auf das Control-Register erreicht dann das im ersten Zugriff ausgewählte Register!

Write-Register

Write-Register 0
(Kommando-Register)

Mit diesem Register wird festgelegt, welches Register beim nächsten Zugriff auf die gleiche Adresse angesprochen werden soll. Außerdem können durch Schreibzugriffe mit entsprechenden Bitkombinationen z. B. die CRC-Stufen im Sende- und Empfangsteil zurückgesetzt werden, Fehlersignalisierungen und "anhängige" Interrupts gelöscht werden.

Die nachfolgende Abbildung zeigt die Bitbelegung des Registers.



Werte in Klammern gelten dann, wenn das POINT HIGH-Kommando in den Bits 5..3 eingestellt ist!

Null Code

Diese Einstellung bewirkt keinerlei Änderung im SCC und wird verwendet, wenn bei einem Zugriff auf das Write-Register 0 lediglich eingestellt werden soll, welches Write-Register beim nächsten Zugriff adressiert werden soll.

Rx-CRC Check Reset

Dieses Kommando wird verwendet, um den CRC-Prüfer im Empfänger zurückzusetzen. Der Programmierer braucht dies nur dann "von Hand" zu machen, wenn im zeichenorientierten Synchronbetrieb (Bisync oder Monosync-Betrieb) zwischen empfangenen Textblöcken der Empfänger nicht sowieso wieder auf den "Hunt mode" (Suchen nach SYNC-Zeichen) eingestellt wurde.

Im SDLC/HDLC-Modus wird der CRC-Prüfer bei Erkennen eines Flags automatisch zurückgesetzt, ebenso bei "Abschalten" (disablen) des Empfängers.

Tx CRC-Generator Reset

Durch dieses Kommando wird im Sender der CRC-Generator initialisiert. Das passiert üblicherweise während der Initialisierung des SCC oder nachdem die CRC-Daten für einen Datenblock ausgesendet wurden.

Tx Underrun Reset/EOM Latch Reset

Mit diesem Kommando wird das Aussenden der CRC-Information zum Ende eines Datenblocks (EOM = End of Message) gesteuert.

Wenn durch dieses Kommando das EOM-Flag zurückgesetzt wurde, wird automatisch bei Auftreten der Tx Underrun-Bedingung (Keine Daten mehr zum Aussenden da, weil anscheinend alles ausgesendet wurde!) die CRC-Prüfinformation ausgesendet! Mit dem Aussenden der CRC-Information wird dann das EOM-Flag gesetzt!

Dieses Kommando kann jederzeit während des Aussendens eines Blocks gegeben werden. Bei ausgeschaltetem Sender erfolgt durch dieses Kommando kein Rücksetzen des EOM-Flags.

POINT HIGH

Wenn dieses Kommando gesetzt wird, wirkt sich das in einer Addition von 8 zum Wert der Registernummer in den Bits 0..2 aus. Damit ist dann auch eine Adressierung der Register 8..15 möglich.

Reset External /Status Interrupts

Wenn eine Steuerleitung (CTS, DCD) sich geändert hat, wird dies in einem entsprechenden Statusbit "gemerkt". Gleiches gilt für spezielle Zustände wie z. B. "Break detect" (im Asynchronverfahren wird eine Dauer-Null-Bitfolge empfangen).

Im SDLC/HDLC-Modus ist eine Sendeabbruch-Folge mit mehr als sieben Eins-Bits festgestellt worden). Durch dieses Kommando lassen sich die Statusbits wieder zurücksetzen, und Interrupts, die auf diesen Statusbits beruhen, können damit erneut ausgelöst werden!

Send abort (SDLC/HDLC)

Durch dieses Kommando wird im SDLC/HDLC-Modus ein Sendeabbruch signalisiert. Der Sender gibt nach Empfang dieses Kommandos zunächst noch den Inhalt des Sendepuffers und danach 8..13 Eins-Bits aus. Anschließend wird das EOM-Flag gesetzt.

Enable Int. on next Rx Character

Empfänger-Interrupts können durch drei voreinstellbare Bedingungen ausgelöst werden. Zum ersten kann jedes Zeichen, welches in den Empfangs-FIFO-Buffer gelangt, einen Interrupt auslösen. Zum zweiten läßt sich die Interruptsignalisierung für den Empfänger so programmieren, daß nur bei speziellen Empfänger-Zuständen (z. B. Empfängerüberlauf, Paritätsfehler) "interrupted" wird. Und zum dritten kann immer dann ein Interrupt signalisiert werden, wenn das erste Zeichen eines Textblocks (nicht SYNC-Zeichen!) in den Empfangs-FIFO-Buffer gelangt. Damit diese dritte Interrupt-Betriebsart jeweils nach Empfang eines Textblocks erneut aktiviert werden kann, wird dieses Kommando verwendet.

Reset TxINT pending

Dieses Kommando wird benutzt, um zu verhindern, daß ein weiterer Sender-Interrupt ausgelöst wird,

obwohl alle zu sendenden Zeichen ausgegeben wurden und der Sendebuffer "leerläuft".

Es sorgt dafür, daß erst wieder ein Senderinterrupt auftreten kann, wenn das nächste zu sendende Zeichen (eines neuen Textblocks!) in das Sendeschieberegister übergeben wurde oder wenn eventuelle CRC-Bytes komplett ausgesendet wurden.

Error Reset

Bei bestimmten Betriebsarten werden die Statusinformationen für die Empfangszeichen "eingefroren".

Mit diesem Kommando können diese Statusbits zurückgesetzt und der eingefrorene Zustand aufgehoben werden.

Reset Highest IUS-Bit

IUS steht für "Interrupt under Service". Mit diesem Kommando läßt sich gezielt der im Moment am höchsten priorisierte Interrupt "abschalten". Das sollte immer am Ende einer Interrupt-Service-Routine geschehen, um dann den "weniger wichtigen" Interrupts auch die Möglichkeit zur Bearbeitung zu geben.

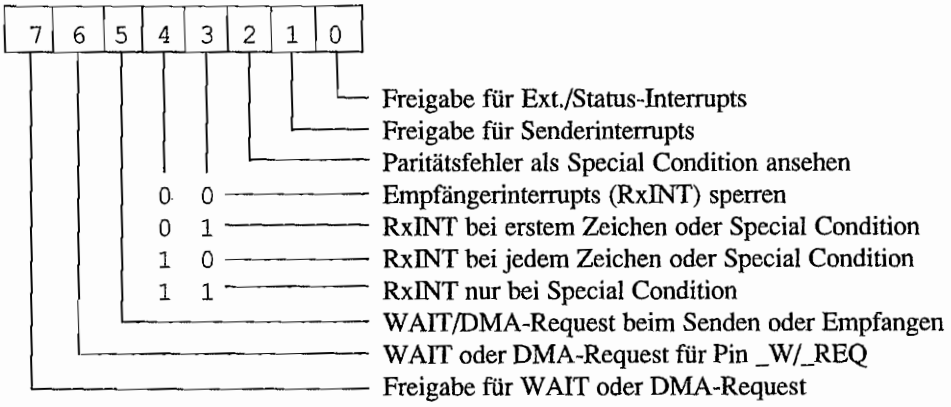
Die Interrupt-Prioritäten sind beim SCC wie folgt gesetzt:

Empfänger Interrupt Kanal A	High
Sender Interrupt Kanal A	↓
Ext./Status-Int. Kanal A	↓
Empfänger Interrupt Kanal B	↓
Sender Interrupt Kanal B	↓
Ext./Status-Int. Kanal B	Low

Write-Register 1
(Sender/Empfänger Interrupts und Modus-Register für Datentransfer)

Dies ist das Steuerregister für verschiedene Interrupts des SCC. Außerdem kann hier das Handshaking zwischen SCC und DMA-Baustein eingestellt werden.

Der SCC besitzt für diese Zwecke je Kanal einen programmierbaren Anschluß.



Freigabe für Ext./Status-Interrupts Dieses Bit ist der "Hauptschalter" für Ext./Status-Interrupts. Mit Setzen dieses Bits werden alle Interruptquellen, deren Freigabe-Bits im Write Register 15 gesetzt sind, "scharfgeschaltet".

Freigabe für Senderinterrupts Bei gesetztem Bit wird durch den Sender immer dann ein Interrupt ausgelöst, wenn der Sendepuffer leer wird.

Paritätsfehler als Special Condition ansehen
Bei gesetztem Bit wird ein Paritätsfehler bei einem empfangenen Zeichen als Special Condition angesehen und kann einen entsprechenden Interrupt auslösen.

Empfängerinterrupts (RxINT) sperren
Wenn der SCC im Polling-Betrieb benutzt wird (die CPU schaut immer mal nach, wie der SCC-Status ist. Dazu werden z. B. die Statusbits im Read Register 0 benutzt), können durch diese Einstellung die Empfängerinterrupts gesperrt werden.

RxINT bei erstem Zeichen oder Special Condition
Ein Interrupt wird ausgelöst, sobald ein erstes Zeichen im Empfangs-FIFO-Buffer vorliegt oder wenn ein Zeichen ausgelesen wird, bei dem eine Special Condition aufgetreten ist. Eine Special Condition ist z. B. ein Paritätsfehler (kann separat als Special Condition programmiert werden!), ein Überlauf des Empfangsregisters, ein

CRC-Fehler oder ein Framing Error (= kein Stopbit bei Asynchronbetrieb erkannt). Der Empfangs-FIFO-Buffer bleibt so lange "eingefroren", bis ein Error Reset über das Write Register 0 ausgelöst wird.

RxINT bei jedem Zeichen oder Special Condition

Jedes empfangene Zeichen löst einen Interrupt aus. Außerdem wird ebenfalls ein Interrupt ausgelöst, wenn eine Special Condition auftrat. Jedoch wird der Empfangs-FIFO-Buffer nicht "eingefroren", bis ein Error Reset ausgelöst wird.

RxINT nur bei Special Condition

Bei dieser Einstellung wird nur ein Interrupt ausgelöst, wenn beim Zeichenempfang eine Special Condition auftrat. Das zugehörige Zeichen bleibt wieder bis zu einem Error Reset im Empfangs-FIFO-Buffer "eingefroren".

Für beide SCC-Kanäle gibt es einen Anschluß, der für Betrieb des SCC mit z. B. DMA-Bausteinen benutzt werden kann. Im TT ist dieser Anschluß aber nur für den SCC-Kanal A benutzt. Der _W/_REQ-Anschluß kann dabei auf verschiedene Betriebsmodi eingestellt werden.

- *Wait on Transmit.* (Bits 7..5 = "100")
  Bei dieser Einstellung liefert der _W/_REQ-Pin so lange ein Low, wie der Sendepuffer noch gefüllt ist. Sobald der Sendepuffer ein neues Zeichen aufnehmen kann, geht der _W/_REQ-Anschluß in einen hochohmigen Zustand.

- *Wait on Receive.* (Bits 7..5 = "101")
  Bei dieser Einstellung liefert der _W/_REQ-Pin ein "Wartesignal" (Low-aktiv), welches die CPU bei einem Lesezugriff auf einen leeren Empfangs-FIFO-Buffer zum "Warten" auffordert, bis ein Zeichen zum Auslesen vorliegt. Sobald ein Zeichen vorliegt, geht _W/_REQ in den hochohmigen Zustand.

- *DMA-Request on Transmit.* (Bits 7..5 = "110")
  Mit diesem Low-aktiven Signal an _W/_REQ bekommt der DMA-Baustein mitgeteilt, wenn der Sendepuffer leer ist und ein neues Zeichen eingeschrieben werden soll. Erst wenn ein Zeichen "nachgefüllt" wurde, geht _W/_REQ wieder auf *High* (wird nicht hochohmig!).

- *DMA-Request on Receive.* (Bits 7..5 = "111")
  Auch hier fordert der SCC mit einer fallenden Flanke an _W/_REQ einen DMA-Baustein

zur Aktion auf. Nur daß jetzt signalisiert wird, daß ein Zeichen aus dem Empfangs-FIFO-Buffer abgeholt werden kann. Sollte der SCC in einer Betriebsart für den Empfangsinterrupt verwendet werden, die den FIFO-Empfangsbuffer bei Auftreten einer Special Condition "einfriert", so werden natürlich nicht laufend _W/_REQ-Impulse erzeugt (schließlich bleibt ja das Zeichen, bei dem diese Special Condition auftrat, im FIFO-Buffer eingefroren!). Sobald das Zeichen einmal ausgelesen wurde, geht _W/_REQ wieder auf High und reagiert erst wieder "normal", sobald mit dem Error Reset-Kommando die Special Condition beseitigt wurde.

WAIT/DMA-Request beim Senden oder Empfangen

Bei gesetztem Bit arbeitet der _W/_REQ-Anschluß für den Empfangsteil. Bei gelöschtem Bit wird der _W/_REQ-Anschluß dagegen vom Sendeteil des SCC gesteuert.

WAIT oder DMA-Request für Pin _W/_REQ

Bit = 0: _W/_REQ-Pin in WAIT-Betriebsweise.
Bit = 1: _W/_REQ-Pin in REQUEST-Betriebsweise

Freigabe für WAIT oder DMA-Request

Bei gesetztem Bit ist WAIT/REQUEST aktiviert. Bei gelöschtem Bit ist weder WAIT- noch REQUEST-Betriebsart freigegeben.

Der SCC besitzt je Kanal noch einen weiteren Pin, der als REQUEST-Anschluß für DMA-Betrieb verwendet werden kann. Auch dieser Pin ist als Multifunktionsanschluß ausgelegt. Er läßt sich sowohl für DMA-REQUEST zur Bedienung des Sendeteils einstellen als auch als normaler Schnittstellenanschluß für das DTR-Signal (Data Terminal Ready an MODEM) benutzen. Atari verwendet den DTR/REQ-Pin aber für beide SCC-Kanäle als DTR-Anschluß! Deshalb soll hier nicht weiter auf die REQUEST-Betriebsart mit diesem Anschluß eingegangen werden.

Write-Register 2
(Interrupt Vektor-Register)

Der SCC kann, ähnlich dem MFP, während eines Interrupt-Acknowledge-Zyklus eine Vektornummer ausgeben. Diese Vektornummer kann hier eingestellt werden. Dabei existiert für beide SCC-Kanäle nur ein Interrupt Vektor-Register, daß aber von "beiden Seiten" erreicht werden kann. Die Vektornummer kann abhängig von Statusinformationen (wie z. B. "Sendepuffer leer") geändert werden. Dabei läßt sich durch das "Status High/_Status Low"-Bit in Write-Register 9 einstellen, welche Bits der Vektornummer durch die Statusinformation

beeinflußt werden. Eine Änderung durch Statusinformationen betrifft entweder die Bits 1..3 ("Status High/_Status Low"-Bit in Write-Register 9 gelöscht) oder die Bits 4..6 ("Status High/_Status Low"-Bit in Write-Register 9 gesetzt) im Interrupt Vektor-Register.

Ob überhaupt Statusinformationen die Vektornummer beeinflussen können, wird durch das Bit "Vektor includes Status (VIS)" in Write-Register 9 eingestellt (Bit gesetzt: Beeinflussung möglich!).

Welche Beeinflussung die Vektornummer durch die Statusbits erfahren kann, zeigt die nachfolgende Tabelle.

Bit 3 4	2 5	1 6	Status High/_Status Low = 0 Status High/_Status Low = 1	Interrupt für
0	0	0	Sendepuffer leer	Kanal B
0	0	1	Externer/Status-Interrupt	"
0	1	0	Empfangszeichen verfügbar	"
0	1	1	Special Condition beim Empfang	"
1	0	0	Sendepuffer leer	Kanal A
1	0	1	Externer/Status-Interrupt	"
1	1	0	Empfangszeichen verfügbar	"
1	1	1	Special Condition beim Empfang	"

Im TT/MEGA STE verwendet ATARI eine Vektornummer von $60. Außerdem ist standardmäßig das "Status High/_Status Low"-Bit gelöscht, so daß die Interrupt Vektornummer durch Statusinformationen in den Bits 3..1 beeinflußt wird. Entsprechend findet man im RAM des TT/MEGA STE ab der Speicherstelle $180 die zugehörigen Interruptvektoren des SCC wieder. Also bei $180 den Zeiger auf die Routine für "Sendepuffer Kanal B leer", bei $188 die Adresse der Routine für "Externer/Status-Interrupt Kanal B" usw.

Write-Register 3
(Empfängersteuerung)

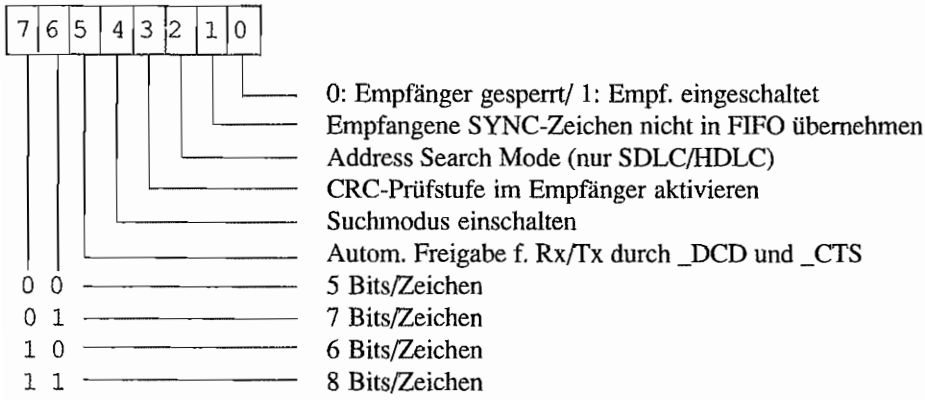
Über dieses Register können verschiedene Empfangsparameter eingestellt werden. Hier einige Erläuterungen zu den Einstellmöglichkeiten, die nicht unmittelbar ersichtlich sind:

Address Search Mode — Bei gesetztem Bit werden im SDLC/HDLC-Modus nur Datenblöcke mit Adressen ausgewertet, die mit der Adresse im Write-Register 6 übereinstimmen. Dabei kann allerdings durch das Bit 1 im Write-Register 3 (= SYNC-Zeichen nicht in FIFO übernehmen) noch bestimmt werden, ob die Empfangsadresse

exakt mit der Adresse übereinstimmen muß (dann muß Bit 1 gelöscht sein). Ist im Address Search Mode das Bit 1 gesetzt, müssen nur die vier höchstwertigen Bits im Adreßfeld des Empfangsblocks mit dem Wert im Write-Register 6 übereinstimmen.

Suchmodus einschalten

Bei gesetztem Bit geht der SCC-Empfangsteil auf "die Suche" nach SYNC-Zeichen (im synchronen, zeichenorientierten Modus) oder nach Flags (im SDLC/HDLC-Modus). Wenn die "Suche" erfolgreich war, wird in Read-Register 0 das Bit 4 (SYNC/HUNT-BIT) gesetzt und evtl. ein Interrupt ausgelöst.



Write-Register 4
(Moduseinstellungen für Sender/Empfänger)

In diesem Register lassen sich neben Sendereinstellungen auch einige Empfängerparameter beeinflussen (Registerbelegung siehe folgende Seite).

Synchronbetrieb einschalten

Wenn Synchronbetrieb gewählt ist, werden auch die Einstellungen in den Bits 5..4 berücksichtigt. Außerdem ist damit automatisch "Takt = 1 x Datenrate" eingestellt (die Einstellungen in Bits 7..6 bleiben dann unberücksichtigt)!

MONOSYNC-/BISYNC-Betrieb

Wenn MONOSYNC-Betrieb ausgewählt ist, wird die Zeichensynchronisierung auf das Zeichen im Write-Register 7 eingerastet.

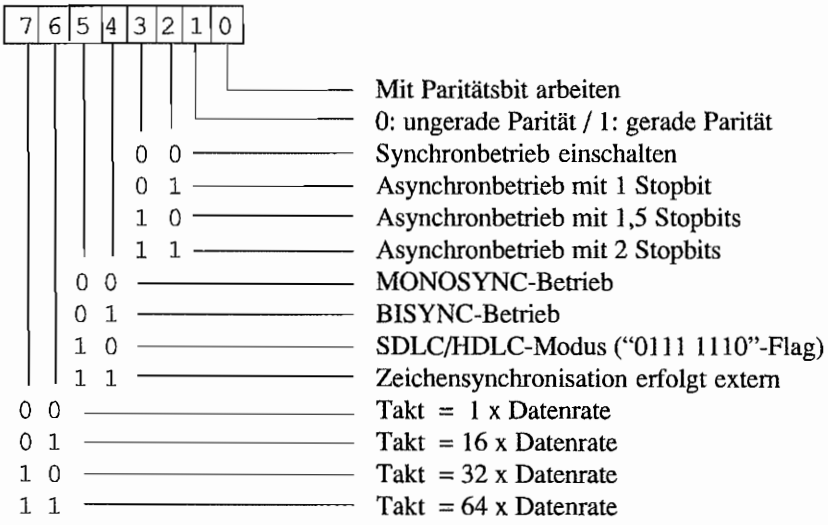
Bei BISYNC-Betrieb wird für die Zeichensynchronisierung das Bitmuster aus Write-Register 7 und 6 benutzt.

SDLC/HDLC-Modus

Für den SDLC/HDLC-Modus ist in Write Register 7 das Bitmuster für das Flag ($7E_{hex}$) einzutragen. Außerdem muß für das Blocksicherungsverfahren mittels CRC das zu verwendende Prüfpolynom über das Write Register 5 eingestellt sein.

Zeichensynchronisation extern

Beim SCC kann die Zeichensynchronisation im Synchronmodus auch extern über den _SYNC-Anschluß erfolgen. Im TT/MEGA STE wird dieser Pin jedoch bei beiden SCC-Kanälen für die Abfrage des "Data Set Ready"-Signals benutzt.



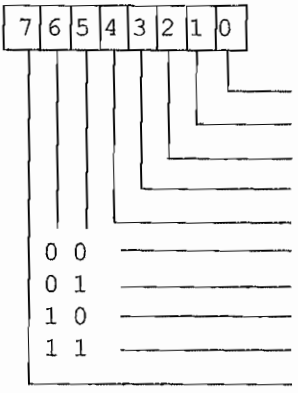
Write-Register 5
(Sendersteuerung)

Bis auf das Bit 2 dieses Registers beeinflussen alle Bits das Verhalten des SCC-Sendeteils.

Freigabe für Sender-CRC-Generator

Bei gesetztem Bit durchlaufen die zu sendenden Zeichen den CRC-Generator und erzeugen so eine Prüfbitfolge, die am

Ende des Blocks mitgesendet wird. Das Aussenden der CRC-Bits erfolgt jedoch erst dann, wenn keine Daten mehr zu senden sind. Der SCC erkennt das daran, daß das Senderegister nicht mehr "nachgefüllt" wird, also eine Tx Underrun-Bedingung entsteht!



Request to Send (RTS)

Dieses Bit steuert den _RTS-Pin des SCC. Bei gesetztem Bit geht der _RTS-Anschluß auf Low, bei gelöschtem Bit dann entsprechend auf H. . . (na, raten Sie selber mal! Ergebnis dann bitte selbst eintragen.)

Auswahl für CRC-Verfahren

Mit diesem Bit läßt sich programmieren, welches Generatorpolynom zur Bildung der Prüfbitfolge benutzt wird. Bei SDLC/HDLC wird das CRC-CCITT-Polynom benutzt (Bit=0). Wenn synchron, zeichenorientiert gearbeitet wird (z. B. MONOSYNC oder BISYNC), kommt meistens das CRC-16-Polynom zur Anwendung (Bit=1). Die Einstellungen gelten sowohl für den Sender als auch den Empfänger.

Freigabe für den Sender

Solange dieses Bit nicht gesetzt ist, werden keine Daten ausgesendet. Statt dessen gibt der Sender eine Dauerlage "1" aus. Wird nach dem Start einer Übertragung dieses Bit zurückgesetzt, werden die Daten im Senderegister aber noch komplett ausgesendet (gilt auch für SYNC-Zeichen). Wenn allerdings während der Übertragung von CRC-Zeichen der Sender abgeschaltet wird, gelangen statt dessen SYNC-Zeichen oder Flags an den Senderausgang.

BREAK senden (Dauerlage "0" ausgeben)
: Sobald das Bit gesetzt wird, gibt der Sender nur noch Dauerlage "0" aus. Wird das Bit wieder zurückgesetzt, geht's normal mit dem Aussenden der Sendeschieberegisterdaten weiter.

Data Terminal Ready (DTR)
: Wenn der SCC im DTR-Modus eingestellt ist (über Bit 2 im Write-Register 14 auswählbar), bestimmt dieses Bit den Zustand des _DTR/_REQ-Pins des SCC. Bei gelöschtem Bit ist der Anschluß dann High, bei gesetztem Bit entsprechend Low.

Der Inhalt dieses Registers kann auch ausgelesen werden. Dazu ist ein Lesezugriff auf das Read-Register 5 durchzuführen, wenn Bit 0 von Write-Register 15 und Bit 6 von Write-Regi-ster 7' gesetzt sind.

Write-Register 6
(SYNC-Zeichen / Adresse im SDLC/HDLC-Betrieb)

Im SDLC/HDLC-Modus enthält dieses Register die Adresse, auf die im "Address Search Mode" beim Frame-Empfang "gewartet" wird. Siehe auch bei Write-Register 3!

Im MONOSYNC-Modus wird in diesem Register das vom *Sender* verwendete SYNC-Zeichen eingestellt.

Bei BISYNC-Betrieb (Betrieb mit zwei SYNC-Zeichen oder einem 16 Bit-SYNC-Zei-chen, je nachdem, wie man die Sache betrachtet) enthält dieses Register die unteren acht Bits des SYNC-Zeichens.

Mit dem Write-Register 7 hat es etwas Besonderes auf sich. Es existiert nämlich im 85*C*30 zweimal. Gegenüber seinem "Vorgänger", dem 8530, hat die im TT/MEGA STE verwendete CMOS-Version des Chips einige zusätzliche Möglichkeiten in bezug auf den SDLC/HDLC-Betrieb "spendiert" bekommen (wie z. B. der 10fach FRAME-Status-FIFO-Buffer). Um an diese zusätzlichen Möglichkeiten und damit auch an das zusätzliche Write-Register 7' heranzukommen, muß SDLC/HDLC-Betrieb eingestellt und das Bit 0 im Write-Register 15 gesetzt sein. Aber nun erst einmal zur Funktion des "normalen" Write-Register 7.

Write-Register 7
(SYNC-Zeichen 2/ Flag im SDLC/HDLC-Betrieb)

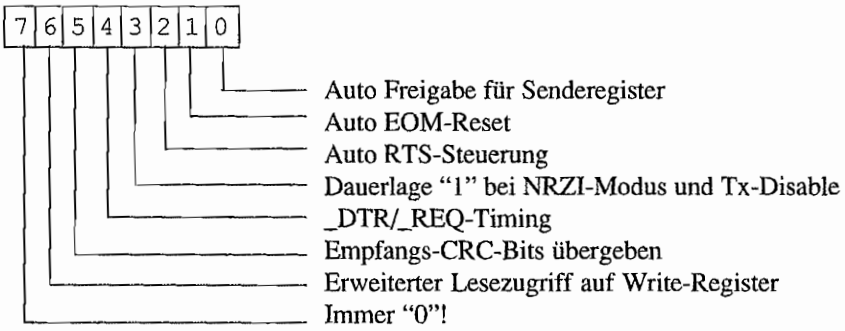
Im MONOSYNC-Modus wird mit dem Write-Register 7 das Empfangs-SYNC-Zeichen festgelegt.

Bei BISYNC-Betrieb enthält dieses Register den unteren Teil (oder das zweite SYNC-Zeichen, je nach Betrachtungsweise!) des SYNC-Zeichens.

Im SDLC/HDLC-Modus muß hier das Flag ($7E_{hex}$) eingetragen sein.

Write-Register 7'
(SDLC/HDLC-Optionsregister)

Nur wenn im SDLC/HDLC-Modus gearbeitet wird, kann auf dieses Register zugegriffen werden.



Auto Freigabe für Senderegister

Wenn dieses Bit gesetzt ist, braucht nicht erst das Aussenden des Flags abgewartet zu werden, bevor das erste Zeichen in das Senderegister gebracht werden kann. Sobald das erste Zeichen in das Senderegister gelangt, sendet der SCC das "Eröffnungs"-Flag und sendet dann die Daten hinterher.

Auto EOM-Reset

Wenn dieses Bit gesetzt ist, braucht das Rücksetzen des Tx Underrun/End Of Message-Flag nicht mehr zu Beginn eines Frames "von Hand" durchgeführt zu werden. Der SCC führt diesen RESET dann automatisch nach Aussenden des ersten Datenbytes durch.

Auto RTS-Steuerung

Bei gesetztem Bit wird das Deaktivieren der RTS-Leitung mit dem Aussenden des letzten Bits eines "Schluß"-Flags synchronisiert. Auch wenn vom Programm schon während des Sendens der CRC-Bits bereits RTS abgeschaltet wird, wird der SCC den RTS-Anschluß erst deaktivieren, wenn das letzte Bit des Schlußflags eines Frames gesendet ist!

Dauerlage "1" bei NRZI-Modus und Tx-Disable

Wenn der Sender abgeschaltet wird, während der SCC in NRZI-Kodierung betrieben wird, geht bei gesetztem Bit der Ausgangsanschluß auf High.

_DTR/_REQ-Timing

Der _DTR/_REQ-Anschluß des SCC kann auch für DMA-Requests programmiert werden. Allerdings nur für den Sendeteil des SCC. Wenn dieses Bit gesetzt ist, verhält sich das Request-Timing an diesem Anschluß genauso wie am _W/_REQ-Anschluß. Siehe dazu auch Erläuterungen zum Write-Register 1.

Empfangs-CRC-Bits übergeben

Bei gesetztem Bit wird die komplette CRC-Prüfbitfolge im Empfangsregister zur Verfügung gestellt. Sonst fehlen wegen einer 3 Bit-Verzögerungsstufe zwischen SYNC-Register und Empfangsschieberegister im Empfangsteil des SCC die letzten zwei Bits.

Erweiterter Lesezugriff auf Write-Register

Bei gesetztem Bit sind die eigentlich "unleserlichen" Write-Register 3..5 und 10 mit einem Lesezugriff erreichbar.

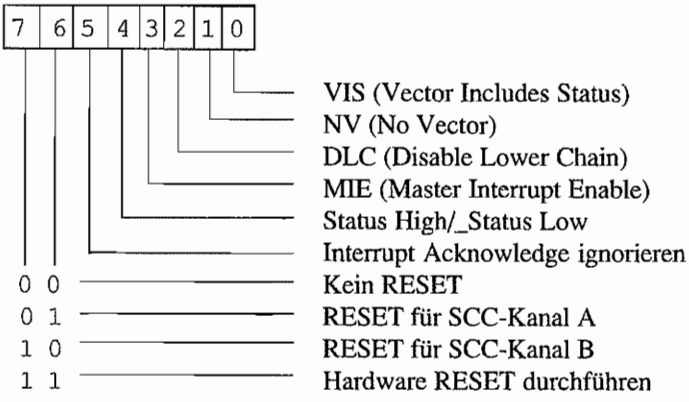
Allerdings taucht dann Write-Register 3 unter Read-Register 9 auf. Write-Register 10 kann aus Read-Register 11 ausgelesen werden. Write-Register 4 und 5 erscheinen unter Read-Register 4 und 5.

Write-Register 8
(Sendedatenregister)

Dieses Register ist das Sendedatenregister. Im TT/MEGA STE sind sowohl das Write-Register 8 als auch das Read-Register 8 (Empfangsdatenregister) unter der gleichen (eigenen) Adresse zu finden ("sccdat_a" an Adresse $(FF)FF 8C83 und "sccdat_b" an Adresse $(FF)FF 8C87).

Write-Register 9
(Master Interrupt Control)

Dieses Register existiert nicht für jeden SCC-Kanal separat, sondern kann von beiden Kanälen erreicht werden!



VIS (Vector Includes Status) Wie ja bereits beim Write-Register 2 erläutert, kann der Interruptvektor, den der SCC bei einem Interrupt-Acknowledge-Zyklus liefert, abhängig von Statusinformationen, die im Zusammenhang mit dem Interrupt stehen, geändert werden. Bei gesetztem Bit liefert der SCC also eine durch Statusinformationen beeinflußte Vektornummer.

NV (No Vector) Bei gesetztem Bit liefert der SCC keine Interrupt-Vektornummer während eines Interrupt-Acknowledge-Zyklus!

DLC (Disable Lower Chain) Wenn z. B. mehrere SCCs in einer Interrupt-Verkettung betrieben würden (wie das z. B. mit den beiden MFPs im TT geschieht), könnten durch Setzen dieses Bits die niedriger gewichteten Devices von Interruptanforderungen abgekoppelt werden. Im TT/MEGA STE nicht verwendet.

MIE (Master Interrupt Enable)
Bei gelöschtem Bit werden keine Interruptanforderungen gestellt.

Status High/_Status Low Der Zustand dieses Bits legt fest, welche Bits der Interrupt-Vektornummer durch Statusinformationen verändert werden können.

Bit = 0: Bits 1..3 der Vektornummer beeinflussen
Bit = 1: Bits 6..4 der Vektornummer beeinflussen.

Siehe dazu die Erläuterungen zu Write-Register 2.

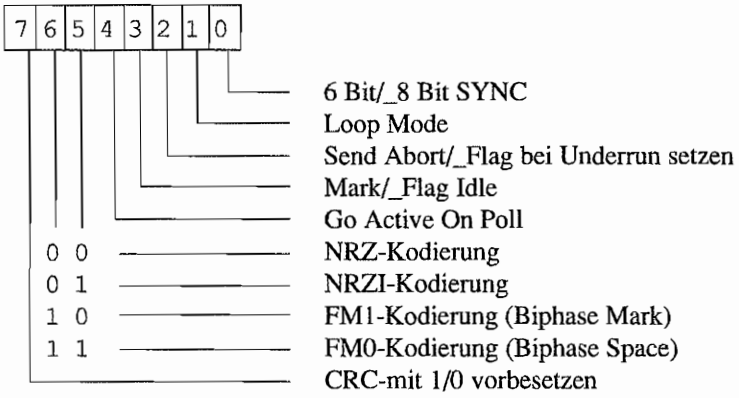
Interrupt Acknowledge ignorieren
: Bei gesetztem Bit wird ein Interrupt-Acknowledge-Zyklus vom SCC ignoriert.

Kein RESET
: Dieses Kommando wird nur benötigt, wenn auf dieses Register zugegriffen werden, dabei aber kein Reset ausgelöst werden soll.

Wenn ein RESET für einen der SCC-Kanäle ausgelöst wird, werden Sender und Empfänger des entsprechenden Kanals gesperrt. Modem-Steueranschlüsse wie RTS gehen auf High. Alle "pending" Interrupts (anstehenden Interrupts) und "Interrupts Under Service" werden zurückgesetzt und Interrupts gesperrt.

Write-Register 10
(Verschiedene Sender/Empfänger-Kontrollbits)

In diesem Register sind noch ein paar Bits zur Steuerung des Sende- und Empfangsteils untergebracht.



6 Bit/_8 Bit SYNC
: Mit gesetztem Bit kann man bei synchroner, zeichenorientierter Betriebsart ein spezielles Format für die SYNC-Zeichen wählen. Diese sind dann im MONOSYNC-Modus nur 6 Bits sowohl für Empfänger als auch Sender lang. Im BISYNC-Modus werden dann nur 12 Bit-SYNC-Zeichen beim Empfang ausgewertet, beim Senden allerdings weiter 16 Bits verwendet.

Loop Mode
: Im SDLC/HDLC-Betrieb wird eine spezielle Betriebsart für Mehrpunktverbindungen verwendet, bei der eine Zentralstation

mehrere Unterstationen steuert. Die Unterstationen sind in einem Ring hintereinander geschaltet und arbeiten als Repeater (Wiederholer). Dabei leitet die Unterstation die von der vorher liegenden Station empfangenen Daten zur dahinterliegenden Station weiter. Dieses Prinzip wird auch als "Token Passing Ring" bezeichnet.

Bei dieser Betriebsart werden vom SCC alle empfangenen Bits mit einem 1 Bit-Delay über den Sender wieder ausgegeben. Wenn bei einem Poll (= Aufforderung zum Senden) die Unterstation "merkt" daß sie angesprochen wurde (über das Adreßfeld erkennbar), "klinkt" sie sich mit einem oder mehreren entsprechenden "Antwort-Frames" in den Ring ein.

Durch Setzen des "Loop Mode"-Bits werden Sender und Empfänger zunächst zusammengeschaltet, um den empfangenen Datenstrom direkt durchzulassen. Wenn dann das "Go Aktive On Poll"-Bit gesetzt wird und mindestens 7 Einsbits (="End Of Poll"-Zeichen = EOP = FE_{hex} oder Ruhezustand mit Dauerlage "1" auf der Empfangsleitung) empfangen wurden, geht der SCC "On Loop" und schaltet einen 1 Bit-Delay zwischen Empfänger- und Senderleitung.

Bevor eine Unterstation aber selbst senden darf, muß nach dem "On Loop"-Schalten ein Flag empfangen werden. Dann hat die Station ein weiteres EOP abzuwarten, von dem sie allerdings das letzte Einsbit direkt in eine Null umsetzt. Damit entsteht daraus für die nächste Unterstation ein Flag! Nun wird der Rest des Antwort-Frames der Unterstation ausgesendet, der am Schluß wieder mit einem Flag und einem EOP abgeschlossen werden muß. Wenn der Antwort-Frame gesendet wurde, verbleibt die Unterstation mit dem 1 Bit-Delay im Ring!

In synchroner, zeichenorientierter Betriebsart (MONOSYNC oder BISYNC) wird dieses Bit gesetzt, um, zusammen mit einem gesetzten "Go Aktive On Poll"-Bit, den Sender erst bei "eingerasteter" Zeichensynchronisierung freizugeben.

Send Abort/_Flag bei Underrun setzen

Nur in SDLC/HDLC-Betrieb von Bedeutung. Durch dieses Bit wird eingestellt, wie der SCC sich bei einem Tx Underrun-

Status verhält (das Senderegister wurde nicht mehr rechtzeitig mit Sendedaten "nachgeladen"). Bei gesetztem Bit und Tx Underrun-Status sendet der SCC ein Abbruch-Signal (8..13 Einsbits) und ein Flag statt einer CRC-Prüfbitfolge!

Bei gelöschtem Bit wird dagegen die CRC-Prüfbitfolge gesendet und zu Beginn dieses Aussendens das Tx Underrun/EOM-Bit in Read Register 0 gesetzt (Kann einen Ext./Status-Interrupt auslösen).

Dieses Bit sollte nach der Ausgabe des ersten Bytes an das Senderegister des SCC gesetzt werden. Ein Rücksetzen hat dann unmittelbar nach Übergabe des letzten Datenbytes an das Senderegister zu erfolgen, um ein einwandfreies Beenden des laufenden Frames mit CRC und Flag zu gewährleisten.

Mark/_Flag Idle

Auch dieses Bit ist nur bei SDLC/HDLC-Betrieb relevant. Bei gelöschtem Bit werden im Ruhezustand (Idle State) einer Verbindung Flags gesendet. Mit gesetztem Bit gibt der Sender Einsbits (in Bytegruppen) als Ruhezustand aus (sinnvoll für die Zentralstation in einem Token Passing Ring, zur Erzeugung des EOP!).

Go Active On Poll

Dieses Bit wird beim SDLC-Loop Mode verwendet, um den Sender des SCC aufzufordern, sich für einen oder mehrere Antwortframes in den Datenstrom einzuklinken.

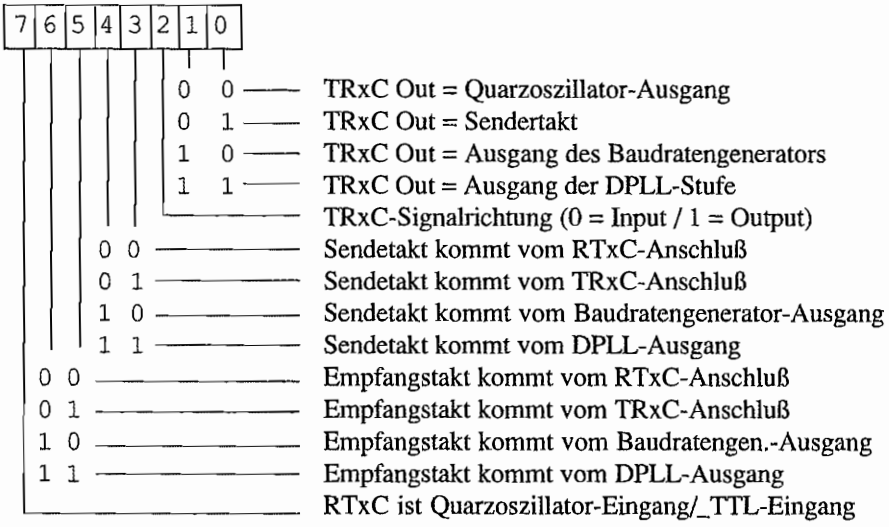
Bei gesetztem Bit "klinkt" sich die Unterstation mit einem Flag in den Ring ein. Die zu sendenden Daten müssen dann in das Sendedatenregister übergeben werden.

Bevor die CRC-Prüfbitfolge gesendet wird, muß dieses Bit wieder zurückgesetzt sein, sonst werden zusätzliche Flags gesendet. Es ist sinnvoll, dieses Bit nur dann zu setzen, wenn ein Poll für die eigene Adresse empfangen wurde.

CRC-mit 1/0 vorbesetzen

Dieses Bit bestimmt den Ausgangszustand des CRC-Schieberegisters im Sender *und* Empfänger. Bei gesetztem Bit werden alle Schieberegisterstufen mit "1" vorbesetzt. Bei gelöschtem Bit erfolgt die Voreinstellung entsprechend mit "0".

Write-Register 11
(Takteinstellung)



TRxC Out = ...

Diese beiden Bits bestimmen, welches Signal am TRxC-Pin abgegriffen werden kann (natürlich nur dann von Bedeutung, wenn TRxC als Ausgang programmiert ist!). Im TT/MEGA STE wird der TRxC-Anschluß für beide SCC-Kanäle als *Eingang* verwendet!

RTxC ist Quarzoszillator-Eingang/_TTL-Eingang

Im TT/MEGA STE wird entweder ein eigener TTL-Taktgenerator (Kanal A, 3,672 MHz) oder der Takt vom Timer C des TT-MFP verwendet (Kanal B). Deshalb ist das Bit gelöscht!

Write-Register 12 + 13
(Low-Byte + High-Byte für Baudratentimer)

Im SCC ist für jeden Kanal ein eigener Baudratengenerator eingebaut, der mit einem voreinstellbaren Abwärtszähler realisiert ist. Der Abwärtszähler ist ein 16 Bit-Zähler, dessen Voreinstellung durch die beiden Bytes in den Write-Registern 12 und 13 vorgenommen wird.

Der Baudratengenerator kann zum einen vom SCC-Mastertakt (8 MHz im TT/MMEGA STE) gespeist werden. Die andere Taktquelle im TT/MEGA STE ist der RTxC-Anschluß.

Die Ermittlung der Zeitkonstante, die für eine bestimmte Baudrate bei einem vorgegebenen Takt gewählt werden muß, geschieht nach folgendem Schema

$$\text{Konstante} = \frac{\text{Taktfrequenz}}{2 \times \text{Baudrate} \times \text{Taktmodus}} - 2$$

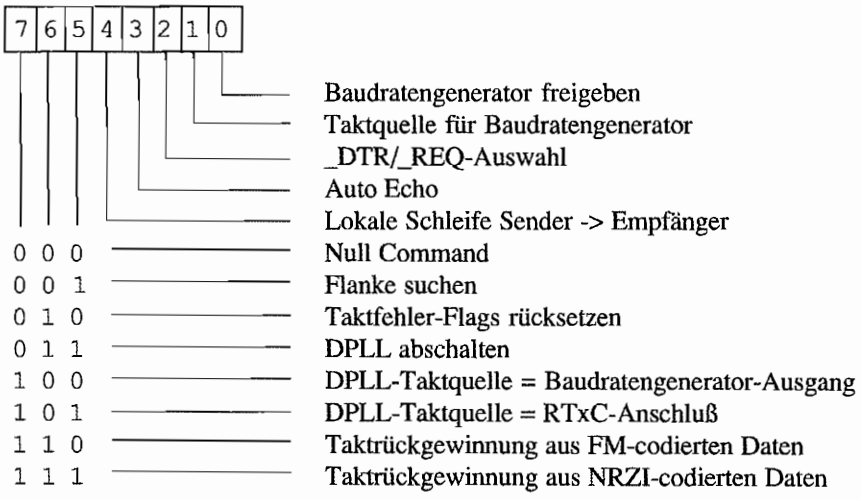
Taktfrequenz: Eingangsfrequenz des Baudratengenerators

Taktmodus: Anzahl Taktzyklen je Bit (bei einem Taktmodus von z. B. 16 werden 16 Taktzyklen für die Ausgabe eines Schritts (Bits) benötigt).

Das Highbyte der ermittelten Zeitkonstante gehört dann in Write-Register 13. Das Lowbyte dementsprechend in Write Register 12.

Write-Register 14
(Allgemeine Steuerbits)

In diesem Write Register finden sich unter anderem Steuerbits für den Baudratengenerator und die Digital Phase Locked Loop-Stufe (DPLL).



Taktquelle für Baudratengenerator

Bei gesetztem Bit wird der PCLK des SCC als Mastertakt für den Baudratengenerator verwendet. Bei dem im TT/

MEGA STE benutzten Takt von 8 MHz sind damit bei einem Taktmodus von 16 Baudraten im Asynchronmodus bis zu 125000 Bit/s erreichbar.

Wenn das Bit gelöscht ist, kommt der Takt vom RTxC-Anschluß. Im TT/MEGA STE liegt für den Kanal A an RTxCA ein 3,762 MHz-Takt an. Der Kanal B wird an RTxCB vom Timer C-Ausgang des TT-MFP gespeist.

_DTR/_REQ-Auswahl

Mit diesem Bit wird eingestellt, ob der _DTR/_REQ-Anschluß des SCC als Data Terminal Ready-Ausgang (Bit = 0) oder als Request-Ausgang für DMA-Betrieb verwendet wird (Bit = 1). Im TT/MEGA STE wird die DTR-Funktion benutzt!

Auto Echo

Bei gesetztem Bit wird intern der Senderausgang auf den Empfängereingang geschaltet. Der Empfänger reagiert aber weiterhin auf Empfangsdaten von "außen". CTS als Senderfreigabe wird ignoriert.

Lokale Schleife Sender -> Empfänger

Bei gesetztem Bit werden die Sendedaten sowohl am Ausgangsanschluß ausgegeben als auch auf den Empfänger zurückgeführt. CTS und DCD werden ignoriert.

Flanke suchen

Die DPLL-Stufe schaltet bei gesetztem Bit in einen Suchmodus, mit dem sie eine Flanke zum "Einrasten" im ankommenden Datenstrom sucht.

Zur einwandfreien Betriebsweise der DPLL muß bei *NRZI-kodierten Datenströmen* ein Quellentakt verwendet werden, der das 32fache der Datenrate beträgt. Nachdem eine Datenflanke erkannt wurde, schaltet die DPLL auf Normalbetrieb um und justiert sich an weiteren Flanken jeweils nach.

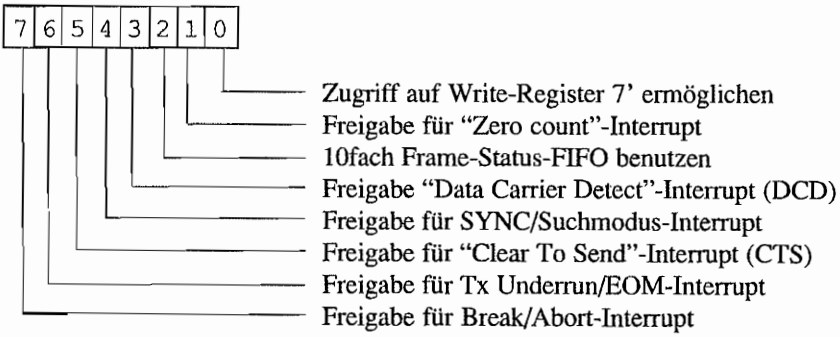
Bei *FM-kodierten Datenströmen* wird die DPLL-Stufe mit Quelltakten "gefahren", die das 16fache der Datenrate aufweisen. Wenn weitere Flanken nicht im erwarteten Taktraster gefunden werden, setzt die DPLL entsprechende Statusbits ("One Clock Missing"- und "Two Clock Missing"-Bit in Read Register 10). Wird zweimal die Datenflanke an der erwarteten Position nicht mehr erkannt, wechselt die DPLL-Stufe automatisch wieder in den "Flanke suchen"-Modus.

Taktfehler-Flags rücksetzen — Mit diesem Kommando werden die zuvor erwähnten "One Clock Missing"- und "Two Clock Missing"-Bits zurückgesetzt.

DPLL abschalten — Die DPLL-Stufe wird abgeschaltet, die "Clock Missing"-Bits zurückgesetzt und ein Dauer-"Flanke suchen"-Modus eingeschaltet.

Write-Register 15
(Ext./Status-Interrupt-Steuerung)

Wenn Ext./Status-Interrupts durch Write-Register 1 eingeschaltet sind, wird in diesem Write-Register festgelegt, welche Zustände einen Interrupt auslösen können.



Zugriff auf Write Register 7' ermöglichen
Bei gesetztem Bit kann im SDLC/HDLC-Modus auf das Write Register 7' mit seinen zusätzlichen Möglichkeiten zugegriffen werden.

Freigabe für "Zero count"-Interrupt
Bei gesetztem Bit wird ein Interrupt ausgelöst, wenn der Zähler im Baudratengenerator den Wert 0 erreicht.

10fach Frame-Status-FIFO benutzen
Bei gesetztem Bit ist im SDLC/HDLC-Modus der 10fach Frame-Status-FIFO eingeschaltet.

Freigabe "Data Carrier Detect"-Interrupt (DCD)
Bei gesetztem Bit führt eine Zustandsänderung am DCD-Anschluß zu einem Interrupt.

Freigabe für SYNC/Suchmodus-Interrupt

Bei gesetztem Bit wird im Asynchronmodus bei einer Zustandsänderung am SYNC-Anschluß des SCC ein Interrupt ausgelöst.

Im TT/MEGA STE wird der SYNC-Anschluß bei beiden SCC-Kanälen für die Abfrage der "Data Set Ready (DSR)"-Leitung benutzt.

Im Synchron-Modus erfolgt eine Interruptauslösung, wenn ein SYNC-Zeichen oder Flag im Empfangsdatenstrom erkannt wurde.

Freigabe für "Clear To Send"-Interrupt (CTS)

Bei gesetztem Bit wird ein Interrupt durch Zustandsänderungen am CTS-Anschluß ausgelöst.

Freigabe für Tx Underrun/EOM-Interrupt

Wenn im Sender das Tx Underrun/EOM-Flag gesetzt wird, löst das einen Interrupt aus.

Freigabe für Break/Abort-Interrupt

Bei gesetztem Bit werden Interrupts ausgelöst, wenn vom Empfänger ein Break (Dauerlage "0") im Asynchronmodus oder ein Abort (mindestens sieben Einsbits) im SDLC/HDLC-Modus erkannt wurde.

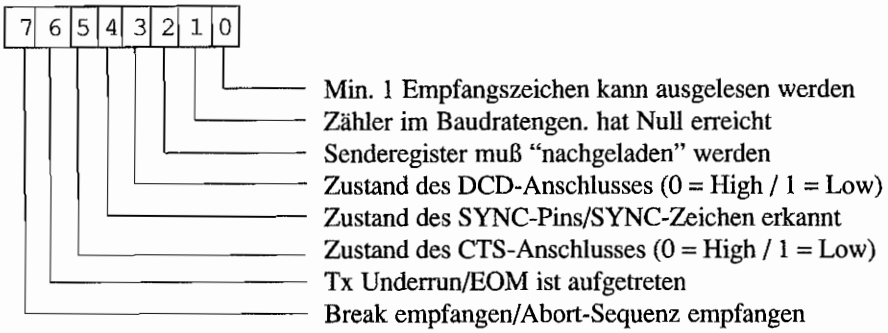
Soviel zu den Write-Registern des SCC.

Read-Register

Nun kommen die Read Register an die Reihe, die dem Programmierer eine Menge an Statusinformationen und natürlich auch die empfangenen Daten liefern.

Read-Register 0
(Status der Sende- und Empfangsbuffer)

In diesem Register finden sich Flags für die wohl wichtigsten Zustände des Sende- und Empfangsteils des SCC wieder. Unter anderem findet man hier auch die Flags wieder, die als Quellen für Ext./Status-Interrupts fungieren.



Hier noch einige zusätzliche Erläuterungen zu bestimmten Flags:

Zähler im Baudratengen. hat Null erreicht

Wenn im Write Register 15 der "Zero Count"-Interrupt freigegeben ist, wird dieses Bit so lange auf "1" gesetzt, wie im Abwärtszähler des Baudratengenerators der Zählerstand Null existiert! Solange bleibt auch eine "Zero Count"-Interruptanforderung "stehen". Wenn allerdings zu diesem Zeitpunkt bereits ein anderer Ext./Status-Interrupt "anhängig" (pending) ist, wird ein "Zero Count"-Interrupt nur dann ausgelöst, wenn die Interruptbedingung (Abwärtszähler ist auf Null) auch nach Beendigung der gerade laufenden Interrupt-Service-Routine noch besteht!

Zustand des DCD-Anschlusses

Wenn der DCD-Interrupt (Bit 3 im Write Register 15) nicht freigegeben ist, kann über dieses Bit der Zustand der DCD-Leitung abgefragt werden (0 = High / 1 = Low).

Ist der DCD-Interrupt freigegeben, wird bei einem Ext./Status-Interrupt der Zustand des DCD-Anschlusses zum Interruptzeitpunkt "eingefroren" und kann durch die Interrupt-Service-Routine abgefragt werden. Wenn gerade kein anderer Ext./Status-Interrupt "pending" ist, löst ein Zustandswechsel am DCD-Anschluß natürlich einen entsprechenden Interrupt aus.

Zustand des SYNC-Pins/SYNC-Zeichen erkannt

Bei Synchronbetrieb wird dieses Bit durch das Kommando "Suchmodus einschalten" in Write Register 0 gesetzt. Sobald der Empfänger dann Zeichensynchronismus erreicht (durch

SYNC-Zeichen oder Flags im SDLC/HDLC-Modus), wird dieses Bit wieder zurückgesetzt. Wenn im Write Register 15 "Freigabe für SYNC/Suchmodus-Interupts" eingestellt ist, wird bei jeder Änderung dieses Flags ein Ext./Status-Interrupt ausgelöst.

Im Asynchronbetrieb signalisiert dieses Bit den Zustand des SYNC-Anschlusses des jeweiligen SCC-Kanals (1 = Low / 0 = High am SYNC-Pin). Im TT/MEGA STE läßt sich über den SYNC-Anschluß so die "Data Set Ready (DSR)"-Leitung eines MODEM auswerten.

Zustand des CTS-Anschlusses

Dieses Bit reagiert für den CTS-Anschluß des SCC-Kanals genauso, wie sich Bit 3 für den DCD-Anschluß verhält.

Tx Underrun/EOM ist aufgetreten

Dieses Bit ist nach einem RESET, bei abgeschaltetem Sender und nach einem "Send abort"-Kommando gesetzt. Ein Rücksetzen ist nur durch das "RESET Tx Underrun/EOM-Bit"-Kommando in Write Register 0 möglich. Sobald dann eine Tx Underrun-Bedingung auftritt, wird dieses Bit wieder gesetzt und ein Ext./Status-Interrupt ausgelöst (wenn dieser über das Bit 6 im Write-Register 15 eingeschaltet wurde!).

Break empfangen/Abort-Sequenz empfangen

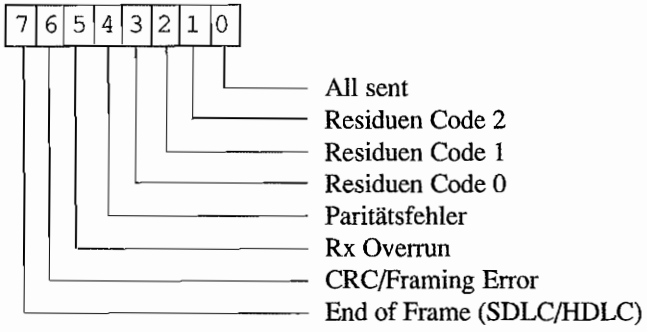
Im *Asynchron-Betrieb* wird dieses Bit gesetzt, wenn ein Break im einlaufenden Datenstrom "gesichtet" wurde (ein Zeichen, bei dem alle Bits Null sind *und* kein Stopbit erkannt wurde = Framing Error). Dieses Break erzeugt im Empfangs-FIFO-Buffer ein einzelnes "Nullzeichen", welches ausgelesen und verworfen werden sollte. Mit Übergabe des "Nullzeichens" in den Empfangs-FIFO-Buffer wird dieses Bit allerdings schon wieder zurückgesetzt! Wenn das zugehörige Enable-Bit im Write Register 15 gesetzt ist, wird natürlich ein Ext./Status-Interrupt ausgelöst.

Im *SDLC/HDLC-Betrieb* wird dieses Bit gesetzt, wenn eine Abort-Sequenz (sieben oder mehr Einsbits) erkannt wurde. Wenn die Abort-Sequenz beendet wird, wird dieses Bit wieder zurückgesetzt. Auf jeden Fall wird bei eingeschalteter "Break/

Abort"-Freigabe Bit in Write Register 15 ein Ext./Status-Interrupt ausgelöst, egal ob bereits ein solcher Interrupt "anhängig" ist oder nicht!!

Read-Register 1
(Special Receive Condition-Bits)

Aus diesem Register können unter anderem die Statusinformationen zu dem parallel im Empfangs-FIFO-Buffer liegenden Empfangszeichen ausgelesen werden.



All sent — Im Asynchronbetrieb wird dieses Bit gesetzt, wenn alle Sendedatenbits das Sendeschieberegister verlassen haben.

Residuen Code 2, 1 und 0 — Diese Bits sind nur im SDLC/HDLC-Betrieb interessant und das auch nur dann, wenn das End of Frame-Bit ebenfalls gesetzt ist.

Wenn nämlich das Datenfeld eines Frames kein ganzzahliges Vielfaches der eingestellten Zeichenlänge (üblicherweise acht Bits) ist, werden diese Residuen Bits herangezogen, um zu ermitteln, wo die Grenze zwischen den letzten Bits des Datenfelds eines Frames und der CRC-Prüfbitfolge liegt! Empfangene Datenbits in einem Zeichen werden immer rechts justiert!

Die folgende Tabelle zeigt die Zusammenhänge zwischen den Residuen Bits und der Lage der Datenbits in den letzten gelesenen Zeichens eines Frames für unterschiedliche Zeichenlängen.

Res. Bits 0 1 2	Bits im letzten Byte 8B /Z	7B /Z	6B /Z	5B /Z	Bits im vorletzten Byte 8B /Z	7B /Z	6B /Z	5B /Z	Bits im drittletzten Byte 8B /Z	7B /Z	6B /Z	5B /Z
1 0 0	0	0	0	0	3	1	0	0	8	7	5	2
0 1 0	0	0	0	0	4	2	0	0	8	7	6	3
1 1 0	0	0	0	0	5	3	1	0	8	7	6	4
0 0 1	0	0	0	0	6	4	2	0	8	7	6	5
1 0 1	0	0	0	0	7	5	3	1	8	7	6	5
0 1 1	0	0	0	-	8	6	4	-	8	7	6	-
1 1 1	1	0	-	-	8	7	-	-	8	7	-	-
0 0 0	2	-	-	-	8	-	-	-	8	-	-	-

Paritätsfehler

Wenn die Paritätsprüfung eingeschaltet ist, wird dieses Bit bei jenen Zeichen gesetzt, die ein nicht übereinstimmendes Paritätsbit aufweisen. Wenn ein Paritätsfehler auftrat, bleibt dieses Bit auch bei allen folgenden Empfangszeichen gesetzt (und es werden evtl. so lange Special Condition-Interrupts ausgelöst), bis mit dem Error Reset-Kommando dieser Zustand gelöscht wird!

Rx Overrun

Ein gesetztes Bit zeigt einen Überlauf des Empfangs-FIFO-Buffers an. Aber nur das Zeichen, für den dieser Überlauf signalisiert wurde ist "überrannt" worden. Allerdings bleibt die Fehlerbedingung für alle weiteren Empfangszeichen gesetzt, bis ein Error Reset ausgeführt wird.

CRC/Framing Error

Im Asynchronbetrieb wird durch ein gesetztes Bit ein Framing Error signalisiert. Damit wird angezeigt, daß zu einem Empfangszeichen kein Stopschritt (Bit) empfangen wurde. Dieser Rahmenfehler wird nur für das Zeichen signalisiert, bei dem dieser Fehler auftrat (also nicht bis zu einem Error Reset-Kommando ge"latched").

Im Synchronbetrieb wird dieses Bit gesetzt, wenn beim Vergleich der augenblicklichen Prüfbitfolge in der CRC-Stufe mit dem festgelegten OK-Prüfmuster keine Übereinstimmung festgestellt wurde. Das dürfte meistens der Fall sein, außer wenn der Datenblock komplett fehlerfrei empfangen wurde. Das Bit wird auch hier nicht gelatched und kann jederzeit mit einem Error Reset-Kommando gelöscht werden. Mit jedem neuen

empfangenen Zeichen wird dieses Bit erneut dem Vergleich entsprechend gesetzt. Um eine korrekte Statusaussage über die CRC-Prüfung zu erhalten, sollte dieses Bit nur in Verbindung mit dem "End of Frame"-Bit ausgewertet werden.

End of Frame — Im SDLC/HDLC-Modus wird durch ein gesetztes Bit angezeigt, wenn ein gültiges "Closing"-Flag empfangen wurde. Damit sind dann auch das CRC/Framing Error-Bit und die Residuen Code-Bits gültig. Mit dem ersten empfangenen Zeichen des nächsten Frames wird der Zustand dieses Bits wieder aktualisiert (also zurückgesetzt!). Ein Rücksetzen ist ebenfalls mit dem Error Reset-Kommando möglich.

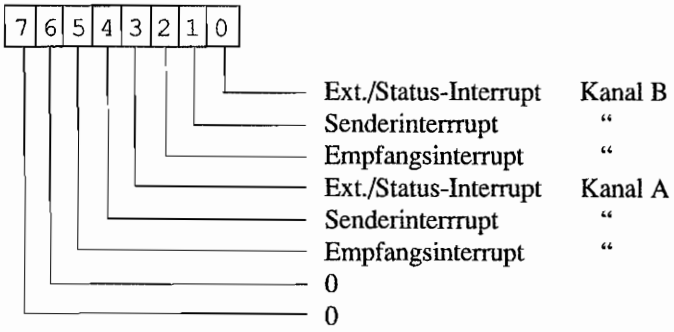
Read-Register 2
(Interrupt-Vektor)

Aus diesem Register kann der Interrupt-Vektor ausgelesen werden, der über das Write Register 2 in den SCC eingeschrieben wurde. Dabei gibt es jedoch eine Besonderheit zu beachten:

- Wenn dieses Register im Kanal B ausgelesen wird, erhält man den aktuellen, evtl. durch Statusinformationen veränderten Interrupt-Vektor.
- Erfolgt das Auslesen aus dem Kanal A, erhält man lediglich den aktuell dort gespeicherten Interrupt-Vektor ohne Statusbeeinflussung.

Read-Register 3
(Interrupt Pending Register)

Hier werden alle "anhängigen" Interrupts durch ein gesetztes Bit signalisiert. Allerdings kann dieses Register nur vom Kanal A ausgelesen werden! Im Kanal B erscheinen alle Bits als Nullbits.



Read-Register 6
(LSB des 14 Bit-Zählerstands aus Frame-Status-FIFO-Buffer)

Wenn der SCC im SDLC/HDLC-Betrieb arbeitet und Bit 2 im Write Register 15 gesetzt ist, können hier die untersten 8 Bits des an oberster Position des Frame-Status-FIFO-Buffers stehenden Frame-Byte-Zählerstandes ausgelesen werden. Die restlichen sechs Zählerstandsbits kommen aus Read Register 7.

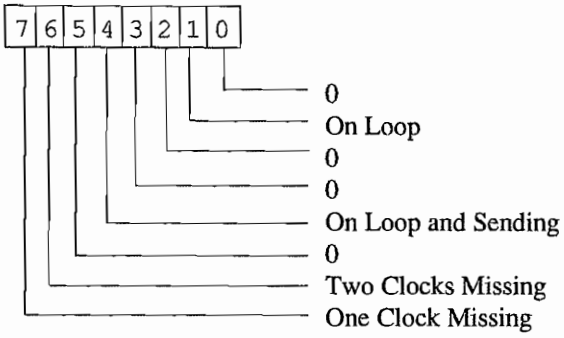
Read-Register 7
(MSB des 14 Bit-Zählerstands aus Frame-Status-FIFO-Buffer)

Auch dieses Register ist nur "zugänglich", wenn im Write Register 15 das Bit 2 gesetzt ist. Die unteren sechs Bits dieses Registers enthalten die höchstwertigen Zählerstandsbits aus dem 10fach Frame-Status-FIFO-Buffer.

Ein gesetztes *Bit 7* dieses Registers signalisiert einen Überlauf des 10fach Frame-Status-FIFO-Buffers.

Wenn *Bit 6* gesetzt ist, kommt die Statusinformation für den Frame aus dem FIFO-Buffer. Bei Bit 6 = 0 kommt die Statusinformation nicht aus dem Zwischenbuffer (der ist dann noch leer), sondern direkt aus dem Empfänger (Read Register 1)!

Read-Register 10
(Loop/Clock Status)



On Loop	Dieses Bit ist gesetzt, wenn sich der SCC im SDLC-Loop-Modus befindet.
On Loop and Sending	Dieses Bit wird im SDLC-Loop-Modus dann gesetzt, wenn der Sender selbst Daten auf den Ring ausgibt.

Two Clocks Missing — Wenn die DPLL-Stufe im FM-Modus betrieben wird und diese bei zwei Versuchen keine Datenflanke erkannt hat, wird dieses Bit gesetzt. Gleichzeitig schaltet die DPLL in den Suchmodus! Das Bit bleibt dann gesetzt, bis ein "Taktfehler Flags rücksetzen"- oder ein "Flanke suchen"-Kommando über Write-Register 14 gegeben wird.

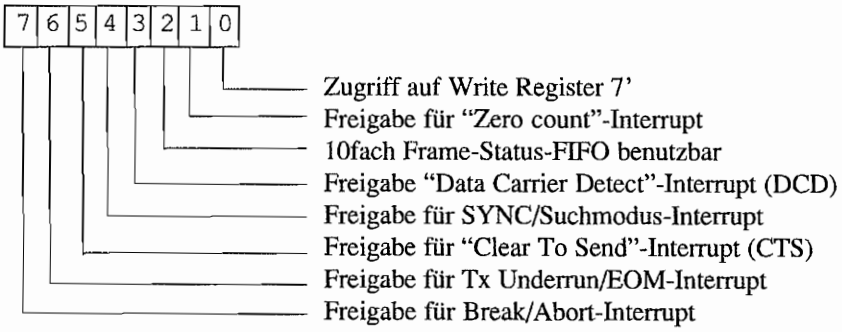
One Clock Missing — Wenn die DPLL-Stufe im FM-Modus betrieben wird und diese keine Datenflanke an der Stelle erkannt hat, wo eigentlich eine sein müßte, wird dieses Bit gesetzt. Das Bit bleibt dann gesetzt, bis ein "Taktfehler Flags rücksetzen"- oder ein "Flanke suchen"-Kommando über Write-Register 14 gegeben wird.

Read-Register 12 und 13
(Low- und High-Byte der Baudratengen.-Zeitkonstanten)

Aus diesen beiden Registern kann die über Write-Register 12 und 13 eingestellte Zeitkonstante für den Abwärtszähler des Baudratengenerators ausgelesen werden.

Read-Register 15
(Ext./Status-Interrupt-Freigabe-Bits)

Die über das Write-Register 15 eingestellten Freigabe-Bits für Ext./Status-Interrupts können hierüber wieder ausgelesen werden. Die Beschreibung der einzelnen Bits kann bei Write-Register 15 nachgeschlagen werden.



Die Programmierung des SCC

Die Programmierung des SCC mit seinen vielfältigen Möglichkeiten muß natürlich entsprechend sorgfältig vorgenommen werden. Aus diesem Grunde hat der Hersteller des SCC ein

bestimmtes Schema vorgeschlagen, um die Initialisierung des SCC vorzunehmen. Die Einstellung des SCC erfolgt dabei in einer vorgegebenen Reihenfolge in drei Abschnitten.

1. Einstellung der Betriebsmodi wie Bits/Zeichen, Parität. Außerdem gehört dazu das Vorbesetzen der einzelnen Register mit Konstanten (z. B. Interrupt-Vektor, Zeitkonstante für Baudratengenerator).

2. Im nächsten Schritt erfolgt die Freigabe der einzelnen Hardware-Funktionen. Dazu gehört unter anderem die Freigabe für Sender und Empfänger sowie Baudratengenerator und DPLL. Wichtig ist dabei, daß die Betriebsmodi vorher eingestellt wurden.

3. Der dritte Abschnitt ist optional und nur von Belang, wenn mit Interrupts gearbeitet wird. Dazu gehört dann die Freigabe der benutzten Interrupts.

Nachfolgend deshalb das Schema für die Initialisierung des SCC. Diese Prozedur ist natürlich für jeden Kommunikationskanal durchzuführen.

Dabei sind jene Register besonders gekennzeichnet, deren Programmierung nicht zwingend erforderlich ist. Alle mit einem "X" markierten Bitpositionen sind dabei vom Programmierer einzustellen. Ein "S"-Bit bedeutet, daß dieses Bit in seinem bereits zuvor eingestellten Zustand zu belassen ist.

Betriebsmodi und Konstanten einstellen

Write Reg.#	Bitbelegung	Aktion	Anm.
9	1 1 0 0 0 0 0 0	Hardware Reset ausführen	
4	X X X X X X X X	Tx/Rx-Cont. Async/Sync-Modus	
1	0 X X 0 0 X 0 0	W/_REQ f. DMA-Betrieb einst.	*opt.*
2	X X X X X X X X	Int.-Vektor programmieren	*opt.*
3	X X X X X X X 0	Empf.-Parameter einstellen	
5	X X X X 0 X X X	Sender-Parameter einstellen	
6	X X X X X X X X	SYNC-Zeichen programmieren	*opt.*
7	X X X X X X X X	SYNC-Zeichen programmieren	*opt.*
9	0 0 0 X 0 X X X	Master-Int. Steuerung	
10	X X X X X X X X	Div. Tx/Rx-Einstellungen	*opt.*
11	X X X X X X X X	Takteinstellungen	
12	X X X X X X X X	BRG-Zeitkonstante Low-Byte	*opt.*
13	X X X X X X X X	BRG-Zeitkonstante High-Byte	*opt.*
14	X X X X X X X 0	Allg. Steuerbits einstellen	
14	X X X S S S S S	Verschiedene Kommandos	*opt.*

Freigabe der Hardware-Funktionen

Write Reg.#	Bitbelegung	Aktion	Anm.
14	0 0 0 S S S S 1	Baudratengen. freigeben	
3	S S S S S S S 1	Empfänger freigeben	
5	S S S S 1 S S S	Sender freigeben	
0	1 0 0 0 0 0 0 0	Tx CRC-Gener. zurücksetzen	*opt.*
1	X S S 0 0 S 0 0	DMA-Betrieb freigeben	*opt.*

Freigabe der Interrupts

Write Reg.#	Bitbelegung	Aktion	Anm.
15	X X X X X X X X	Ext./Status-Int. selektieren	
0	0 0 0 1 0 0 0 0	Reset Ext./Status-Int.	
0	0 0 0 1 0 0 0 0	Reset Ext./Status-Int. (nochmal!)	
1	S S S X X S X X	Rx/Tx + Ext./Status-Int. aktivieren	
9	0 0 0 S X S S S	"Hauptschalter" für Int. auf "Ein"	

Vor der Initialisierung des SCC sollte auf jeden Fall ein Reset ausgeführt werden. Wenn bereits ein Kanal initialisiert ist, braucht natürlich für die Initialisierung des zweiten Kanals kein erneuter Hardware-Reset durchgeführt zu werden. Dazu "reicht" dann ein Kanal-Reset für den zweiten Kanal!

Betrieb des SCC im TT

Auch dem SCC hat ATARI für den schnellen Datenaustausch eine eigene DMA-Einheit zur Seite gestellt. Verwendet wird ein zweiter Satz mit den gleichen DMA-Chips, wie bereits beim SCSI-Port des TT beschrieben. Allerdings gilt die DMA-Unterstützung nur für den Kanal A des SCC!

SCC-DMA-Betrieb ist auch hier auf jede RAM-Adresse möglich. Weil die grundsätzlichen Angaben zu den DMA-Bausteinen ja bereits bei der Beschreibung des SCSI-Ports gemacht wurden, soll hier nur auf die Lage der SCC-DMA-Register im Adreßraum eingegangen werden.

Die SCC-DMA-Register

Die Registeradressen des SCC-DMA-Chips liegen sowohl am oberen Ende des ST-Adreßraums ($FF 8CXX) als auch ganz oben im TT-Adreßraum ($FFFF 8CXX). Angegeben ist hier jeweils die Lage im ST-Adreßraum!

Adresse	Zugriff	Bezeichnung	Label
$FF 8C01	R/W	DMA-Address-Pointer (Highest Byte)	"scdmabas"
$FF 8C03	R/W	DMA-Address-Pointer (High Byte)	"scdmabas"+2
$FF 8C05	R/W	DMA-Address-Pointer (Low Byte)	"scdmabas"+4
$FF 8C07	R/W	DMA-Address-Pointer (Lowest Byte)	"scdmabas"+6
$FF 8C09	R/W	DMA-Bytezähler (Highest Byte)	"scdmacnt"
$FF 8C0B	R/W	DMA-Bytezähler (High Byte)	"scdmacnt"+2
$FF 8C0D	R/W	DMA-Bytezähler (Low Byte)	"scdmacnt"+4
$FF 8C0F	R/W	DMA-Bytezähler (Lowest Byte)	"scdmacnt"+6

Auch hier kann für den Long-Zugriff auf diese Register in Maschinensprache ein "movep.l"-Befehl benutzt werden!

Adresse	Zugriff	Bezeichnung	Label
$FF 8C10	R	Restdatenregister (High-Word)	"scdmarsd"
$FF 8C12	R	Restdatenregister (Low-Word)	"scdmarsd"+2

Wenn am Schluß einer DMA-Operation kein voller Long mehr in den Speicher übertragen wurde, muß hier der Rest an Datenbytes "per Hand" ausgelesen und durch die CPU an die richtigen Adressen gebracht werden! Wie viele Restbytes das sind, erfährt man aus den letzten beiden Adreßbits des DMA-Address-Pointers.

Die Startadresse n für die Restbytes ergibt sich aus dem Inhalt des DMA-Address-Pointers minus der Anzahl an Restbytes. Die Restbytes sind "linksbündig" im Restdatenregister angeordnet. Also gehört das Highbyte im High-Word an die Startadresse n. Das Lowbyte im Highword kommt nach Startadresse n+1 usw.!

Adresse	Zugriff	Bezeichnung	Label
$FF 8C14	R/W	Kontrollregister (Word)	"scdmactl"

Obwohl dieses Register mit Wordzugriffen bearbeitet wird, sind nur die untersten 8 Bit benutzt. Man kann deshalb auch mit Bytezugriffen auf die Adr. $(FF)FF 8C15 arbeiten, um diese

Kontrollbits zu beeinflussen bzw. auszuwerten. Hier deshalb nur die interessanten unteren 8 Bits des Registers:

Bit 7	6	5	4	3	2	1	0
Bus Error	Byte count zero	Reser-viert	Reser-viert	Reser-viert	Reser-viert	DMA Enable	DMA-Direct-ion

Bus Error — Ein gesetztes Bit weist auf einen Busfehler während des DMA-Betriebs hin. Durch Auslesen dieses Registers wird der Busfehler-Status wieder zurückgesetzt. Diese Fehlersignalisierung geht auch an den TT-MFP, I/O-Port #2 und kann bei entsprechender Programmierung einen Interrupt auslösen.

Byte count zero — Ein gesetztes Bit zeigt an, daß der DMA-Bytezähler auf Null gezählt hat, also alle Bytes transferiert sind.

DMA-Enable — Bei gesetztem Bit ist der DMA-Baustein "eingeschaltet".

DMA-Direction — Bit gesetzt: Schreiben an SCC-Controller.
Bit gelöscht: Lesen von SCC-Controller.

Um DMA-Betrieb durchzuführen, ist zunächst die Transferrichtung im DMA-Baustein (Bit 0 in "scdmactl") einzustellen. Anschließend wird die Basisadresse im DMA-Address-Pointer ("scdmabas") gesetzt und die Zahl der zu übertragenden Bytes im DMA-Bytezähler ("scdmacnt") eingestellt.

Wenn dann auch der SCC-Controller entsprechend programmiert ist, wird durch Setzen des "DMA-Enable"-Bits in "scdmactl" der DMA-Prozeß gestartet.

Die hardwaremäßige Einbindung des SCC beim TT

Der Schaltungsauszug in der nachfolgenden Abbildung zeigt die hardwaremäßige Einbindung des SCC beim TT. Mittels des XLAN-Signals vom Soundchip kann der Kanal A des SCC softwaregesteuert auf den LAN-Anschluß oder den SERIAL 2-Anschluß (9pol. SubD-Buchse) geschaltet werden. Nach neuesten Infos plant ATARI aber hier eine Änderung, die diese Umschaltmöglichkeit evtl. nicht mehr enthält. Vielmehr soll der hierfür benutzte Soundchip-Anschluß zusätzlich am Parallelport zur Verfügung gestellt werden, um eine einfachere Anschlußmöglichkeit für handelsübliche Scanner zu ermöglichen.

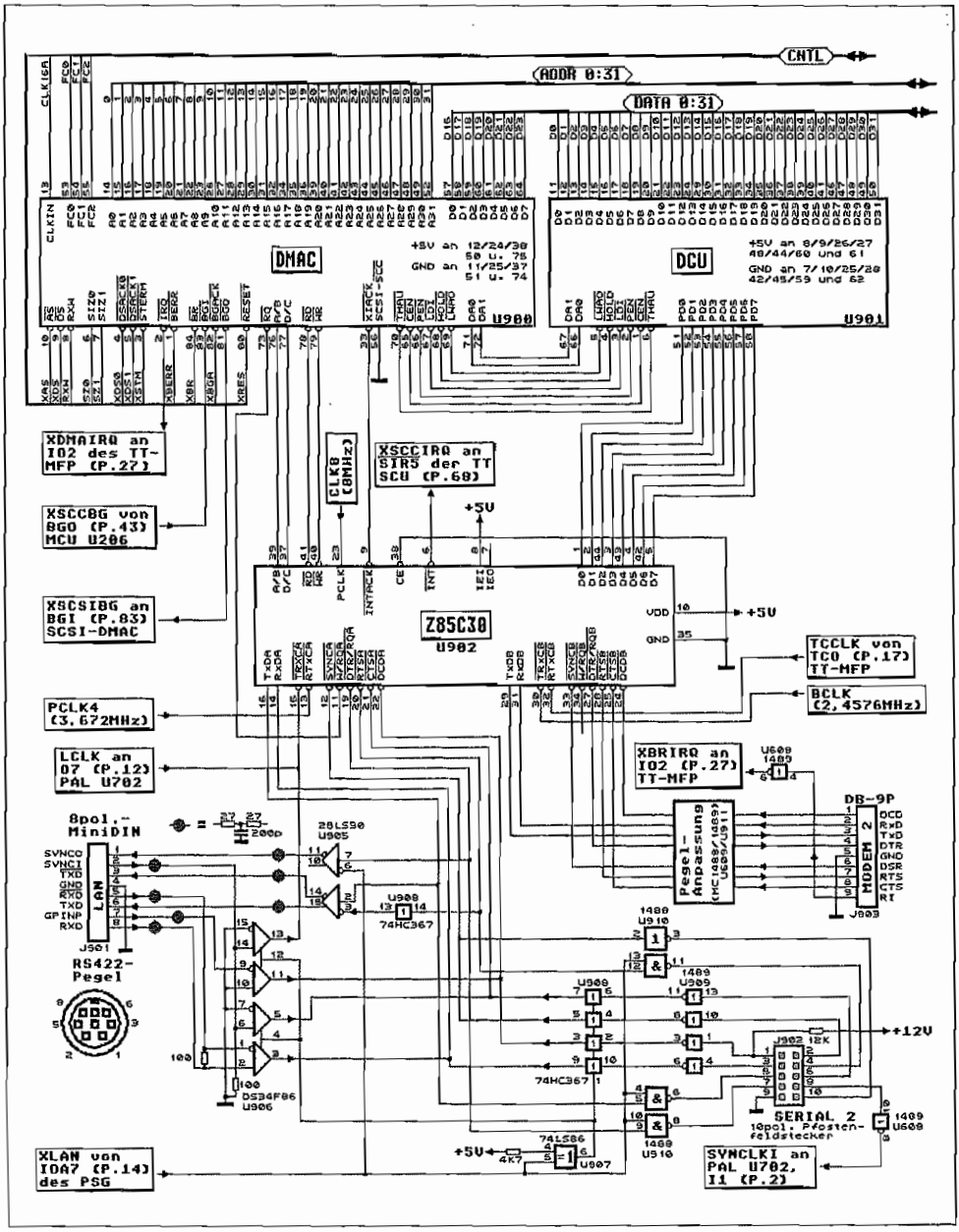


Abb. 7.8: Die hardwaremäßige Einbindung des SCC im TT

Initialisierung für Asynchronbetrieb (Poll-Betrieb)

Nachfolgendes "Mini"-Listing zeigt eine Beispielinitialisierung, um beim TT/MEGA STE den SCC-Kanal B (Buchse MODEM 2) für Asynchronbetrieb mit 38400 Bit/s einzustellen.

Dabei wird als Sende- und Empfangstakt der 2,4576 MHz-Takt der MFPs über den TRxCB-Anschluß dem SCC zugeführt. Durch Einstellung eines Verhältnisses von 64 Taktimpulsen pro Datenschritt ergibt sich eine Übertragungsgeschwindigkeit von 38400 Bit/s.

```
;-------------------------------------------------------------------
; Beispielinitialisierung des SCC-Kanals B für Asynchronbetrieb
; ohne Interrupts. Sender auf Empfänger "geschleift"!
; Taktung erfolgt über TRxCB-Anschluß mit 2,4576 MHz-MFP-Takt.
; Durch Takt/Datenrate = 64 ist eine Baudrate von 38400 Bd ein-
; gestellt.
; (H.-D. Jankowski / Juli '91 für ATARI TT/ST/STE Profibuch)

sccctl_b        equ   $ffff8c85  ; Controlreg. für SCC-B
sccdat_b        equ   $ffff8c87  ; Datenregister "  SCC-B

;-------------------------------------------------------------------
iniscc:
     lea        sccctl_b,a0      ; Pointer auf SCC-Kontrolreg. B
     lea        sccdat_b,a1      ; Pointer auf SCC-Datenreg. B
     lea        initable,a2      ; Zeiger auf Tabelle mit Ini.-Daten
     move.w     (a2)+,d1         ; Anzahl Tabelleneinträge
iniloop:
     move.b     (a2)+,(a0)       ; Initialisierungsdaten aus Tabelle
     move.b     (a2)+,(a0)       ; lesen und in SCC-B schreiben.
     dbra       d1,iniloop
iniend:

;-------------------------------------------------------------------
; Diese kleine Routine liest lediglich die zu sendenden Zeichen
; ab Adresse "sendtxt" ein und sendet sie über den SCC-B aus.
; Da dieser im Schleifenbetrieb gefahren wird, können diese so
; "ausgesendeten" Daten unmittelbar wieder eingelesen werden.
; Es erfolgt z. B. keinerlei Fehlerabfrage!

senden:
     lea        sendtxt,a3       ; Zeiger auf Textanfang
```

```
      lea       buffer,a4       ; Zielbuffer für empfangenen Text
recv:
      move.b    (a0),d1         ; Read Register 0 auslesen
      btst      #0,d1           ; Rx Buffer-Bit testen
      beq.s     send            ; Mind. 1 Zeichen da?
      move.b    (a1),(a4)+      ; Ja! Zeichen holen.
      beq.s     rcvend          ; Wenn Null empfangen wurde, raus!
send:
      move.b    (a0),d1         ; Read Register 0 auslesen
      btst      #2,d1           ; Tx Buffer empty-Bit testen
      beq.s     recv            ; Sendebuffer leer?
      move.b    (a3)+,(a1)      ; Ja! Zeichen holen und ausgeben
      bra.s     recv
rcvend:
      rts

      .data
;-------------------------------------------------------------------
; Tabelle mit Initialisierungsdaten für SCC-Kanal B

initable:
      dc.w      9               ; Anzahl Einträge in Ini.-Tabelle
      dc.b      $09,$C0         ; Hardware-Reset ausführen
      dc.b      $04,$CC         ; x64 Takt, 2 Stopbits, No Parity
      dc.b      $03,$C0         ; Rx 8 Bits/Zeichen, Rx disabled
      dc.b      $05,$60         ; Tx 8 Bits/Zeichen, DTR,RTS,Tx off
      dc.b      $09,$00         ; Interrupts abgeschaltet
      dc.b      $0A,$00         ; NRZ-Modus
      dc.b      $0B,$28         ; Tx & Rx = TrxC = Input
      dc.b      $0E,$10         ; BRG off, loopback
sccenabl:
      dc.b      $03,$C1         ; Rx enable
      dc.b      $05,$68         ; Tx enable

sendtxt:
      dc.b 'Dies ist der zu sendende Text!!!',0

      .bss
buffer:
      ds.b      50              ; Platz für 50 Empfangszeichen
```

Kapitel 8: Der VME-Busanschluß beim TT/MEGA STE

Die Atari-Computer wurden immer wieder wegen fehlender Erweiterungsmöglichkeiten (z. B. durch Erweiterungskarten) kritisiert. Diesen Kritikpunkt hat Atari erstmals mit Vorstellung des MEGA ST aufgegriffen und eine Erweiterungsmöglichkeit durch den Megabus geschaffen (siehe dazu auch im ST-Teil im Kapitel "Die Zentraleinheit"). Hierbei handelte es sich letztlich um einen Anschluß, an dem alle MC68000er-Prozessorsignale zur Verfügung stehen.

Beim TT/MEGA STE wollte man dann wohl eine einheitliche Linie finden und sich nicht auf die MC68000-Signale festbeißen (Schließlich hat man ja im TT auch eine CPU mit ganz anderen Möglichkeiten und z. T. auch anderem Signalverhalten vor sich). Fündig geworden ist Atari dann beim VME-Bus, der ein weitverbreiteter Standardbus in der Industrie ist. Zudem ähnelt das Verhalten der Steuer- und Quittungssignale sehr dem der Signale der Motorola-CPUs (was Wunder, schließlich kam die Initiative zu diesem Bussystem ja von dort).

Also findet sich im TT/MEGA STE jetzt eine Anschlußmöglichkeit für Erweiterungskarten nach dem VME-Bus-Prinzip. Dazu werden von zwei 50pol. Pfostensteckverbindern auf der Hauptplatine die nötigen Signale über Flachbandkabel zu der beim VME-Bus verwendeten 96pol. Federleiste nach DIN 41612, Bauform C, geführt. Im Gehäuse des Rechners ist Platz (so gerade!) für eine Einfach-Europakarte (100mm x 160mm).

Das VME-Buskonzept

Der VME-Bus unterstützt bei Einfach-Europakartenformat einen 24-Bit-Adreß- und einen 16-Bit-Datenbus. Durch Übergang auf ein Doppel-Europakartenformat kann der Bus auf volle 32 Bit für den Adreß- und Datenbereich erweitert werden. Dabei werden über den dann zusätzlich erforderlichen Steckverbinder lediglich die noch fehlenden Adreß- und Datenbussignale geführt. Alle anderen Signale sind bereits auf der Einfach-Europakartenversion verfügbar. Der gesamte VME-Bus gliedert sich dabei in vier Teilsysteme:

- Der *Daten-Transfer-Bus* (DTB) verfügt über alle Daten- und Adreßleitungen sowie die für den Datentransfer erforderlichen Steuer- und Quittungsleitungen.
- Über den *Arbitrations-Bus* laufen alle für die Steuerung eines Multi-Master-Systems notwendigen Signale.
- Der *Interrupt-Bus* stellt die Signalwege für die Behandlung von Unterbrechungen zur Verfügung.

– *Versorgungs- und Hilfsleitungen* bilden die restlichen Signale zur Stromversorgung und Fehlererkennung.

Die Steckerbelegung des 96pol. VME-Bus-Steckers zeigt die nachfolgende Liste.

Alle mit einem "_" beginnenden Signale sind low-aktiv!

Pin-Nr.	Reihe a	Reihe b	Reihe c
1	D0	_BBSY	D8
2	D1	_BCLR	D9
3	D2	_ACFAIL	D10
4	D3	_BG0IN	D11
5	D4	_BG0OUT	D12
6	D5	_BG1IN	D13
7	D6	_BG1OUT	D14
8	D7	_BG2IN	D15
9	GND	_BG2OUT	GND
10	SYSCLK	_BG3IN	_SYSFAIL
11	GND	_BG3OUT	_BERR
12	_DS1	_BR0	_SYSRESET
13	_DS0	_BR1	_LWORD
14	_WRITE	_BR2	_AM5
15	GND	_BR3	A23
16	_DTACK	AM0	A22
17	GND	AM1	A21
18	_AS	AM2	A20
19	GND	AM3	A19
20	_IACK	GND	A18
21	_IACKIN	SERCLK	A17
22	_IACKOUT	SERDAT	A16
23	AM4	GND	A15
24	A7	_IRQ7	A14
25	A6	_IRQ6	A13
26	A5	_IRQ5	A12
27	A4	_IRQ4	A11
28	A3	_IRQ3	A10
29	A2	_IRQ2	A9
30	A1	_IRQ1	A8
31	–12V	+5V (Stand By)	+12V
32	+5V	+5V	+5V

Daten-Transfer-Bus (DTB)

Der DTB wird asynchron betrieben, um auch Devices mit unterschiedlichen Geschwindigkeiten am Bus betreiben zu können. Adreß- und Datenbus sind nicht gemultiplext.

Der Datentransfer zwischen Sender (Master) und Empfänger (Slave) läuft nach folgendem Schema ab:

- Die Gültigkeit einer Adreßinformation auf dem Bus signalisiert der Master durch Aktivierung von Address Strobe (_AS).
- Über das _WRITE-Signal erfährt der Slave, ob gelesen (_WRITE = High) oder geschrieben wird.

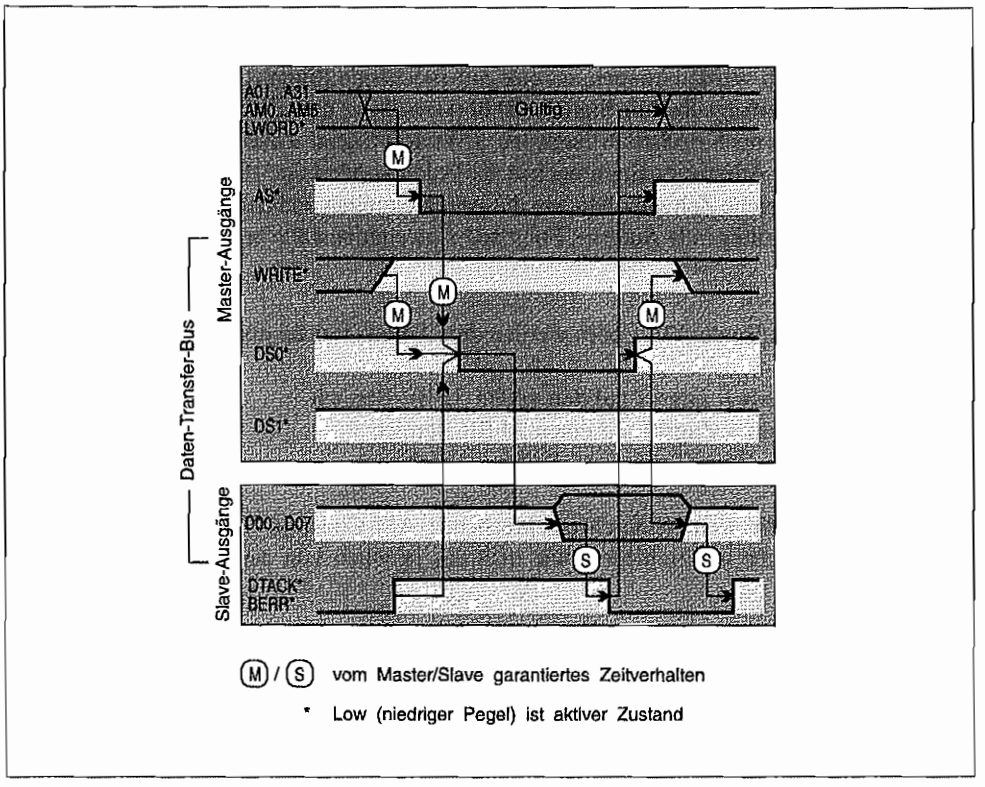


Abb. 8.1: Byte-Lesezyklus auf dem Daten-Transfer-Bus

- Bei einem *Lesezugriff* wird dem Slave über die beiden Data-Strobe-Leitungen _DS0 und _DS1 mitgeteilt, wenn der Master zum Einlesen der Daten bereit ist!

- Für einen *Schreibzugriff* werden vom Master zunächst die Daten auf den Datenbus plaziert und anschließend diese über _DS0 und _DS1 als "gültig" erklärt.

- Der Bus wird vom Master erst wieder freigegeben, wenn vom Slave ein _DTACK (Data Transfer Acknowledge) empfangen wurde. Der Slave "fühlt" sich so lange an den Master gebunden (und gibt den Bus so lange auch nicht frei!), wie der nicht _DS0 und _DS1 wieder deaktiviert hat.

- Wenn der Slave sich "nicht in der Lage sieht", die Schreib-/Leseanforderung korrekt auszuführen, aktiviert er statt _DTACK das Bus-Error-Signal (_BERR). Als Reaktion darauf wechselt der Master in eine entsprechende Fehlerbehandlungsroutine.

Da über den Datentransferbus sowohl Byte-, Word- (16 Bit) und auch Langwort-Zugriffe (32 Bit) abgewickelt werden können, ist eine entsprechende Signalisierung der Zugriffsart erforderlich. Diese erfolgt über die Signale _DS0 (Low-Byte) und _DS1 (High-Byte) und bei Langwort-Zugriffen noch zusätzlich durch das _LWORD-Signal. Zusätzlich zu den Adreßleitungen existieren zur Adreßverwaltung noch sechs "Address-Modifier"-Leitungen AM0..AM5. Diese zeigen das gleiche Zeitverhalten wie die Adreßleitungen und liefern zusätzliche Informationen über die anliegende Adresse. Folgende Anwendungsfälle lassen sich damit signalisieren:

- Durch entsprechende AM-Codes werden drei Adreßbereiche unterschieden:

  Extended Access = Zugriff auf den vollen 32 Bit-Bereich
  Standard Access = Zugriff auf einen 24 Bit-Bereich
  Short-Access = Zugriff auf einen 16 Bit-Bereich

- Speicherzugriffe werden durch entsprechende AM-Codes in Supervisor- und User-Speicherbereiche unterschieden, wie man das ja schon von den Motorola-CPUs her kennt.

- Unterscheidung des Speichers in Daten- und Programmbereich.

- Signalisierung, ob Speicher- oder E/A-Zugriff (Memory-Mapped I/O!) stattfindet.

- Spezielle Formen des Speicherzugriffs, indem vom Master nur die Beginnadresse des anzusprechenden Speicherbereichs geliefert wird und die darauf folgenden Adressen durch Inkrementieren der Adresse auf dem Slave selbst angesteuert werden.

Die nachfolgende Tabelle zeigt die Werte für die Address-Modifier und die damit ausgewählte Funktion.

AM-Code in Hex.	Binär	Funktion
3F	1 1 1 1 1 1	Standard-Supervisory-Zugriff, Inkrementell
3E	1 1 1 1 1 0	Standard-Supervisory-Zugriff, Programmbereich
3D	1 1 1 1 0 1	Standard-Supervisory-Zugriff, Datenbereich
3B	1 1 1 0 1 1	Standard-User-Zugriff, Inkrementell
3A	1 1 1 0 1 0	Standard-User-Zugriff, Programmbereich
39	1 1 1 0 0 1	Standard-User-Zugriff, Datenbereich
2D	1 0 1 1 0 1	Short-Supervisory-Zugriff, E/A-Bereich
29	1 0 1 0 0 1	Short-User-Zugriff, E/A-Bereich
0F	0 0 1 1 1 1	Extended-Supervisory-Zugriff, Inkrementell
0E	0 0 1 1 1 0	Extended-Supervisory-Zugriff, Programmbereich
0D	0 0 1 1 0 1	Extended-Supervisory-Zugriff, Datenbereich
0B	0 0 1 0 1 1	Extended-User-Zugriff, Inkrementell
0A	0 0 1 0 1 0	Extended-User-Zugriff, Programmbereich
09	0 0 1 0 0 1	Extended-User-Zugriff, Datenbereich

Die nicht aufgeführten AM-Codes sind reserviert!

Der Arbitrations-Bus

Damit in einem Multiprozessorsystem mehrere Master auf den DTB zugreifen können, muß eine gewisse Absprache zwischen den Mastern für die Busbenutzung erfolgen. Es wird also entschieden (= to arbitrate), welcher Master jeweils den Bus benutzen darf.

Dafür gibt es in Multiprozessorsystemen einen eigenen "Schiedsrichter", den Arbiter.

Die Anforderung des Busses nimmt ein Master über eine der vier Bus-Request-Leitungen (_BR0.._BR3) vor. Jeder dieser vier Request-Leitungen ist eine eigene Priorität zugeordnet (_BR3 ist höchste Priorität). Der Arbiter vergleicht die Priorität des anfordernden Masters mit der Priorität des Masters, der gerade den Bus kontrolliert.

Hat der anfordernde Master eine höhere Priorität, sendet der Arbiter ein Bus-Clear-Signal (_BCLR), um den gegenwärtigen Busmaster zur schnellstmöglichen Freigabe des Busses aufzufordern. Dieser reagiert darauf zum nächstmöglichen Zeitpunkt (üblicherweise nach Beendigung des gegenwärtigen Buszyklusses) durch Deaktivierung von Bus-Busy (_BBSY). Der Arbiter sendet daraufhin der Einheit mit der höchsten anstehenden Priorität ein Bus-Freigabe-Signal (Bus-Grant = _BG0.._BG3) und deaktiviert _BCLR.

Wenn der anfordernde Master niedriger priorisiert ist als der gegenwärtige Busmaster, wird die Busanforderung erst bedient, wenn der Bus frei ist!

Weil nach diesem Schema nur vier Busmaster vom Arbiter verwaltet werden können, hat man noch eine zusätzliche Prioritätsebene vorgesehen. Man verwendet hier das Prinzip des "Daisy-chaining", der Prioritätsverkettung. Das bedeutet in der Praxis, daß der physikalisch am nächsten zum Arbiter gelegene Master, mit ansonsten gleicher Priorität, "bevorzugter" behandelt wird! Mit wachsender Entfernung nimmt die "Bevorzugung" ab. Jede der vier Anforderungsebenen _BR0.._BR3 hat eine eigene "Daisy-Chain". In der Praxis sieht das dann so aus, daß das Busfreigabe-Signal _BG, welches dem Bus anfordernden Master die "Herrschaft" über den Bus zuweist, vom Arbiter zum _BG#IN-Anschluß des nächstgelegenen Masters geführt wird. Wenn dieser den Bus belegen will, gibt er das Busfreigabe-Signal _BG nicht weiter; andernfalls aktiviert er _BG#OUT, was dann zu _BG#IN des nächsten Masters führt usw. Mittels dieser Technik können beliebig viele Master auf dem VME-Bus zusammenarbeiten.

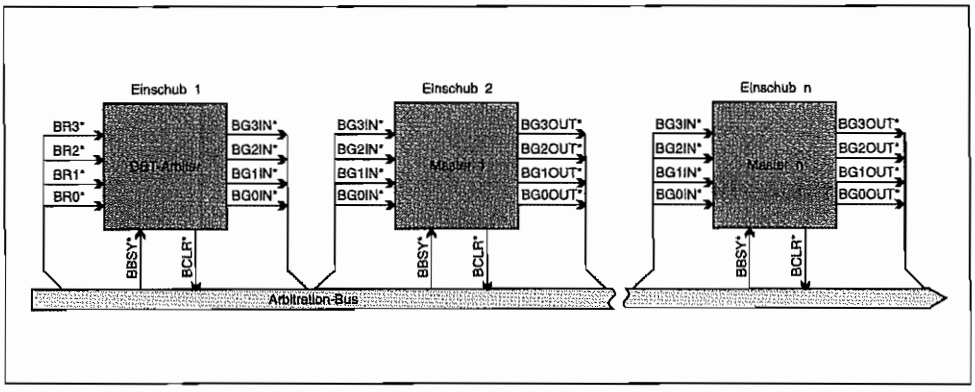


Abb. 8.2: Block-Diagramm zur Bus-Arbitration

Der Interrupt-Bus

Auch hier wird, wie beim Arbitrations-Bus, mit unterschiedlichen Prioritäten gearbeitet. Auf dem VME-Bus wird zur Auswahl der Interrupt-Service-Routine im Interrupt-Handler mit Interruptvektoren gearbeitet. Die *Interrupt-Anforderung* erfolgt über sieben Interrupt-Request-Leitungen _IRQ7.._IRQ1, wobei _IRQ7 die höchste Priorität zugeordnet ist. Mit entsprechender Maskierung (z. B. durch eine Interruptmaske bei MC680XX-Prozessoren) entscheidet der

Interrupthandler, ob der Interrupt zugelassen wird oder nicht. Um auch bei nur sieben IRQ-Ebenen eine beliebige Anzahl von Interruptquellen gewichten zu können, kann wieder mit der schon beim Arbitrations-Bus geschilderten "Daisy-Chain"-Technik gearbeitet werden. Dazu wird das Interrupt-Erkennungssignal des Interrupthandlers von Modul zu Modul weitergereicht (an _IACKIN rein und bei _IACKOUT wieder raus zum nächsten Modul!), bis es bei jenem Modul angekommen ist, dessen Interrupt-Anforderung bearbeitet werden soll.

Die *Interrupt-Erkennung* erfolgt durch den Interrupt-Handler unter Benutzung des DTB, den der Int.-Handler sich dazu evtl. vorher vom Arbiter "zuteilen" lassen muß! Wenn der Int.-Handler den DTB "hat", aktiviert er Interrupt-Acknowledge (_IACK) und legt den Interrupt-Anforderungskode (Priorität von 1..7) auf A1..A3 des Adreßbusses. Der nun folgende, normale Lesezyklus liefert dann von dem Modul, das den Interrupt angefordert hat, die Vektornummer.

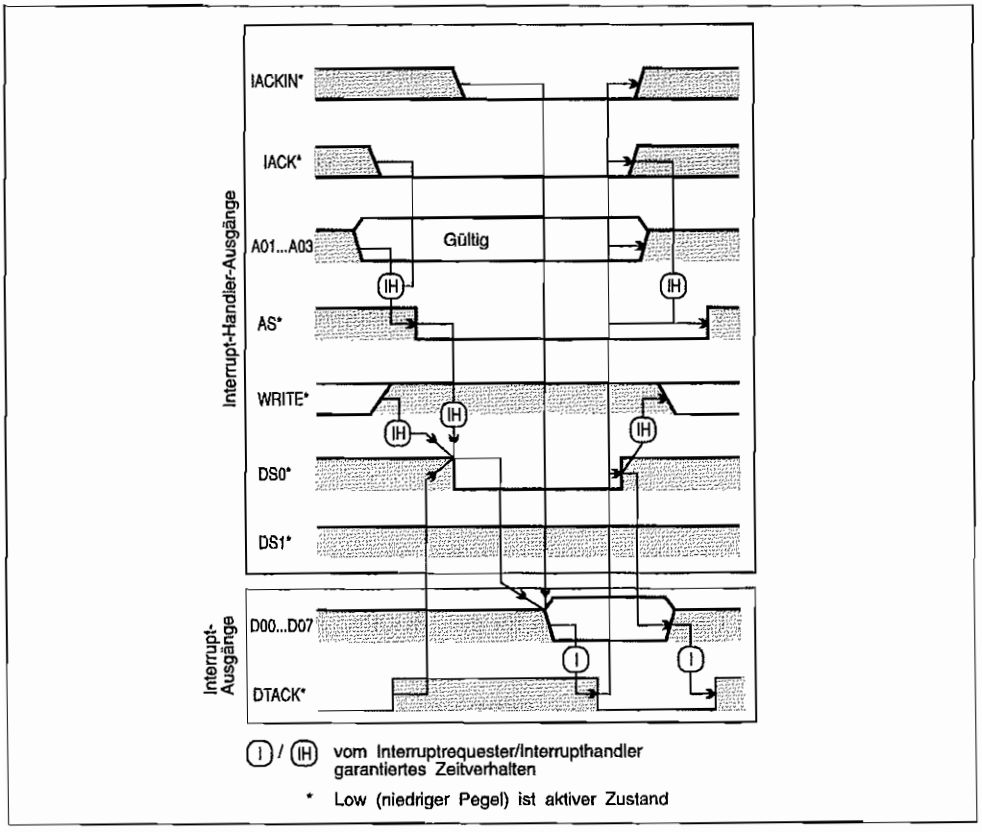


Abb. 8.3: Zeitverhalten der Interrupterkennung

Diese Vektornummer darf aber erst auf den Datenbus (Low-Byte, _DS0 aktiviert, _DS1 nicht!) ausgegeben werden, wenn das den Interrupt anfordernde Modul auch das Signal _IACK des Interrupt-Handlers an seinen _IACKIN-Anschluß über die "Daisy-Chain" empfangen hat. Die *Interrupt-Service-Routine* wird im Interrupt-Handler über die Vektornummer berechnet und enthält die zur Interrupt-Bearbeitung nötigen Befehle.

Die Hilfs- und Versorgungsleitungen

Neben den Versorgungsspannungen werden über diesen Teil des VME-Busses noch einige weitere Hilfssignale für die Initialisierung, Synchronisation und zur Fehlerdiagnose geführt.

SYSCLK ist ein 16 MHz-Takt, der nicht mit dem Prozessortakt synchron sein muß. Dieses Signal kann z. B. für Zähleranwendungen herangezogen werden.

_SYSRESET dient dem Rückstellen aller Systemmodule in einen definierten Anfangszustand. Es kann manuell oder vom Netzteil gesteuert aktiviert werden.

_SYSFAIL kann einen Fehler im System melden. So kann ein Modul nach einem Systemreset und einem durchgeführten (fehlerhaften) Selbsttest _SYSFAIL nicht wieder deaktivieren und so einen Fehlerzustand melden.

_ACFAIL signalisiert einen Fehler in der Spannungsversorgung (Spannungseinbruch). Damit bleibt dem System laut VME-Bus-Spezifikation noch mindestens 4ms für eine ordnungsgemäße Abbruch-Bearbeitung.

Einschränkungen beim VME-Busanschluß des TT/MEGA STE

ATARI verspricht Kompatibilität zum VME-Bus-Standard, jedoch mit einigen Ausnahmen.

- Bus-Arbitration wird *nicht* unterstützt! Alle Bus-Request-Signale _BR0.._BR3 sind auf dem Motherboard miteinander verbunden und mit einem 1KΩ Pull-up-Widerstand an +5V "gebunden". Gleiches gilt auch für die _BG0IN.._BG3IN-Signale! _BBSY und _BCLR liegen ebenfalls jeweils über 1KΩ an +5V und werden anderweitig nicht benutzt.
- Der serielle Bus mit SERCLK und _SERDAT ist nicht vorhanden.
- _SYSCLK wird mit dem 16,107953 MHz-Systemtakt gespeist.
- _ACFAIL ist low, solange die Spannungsversorgung nach dem Einschalten noch nicht stabil läuft. ATARI verspricht 1ms vor dem Verlassen der gültigen Spannungsbereiche eine Aktivierung von _ACFAIL. Das Signal ist mit 1KΩ gegen +5V geschaltet.

- _SYSRESET ist direkt mit dem RESET-Signal des Motherboards verbunden (1,2KΩ-Pull-up-Widerstand gegen +5V). Es kann sowohl vom Motherboard (z. B. CPU) als auch von einer VME-Bus-Karte aktiviert werden.

- _BERR ist ebenfalls direkt mit dem Bus-Error-Signal des Motherboards verbunden (1,2KΩ-Pull-up-Widerstand gegen +5V). "Sieht" die Überwachungsschaltung (in der SCU) auf dem Motherboard 255 Taktzyklen (16 MHz-Takt!) nach Aktivierung von _AS kein _DTACK, wird ein Bus-Error ausgelöst!

- ATARI verwendet aktiv nur die Address-Modifier AM0, AM1, AM2 und AM4. AM3 und AM5 liegen über 1KΩ-Widerstände an +5V. Damit ist alles bis auf Extended-Zugriffe möglich. Inkrementelle Zugriffe (Blockzugriffe) werden aber nicht unterstützt! Da nur ein 16 Bit-Datenbus am ATARI-VME-Bus zur Verfügung steht, wird das _LWORD-Signal nicht benötigt. Es ist mit 1KΩ gegen +5V abgeschlossen.

  Im TT030 findet sich für Standard-Zugriffe (24 Bit!) der VME-Bus-Adreßraum bei $FE00 0000..$FEFE FFFF. Für Short-Zugriffe liegt der Adreßbereich bei $FEFF 0000.. $FEFF FFFF. Im MEGA STE liegt der 24 Bit-Adreßraum bei $ A0 0000 bis $DE FFFF. Short-Zugriffe "landen" im MEGA STE bei $DF 0000..$DF FFFF.

- Die Interrupt-Request Signale _IRQ1.._IRQ7 sind auf dem Motherboard mit je einem 1KΩ-Widerstand gegen +5V abgeschlossen und können verwendet werden. _IRQ3, _IRQ5 und _IRQ6 können auch vom Motherboard aktiviert werden. Ein Level-7-Interrupt kann durch das _SYSFAIL-Signal (mit 1KΩ-Pull-up auf Motherboard abgeschlossen!) einer VME-Bus-Karte ausgelöst werden. _IACK und _IACKIN werden vom Motherboard gesteuert und sollten deshalb nicht von einer Karte angesteuert werden (TT/MEGA STE ist Interrupthandler!). Das _IACKOUT-Signal wird nicht verwendet.

  Der Interrupt-Vektor, welcher während des Interrupt-Acknowledge-Zyklus von einer VME-Bus-Karte geliefert wird, ist gleichzeitig Interrupt-Vektor für die System-CPU! Atari hat den Vektor $FF für sich reserviert! Für Drittanbieter empfiehlt Atari die Interrupt-Vektoren von $80..$BF und damit die Vektor-Adressen $200..$2FF. Alle VME-Bus- und System-Interrupts sind unabhängig voneinander in der *System Control Unit* (SCU) maskierbar.

Dann gilt noch für alle Ausgänge, daß sie mindestens 1 LS-TTL-Eingänge treiben können (was nicht das meiste ist!). Eingänge am TT/MEGA STE belasten den VME-Bus mit nicht mehr als 2 LS-TTL-Eingangslasten. Die Stromversorgungsanschlüsse des VME-Bus-Anschlusses haben folgende "Reserven":

+5V bei max. 2A
+12V bei max. 50mA
–12V bei max. 50mA

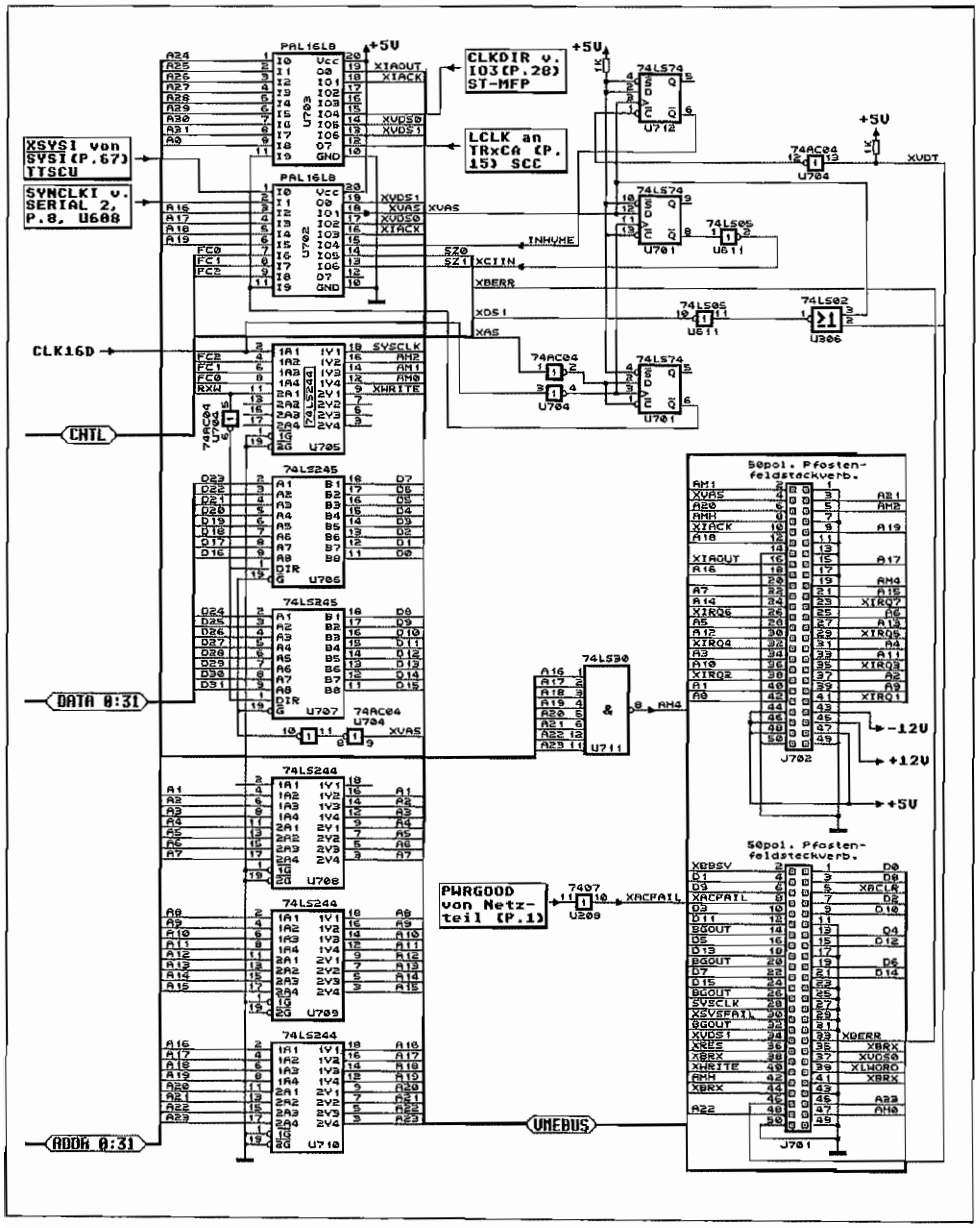


Abb. 8.4: Die hardwaremäßige Realisierung des VME-Busses im TT/MEGA STE

Kapitel 9: Die System Control Unit im TT/MEGA STE

Zur Überwachung des Systems auf Bus-Errors/Interrupts und zur Erzeugung einiger Chip-Select-Signale (MFPs und FPU) hat ATARI im TT/MEGA STE einen eigenen Custom-Chip eingesetzt, die System Control Unit (SCU). Die wichtigste Funktion der SCU im System dürfte wohl die Verwaltung der Interrupts sein. Im Gegensatz zum ST existieren ja bei den neuen Maschinen mit VME-Bus-Anschluß einige Interruptquellen mehr, die verwaltet werden müssen. Außerdem verfügt die SCU noch über einige Register, durch deren Beschreiben Interrupts softwaregesteuert ausgelöst werden können.

Interrupts unter Kontrolle

Die SCU besitzt zwei voneinander unabhängige Mask-Register, über die einstellbar ist, welcher Interrupt-Level überhaupt bis zur CPU durchdringt.

Systemboard-Interrupts

Dabei maskiert das "sys_mask"-Register die Interrupt-Anforderungen vom Systemboard. Dazu gehört ein "sys_stat"-Register, in dem die Interrupt-Anforderungen vom Systemboard "bereitgehalten" werden. Die Bits dieses Registers zeigen den jeweiligen Zustand der sieben möglichen Interrupt-Anforderungen an. An dieser Stelle ist die Maskierung noch nicht wirksam, d. h. alle Interruptanforderungen setzen ihr zugehöriges Bit!

Adresse	Zugriff	Funktion	Label
$FF 8E01	R/W	System Int. Mask-Register	"sys_mask"

Das Bit 0 des Registers wird nicht verwendet. Die Bits 7..1 bilden die Maske für die Systeminterrupts. Ein gesetztes Bit zeigt an, daß der zugehörige Interrupt entsprechender Priorität an die CPU weitergereicht wird. Bei gelöschtem Bit unterbleibt eine Interruptanforderung an die CPU. Folgende Systeminterruptquellen sind vorgesehen:

Int.-Level	Interrupt-Quelle	Interrupt-Vektor
7	_SYSFAIL von VME-Bus	Autovektor
6	IRQ von MFPs	Programmierbar
5	IRQ von SCC	Programmierbar

Int.-Level	Interrupt-Quelle	Interrupt-Vektor
4	VSYNC	Autovektor
3	Nicht vorhanden	–
2	HSYNC	Autovektor
1	System Software-Interrupt	Autovektor

Im TOS 3.01 wird das "sys_mask"-Register mit dem Wert $14 initialisiert. Also sind alle Interrupt-Requests bis auf VSYNC und HSYNC maskiert!

Adresse	Zugriff	Funktion	Label
$FF 8E03	R	System Int. Status-Register	"sys_stat"

Auch hier wird das Bit 0 nicht verwendet. Die anderen Bits stellen direkt den jeweiligen Status der IRQ-Anforderungen dar (gesetztes Bit = IRQ liegt vor), ohne die Maskierung durch das "sys_mask"-Register zu berücksichtigen. Dabei gilt die gleiche Zuordnung, wie bereits beim "sys_mask"-Register gezeigt. Dieses Register sollte *vor* einem Zugriff auf das "sys_mask"-Register ausgelesen werden, da ein Zugriff auf das Mask-Register die Interrupt-Requests im "sys_stat"-Register zurücksetzt!

VME-Bus-Interrupts

Für Interrupts vom VME-Bus existiert so ein Registersatz ebenfalls ("vme_mask" und "vme_stat"). Dabei ist allerdings zu beachten, daß die System-Interruptanforderungen 5 und 6 auch auf die entsprechenden IRQ-Leitungen des VME-Busses aufgeschaltet sind und somit noch zusätzlich als Interrupt-Requests im "vme_stat"-Register anstehen!

Damit wird erreicht, daß ein entsprechender VME-Bus-Master diese Systeminterrupts ebenfalls "bemerkt" und abarbeiten kann! Es bedeutet aber auch, daß diese System-Interrupts für die CPU wie VME-Bus-Interrupts "aussehen" und nicht durch ausschließliches Maskieren im "sys_mask"-Register "unwirksam" gemacht werden können.

Ferner läßt sich vom System softwaremäßig ein Interrupt erzeugen, der als _IRQ3 auf dem VME-Bus erscheint. Dieser Interrupt-Request wird natürlich ebenfalls im "vme_stat"-Register "bemerkt" und kann im "vme_mask"-Register maskiert werden!

Adresse	Zugriff	Funktion	Label
$FF 8E0D	R/W	VME-Bus Int. Mask-Register	"vme_mask"

Das Bit 0 des Registers wird wieder nicht verwendet. Die Bits 7..1 bilden die Maske für die VME-Bus-Interrupts. Ein gesetztes Bit zeigt an, daß der zugehörige Interrupt entsprechender Priorität an die CPU weitergereicht wird.

Bei gelöschtem Bit unterbleibt eine Interruptanforderung an die CPU.

Folgende VME-Bus-Interruptquellen sind vorgesehen:

Int.-Level	Interrupt-Quelle	Interrupt-Vektor
7	_IRQ7 von VME-Bus	Programmierbar
6	_IRQ6 von VME-Bus bzw. IRQ von MFPs	Programmierbar
5	_IRQ5 von VME-Bus bzw. IRQ von SCC	Programmierbar
4	_IRQ4 von VME-Bus	Programmierbar
3	_IRQ3 von VME-Bus bzw. durch Software	Progr./Autovek.
2	_IRQ2 von VME-Bus	Programmierbar
1	_IRQ1 von VME-Bus	Programmierbar

Im TOS 3.01 wird dieses Register mit $60 initialisiert. Damit sind also nur der _IRQ6 (MFP-IRQ) und der _IRQ5 (SCC-IRQ) freigegeben!

Adresse	Zugriff	Funktion	Label
$FF 8E0F	R	VME-Bus Int. Status-Register	"vme_stat"

Bit 0 ist wieder nicht verwendet. Die anderen Bits stellen wieder den jeweiligen Status der IRQ-Anforderungen dar, ohne eine evtl. Maskierung durch das "vme_mask"-Register zu berücksichtigen. Es gilt die gleiche Zuordnung, wie beim "vme_mask"-Register gezeigt. Dieses Register sollte *vor* einem Zugriff auf das "vme_mask"-Register ausgelesen werden, da ein Zugriff auf das Mask-Register die Interrupt-Requests im "vme_stat"-Register zurücksetzt!

Software-Interrupts durch die SCU

Die SCU besitzt zwei Register, durch deren Beschreiben ein Interrupt ausgelöst werden kann. Beide Interrupts werden als Autovektor-Interrupts entsprechender Priorität behandelt.

Adresse	Zugriff	Funktion	Label
$FF 8E05	R/W	System-Software-Int. erzeugen	"sys_int"

Setzt man durch Einschreiben in dieses Register das Bit 0, wird ein System-Interrupt-Request der Priorität 1 ausgelöst. Dieser gelangt natürlich nur zur CPU, wenn er nicht in "sys_mask" ausmaskiert ist. Der Pointer auf die Autovektor-Serviceroutine für diesen Interrupt steht im RAM in Adresse $64.

Adresse	Zugriff	Funktion	Label
$FF 8E07	R/W	VME-Bus-IRQ Level 3 erzeugen	"vme_int"

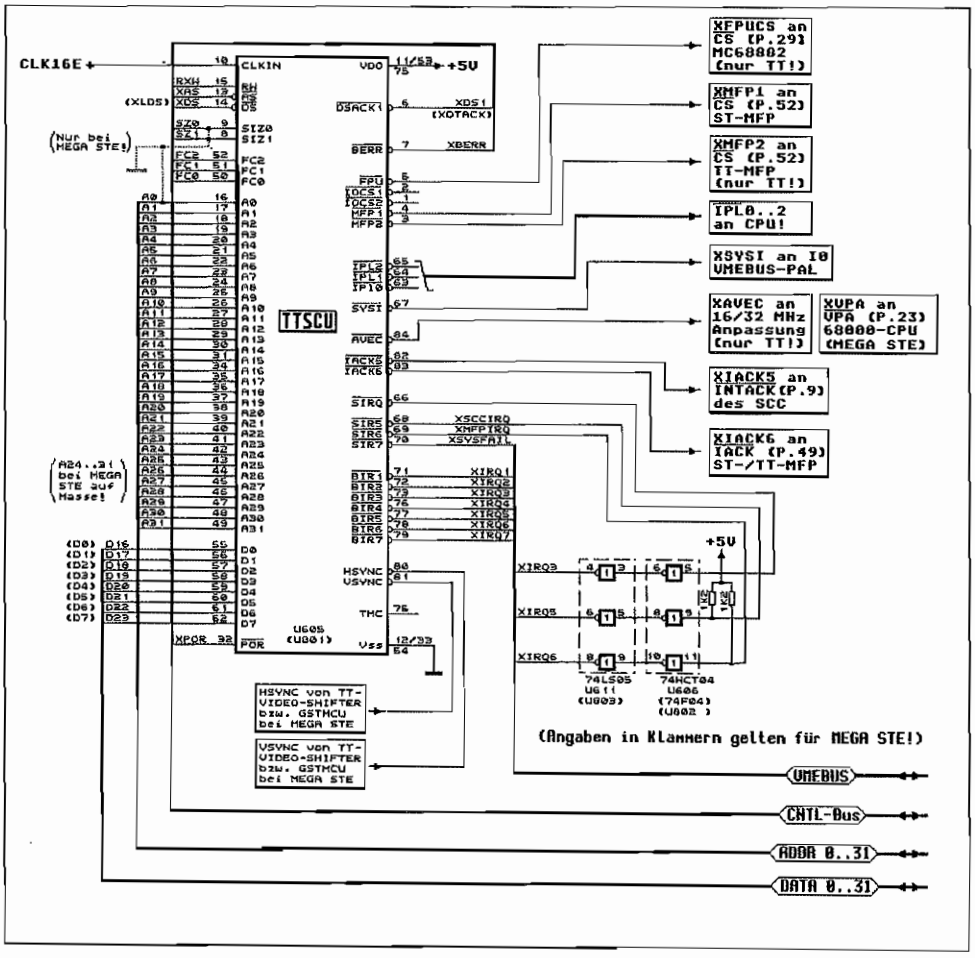


Abb. 9.1: Die Einbindung der SCU im TT/MEGA STE

Durch Einschreiben einer $01 in dieses Register wird ein System-Interrupt-Request der Priorität 3 auf dem VME-Bus ausgelöst. Dazu muß aber im "sys_mask"-Register der IRQ 3 freigegeben sein! Der Pointer auf die zugehörige Autovektor-Serviceroutine muß dann im RAM in Adresse $6C stehen.

Generell gilt für die Interrupt-Behandlung durch die SCU, daß bei Interrupt-Requests gleicher Priorität vom Systemboard *und* vom VME-Bus der Systeminterrupt höher priorisiert ist!

Sonstige Register in der SCU

Die SCU verfügt noch über zwei Universal-Register, die z. B. zur Speicherung von Konfigurationswerten für das System benutzt werden können. Diese Register sind 8 Bit breit und an folgenden Adressen zu finden:

Adresse	Zugriff	Funktion	Label
$FF 8E09	R/W	SCU General Purpose Reg. 1	"scu_gp1"
$FF 8E0B	R/W	SCU General Purpose Reg. 2	"scu_gp2"

Bei einem System-RESET werden alle SCU-Register gelöscht!

Kapitel 10: Der Uhrenchip im TT

Im TT verwendet ATARI einen anderen Uhrenchip als im MEGA ST/MEGA STE. Bei diesem Uhrenchip sind zusätzlich zu den Zeit- und Datumsfunktionen noch 50 Bytes an batteriegepuffertem RAM nutzbar. Darin lassen sich zum Beispiel Initialisierungswerte ablegen, die auch nach dem Ausschalten des Systems nicht vergessen werden. Zur "Verwaltung" dieses "alternativen Speicherbereichs" hat ATARI sogar eigene Betriebssystemaufrufe bereitgestellt.

Verwendet wird der Motorola-Uhrenchip MC146818A, der sich sowohl in Mo-torola- als auch in Intel-Prozessor-Umgebungen betreiben läßt. Die Betriebsart wird über den Pin 1 (MOT) eingestellt.

Eine 6V-Batterie sorgt für das "Überleben" der Daten bei ausgeschaltetem Rechner. Damit der Takt für den "Wecker" bei ausgeschaltetem Rechner nicht stehenbleibt, wird ein eigener Oszillator (auch durch Batterie gepuffert) mit einer Frequenz von 32,768 kHz benutzt.

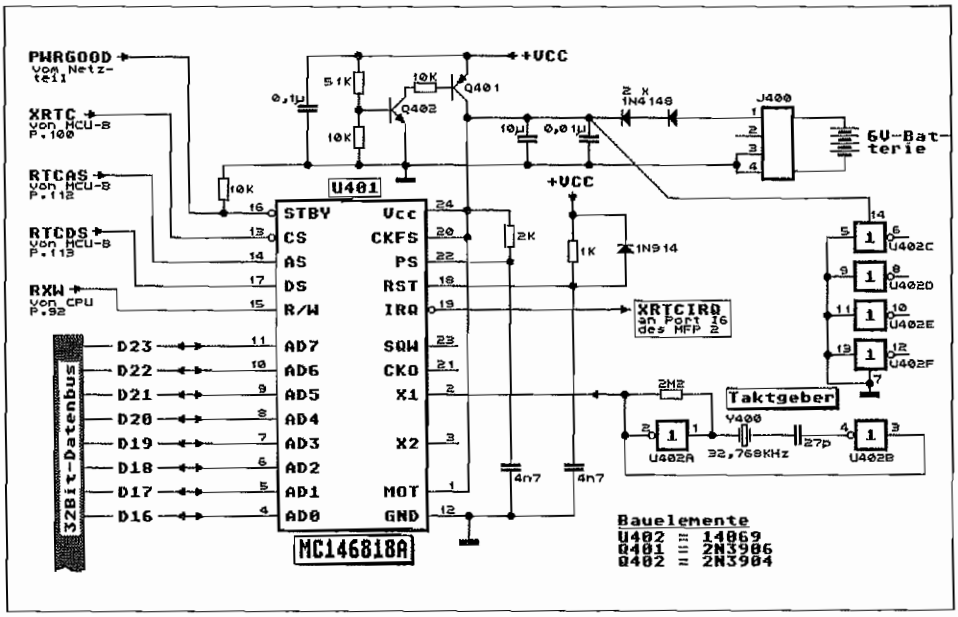


Abb. 10.1: Die hardwaremäßige Einbindung des TT-Uhrenchips

Ähnlich dem "alten" Uhrenchip z. B. im MEGA ST besitzt auch dieser Chip einen Rechtecksignal-Ausgang, bei dem die Ausgangsfrequenz softwaremäßig eingestellt werden kann. Aber auch im TT verwendet Atari diesen Ausgang nicht. Der Anschluß aber, der diesmal benutzt wird, ist der _IRQ-Anschluß des Chips, der unter bestimmten Bedingungen Interrupts auslösen kann (z. B. "Weckzeit erreicht"!). Im TT läuft der Low-Aktive Interruptausgang des Uhrenchips an Port I6 des TT-MFP auf und kann abgefragt werden oder einen Interrupt auslösen.

Zugriffe auf das RAM und die Uhrenregister erfolgen als Bytezugriffe ähnlich wie beim Soundchip in zwei Stufen. Zuerst wird das gewünschte Register durch Einschreiben der Registernummer in "rtc_rnr" ausgewählt. Anschließend kann über "rtc_data" das zuvor adressierte Register angesprochen werden. Nachfolgend die Adressen der beiden Uhrenregister (angegeben ist nur die Lage im ST-Adreßraum; weiter "oben", also bei $FFFF 896X, sind sie noch einmal zu finden!):

Adresse	Zugriff	Funktion	Label
$FF 8961	R/W	Registerauswahl im Uhrenchip	"rtc_rnr"
$FF 8963	R/W	Daten des selekt. Uhrenregisters	"rtc_data"

Die Register des Uhrenchips

Neben 50 Bytes an batteriegepuffertem RAM existieren noch 14 weitere Uhrenregister zur Darstellung der Uhrzeit/Datums und für z. B. Alarmzeit/Statusbits. Die Registerbelegung des Chips zeigt die Abbildung 10.2.

Die Zeit- und Datumsregister

Dabei gilt sowohl für die Uhrzeit/Datumsregister als auch für die Alarmzeitregister, daß die Werte im Binärformat oder im BCD-Format verwendet werden können. Die Einstellung, welches Format benutzt wird, ist über das "DM"-Bit (Bit 2) in Register $0B vorzunehmen (Gelöscht = BCD-Format). Das Datenformat kann nicht während des Betriebs geändert werden, ohne eine Neuinitialisierung der Uhrenregister vorzunehmen! Das TOS arbeitet mit dem Binärformat.

Außerdem läßt sich über das "24/12"-Bit (Bit 1) im Register $0B festlegen, ob die Zeit im 24-Stunden oder 12-Stunden-Format mit AM/PM-Signalisierung dargestellt wird. Wenn das "24/12"-Bit gesetzt ist, wird der 24-Stunden-Modus benutzt, den auch das TOS verwendet. Falls mit 12-Stunden-Darstellung gearbeitet wird, erfolgt die AM/PM-Unterscheidung über das

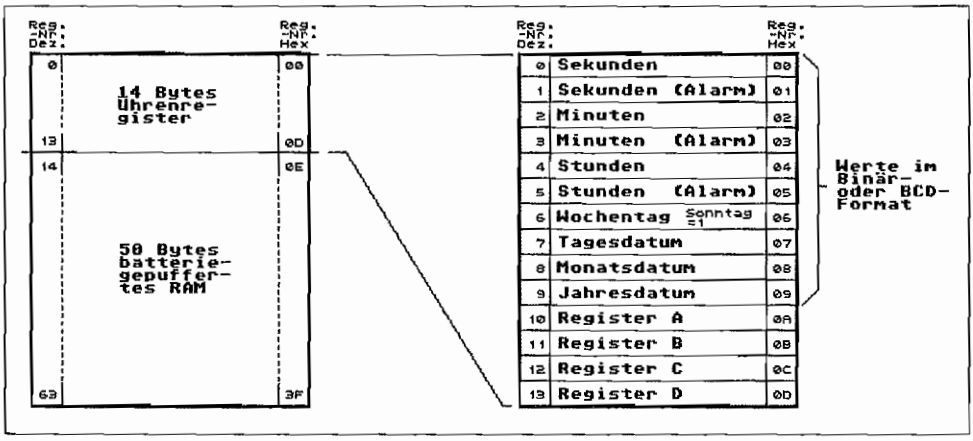


Abb. 10.2: Die Register des TT-Uhrenchips

höchstwertige Bit der Stundenregister (Zeit und Alarmzeit!). PM-Zeiten (= Nachmittagszeiten) werden durch ein gesetztes Bit 7 in den Stundenregistern (Zeit als auch Alarmzeit!) kenntlich gemacht!

Alarmzeitregister

Mit Hilfe der drei Alarmzeitregister kann eingestellt werden, wann eine Alarmsignalisierung (evtl. auch per Interrupt) erfolgen soll. Sobald Alarmzeit und aktuelle Zeit übereinstimmen, wird ein Alarmflag ("AF"-Bit = Bit 5 in Register $0C) gesetzt und abhängig von einem Enable-Bit ("AIE"-Bit = Bit 5 in Register $0B) ein Interrupt ausgelöst. Damit läßt sich eine tägliche Alarmsignalisierung erreichen.

Über bestimmte "Don't care"-Werte (alle Hex-Werte von $C0..$FF sind solche "Don't care"-Codes!) in den Alarmzeit-Registern lassen sich aber z. B. auch stündliche, minütliche oder sogar sekundliche Alarmsignalisierungen erreichen. Eine "Don't care"-Kombination bei den Alarmzeit-Stunden erzeugt jede Stunde eine Alarmierung. Bei "Don't care" in Alarmzeit-Stunden und -Minuten "kommt" der Alarm jede Minute. Wenn alle drei Alarmzeit-Register mit "Don't care"-Codes versehen sind, erfolgt jede Sekunde eine Alarmierung.

Die Kontrollregister des Uhrenchips

Mit Hilfe von vier Kontrollregistern ($0A..$0D) läßt sich das Verhalten des Uhrenchips beeinflussen und auswerten. Im Gegensatz zu den Registern $00..$09 sind diese vier Register

auch während des sogenannten "Update-Zyklus" erreichbar. Während des jede Sekunde einmal stattfindenden Update-Zyklus erfolgt im Uhrenchip die "Weiterschaltung" der Zeit- und Datumsregister und deren Vergleich mit den Alarmzeitregistern.

Während dieser Zeit (maximal 2ms) sind die Register $00..$09 des Uhrenchips intern abgekoppelt, und ein Lesezugriff darauf liefert keine korrekten Werte!

Um den richtigen Zeitpunkt für einen Zugriff auf die unteren zehn Uhrenregister zu "erwischen", gibt es mehrere Möglichkeiten.

- Am Ende eines jeden Updates kann ein Interrupt ausgelöst werden. Wenn dieser "Update ended"-Interrupt aufgetreten ist, hat man anschließend genug Zeit (min. 998ms!), um die gewünschten Register anzusprechen.

- Eine weitere Möglichkeit liegt in der Abfrage des "Update in Progress"-Bits (UIP-Bit = Bit 7 in Register $0A). Bei gesetztem Bit sollten die Zeit-/Datums-Register "in Ruhe gelassen werden".

  Wenn das Bit einen Wert von Null aufweist, hat man mindestens noch 244µs "Zeit" bis zum nächsten Update-Zyklus.

Register $0A

Die Bitbelegung dieses Registers und deren Bedeutung zeigt nachfolgende Tabelle:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
UIP	DV2	DV1	DV0	RS3	RS2	RS1	RS0

UIP Update in Progress

Ein gesetztes Bit zeigt einen laufenden oder kurz bevorstehenden Update-Zyklus der Zeit-/Datumsregister an, während dem ein Zugriff auf diese Register keine korrekten Werte liefert.

Bei gelöschtem Bit sind noch mindestens 244µs Zeit bis zum nächsten Update. Dieses Bit kann nur gelesen werden!

DV2, DV1, DV0 Divider-Steuerbits

Diese drei Bits werden dem Takt des Uhrenchips entsprechend festgelegt. Davon ist nämlich die Programmierung der Teilerstufen für die Erzeugung des Sekundentakts abhängig.

Taktfrequenz	DV2	DV1	DV0	Teiler in Betrieb	Teiler Reset
4,194304 MHz	0	0	0	Ja	–
1,048576 MHz	0	0	1	Ja	–
32,768 kHz	0	1	0	Ja	–
Egal	1	1	0	Nein	Ja
Egal	1	1	1	Nein	Ja

Andere Bitkombinationen sind nicht zulässig!

Da Atari mit einem 32,768 kHz-Takt arbeitet, werden die DV-Bits auf "010" gesetzt.

RS3, RS2, RS1, RS0 Rate-Select-Bits

Über diese vier Bits läßt sich programmieren, in welchen zeitlichen Abständen ein periodischer Interrupt ausgelöst werden kann.

Außerdem wird dadurch gleichzeitig die Ausgangsfrequenz des Rechtecksignals am SQW-Ausgang (im TT nicht verwendet!) eingestellt.

Betrachtet man die vier Rate-Select-Bits als eine vierstellige Binärzahl mit RS3 als höchstwertigem Bit, so berechnen sich die Ausgangsfrequenz für den SQW-Ausgang und die Zeitabstände zwischen zwei periodischen Interrupts wie folgt.

Mit RS = 8 x RS3 + 4 x RS2 + 2 x RS1 + RS0 ergibt sich:

$$\text{SQW-Freq.} = \frac{32768\ \text{Hz}}{2^{(RS-1)}}$$

$$\text{INT-Zeit} = \frac{1}{\text{SQW-Frequenz}}$$

Register $0B

Alle Bits dieses Registers können beschrieben und ihr aktueller Zustand ausgelesen werden.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SET	PIE	AIE	UIE	SQWE	DM	24/12	DSE

SET — Bei *gelöschtem Bit* arbeitet der Uhrenchip in gewohnter Weise mit einem Update-Zyklus pro Sekunde. Durch *Setzen des Bits* wird ein evtl. Update-Zyklus unterbrochen oder gar nicht erst zugelassen, um in Ruhe eine Initialisierung der Zeit-/Datum-Register vornehmen zu können.

PIE (Periodic Int. Enable) — Bei gesetztem Bit werden periodische Interrupts in einem zeitlichen Abstand, der durch die RS3..RS0-Bits bestimmt wird, ausgelöst. Zur Signalisierung, daß ein periodischer Interrupt aufgetreten ist, wird im Register $0C durch ein gesetztes PF-Bit angezeigt.

AIE (Alarm Int. Enable) — Die Funktion dieses Bits ist ähnlich dem vorgenannten PIE-Bit. Bei gesetztem Bit wird ein Interrupt ausgelöst, wenn die aktuelle Zeit die Alarmzeit erreicht (unter Berücksichtigung von "Don't care"-Werten in den Alarmzeitregistern). Die Signalisierung, daß der Interrupt ein Alarmzeit-Interrupt ist, erfolgt über ein gesetztes AF-Bit im Register $0C.

UIE (Update ended Interrupt Enable) — Freigabebit für den Update-ended Interrupt (Bit gesetzt = Interrupt ist freigegeben). Die Interrupt-Service-Routine erkennt dann durch ein gesetztes UF-Bit in Register $0C, daß die Bedingung "Update beendet" die Quelle dieses Interrupts ist.

SQWE (Square-Wave-Output Enable) — Mit gesetztem Bit wird der SQW-Ausgang des Uhrenchips freigegeben und liefert ein Rechtecksignal der Frequenz, die mit den RS0..RS3-Bits in Register $0A eingestellt ist. Bei gelöschtem Bit ist der SQW-Ausgang auf Low.

DM (Data Mode) — Bei gesetztem Bit wird in den Zeit- und Datumsregistern mit Binärwerten gearbeitet. Bei gelöschtem Bit erwartet und liefert

der Uhrenchip die Werte im BCD-Format. Ein zwischenzeitlicher Moduswechsel verlangt auch immer eine Neu-Initialisierung des Uhrenchips, da dieser keine eigene Umrechnungsfunktion besitzt.

24/12 Wenn das Bit gesetzt ist, wird im 24-Stunden-Modus gearbeitet, sonst im 12-Stunden-Modus. Die Anzeige für "Nachmittagszeit" (PM-Anzeige) erfolgt durch ein gesetztes höchstwertiges Bit in den Stundenregistern.

DSE (Daylights Savings Enable)
Hiermit kann der Uhrenchip "angewiesen" werden (durch ein gesetztes Bit!), Sommer- und Winterzeit automatisch umzuschalten. Wenn diese Funktion eingeschaltet ist, springt die Uhrzeit am letzten Aprilsonntag von 01:59:59 auf 03:00:00. Entsprechend wird am letzten Sonntag im Oktober von 01:59:59 auf 01:00:00 gewechselt!

Register $0C

Alle Bits dieses Registers sind Statusbits und können nur ausgelesen werden. Nach einem Lesezugriff werden die Flags wieder zurückgesetzt!

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
IRQF	PF	AF	UF	0	0	0	0

IRQF Int. (Request Flag) Dieses Bit wird gesetzt, sobald einer der freigegebenen Interrupts auftritt.

PF (Periodic Int. Flag) Abhängig von der Einstellung der RS0..RS3-Bits im Register $0A wird in bestimmten zeitlichen Abständen dieses Bit gesetzt. Wenn der zugehörige Interrupt freigegeben ist (d.h. Bit PIE im Register $0B gesetzt), wird dazu ein Interrupt ausgelöst, der durch ein gesetztes IRQF-Bit in diesem Register angezeigt wird.

AF (Alarm Int. Flag) Ein gesetztes AF-Bit zeigt an, daß die aktuelle Zeit die eingestellte Alarmzeit (unter Berücksichtigung von evtl. "Don't care"-Einstellungen in den Alarmzeitregistern!) erreicht hat. Wenn das zugehörige AIE-Bit im Register $0B ebenfalls ge-

setzt ist, wird auch ein Interrupt ausgelöst, der dann auch durch ein gesetztes IRQF-Bit signalisiert wird.

UF (Update ended Int. Flag) Nach jedem Update-Zyklus wird dieses Bit gesetzt. Wenn der zugehörige Interrupt freigegeben ist (= Bit UIE in Register $0B gesetzt), erfolgt ebenfalls ein Interrupt, der zusätzlich über ein gesetztes IRQF-Bit angezeigt wird.

Register $0D

Nur das Bit 7 dieses Registers ist belegt und dient als Statusbit. Das Register kann nur ausgelesen werden.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
VRT	0	0	0	0	0	0	0

VRT (Valid RAM and Time) Dieses Bit zeigt die Gültigkeit der Daten im Uhrenchip an. Es ist auf Null, wenn der PS-Anschluß auf Low liegt (bei ausgeschaltetem TT und kurz nach dem Einschalten der Fall). Der Prozessor kann dieses Bit durch einen Lesezugriff auf dieses Register setzen, wenn Datum/Zeit und RAM-Inhalte korrekt initialisiert wurden. Durch Testen dieses Bits nach dem Einschalten des Systems kann der Prozessor feststellen, ob die Daten noch gültig sind.

Die Programmierung des TT-Uhrenchips

Als Beispiel für die Programmierung des Uhrenchips des TT soll das nachfolgende Listing dienen. Es handelt sich dabei um disassemblierte Routinen aus dem Original TOS 3.01. Die ausgeführten Aktionen sind jeweils kommentiert und zeigen letztlichdas Setzen der Zeit-/Datumswerte im Uhrenchip mit Ausgangswerten im GEMDOS-Zeit-/Datumsformat.

```
;=============================================================
; Folgende Routinen stammen aus dem TOS 3.01 und demonstrieren, wie
; Atari die Initialisierung (das Stellen) des Uhrenchips durchführt
;=============================================================
; Parameterübergabe für RTC-Chip als Long auf dem Stack
; [bei 4(a7)] im GEMDOS-Format!
;
```

```
setrtc:
        bsr     chkrtc          ; Zugriff auf RTC versuchen
        bcs     seterr          ; RTC vorhanden?
        move.l  4(a7),d0        ; Ja! Parameter nach d0 holen
        move.b  #$B,rtc_rnr     ; In Register B SET-Bit auf 1, um

        move.b  #$80,rtc_data   ; Update-cycle zu unterbrechen!
        move.b  #$A,rtc_rnr     ; In Reg. A auf 32 kHz-Timebase
        move.b  #$2A,rtc_data   ; und Period-Int. auf 15,625 ms
        move.b  #$B,rtc_rnr     ; In B-Reg. SQWE-, DM-, und
        move.b  #$8E,rtc_data   ; 24-hour-Bit setzen.
        move.b  #0,rtc_rnr      ; Reg. 0 ansteuern
        BFEXTU  d0{27:5},d1     ; Sekunden-Wert (Bits 0..4) aus
        add.b   d1,d1           ; Parameterdaten verdopp. und
        move.b  d1,rtc_data     ; in SEKUNDEN-Reg. eintragen
        move.b  #2,rtc_rnr      ; Minuten-Wert (Bits 5..10) aus
        BFEXTU  d0{21:6},d1     ; Parameterdaten holen und
        move.b  d1,rtc_data     ; in MINUTEN-Reg. eintragen.

        move.b  #4,rtc_rnr      ; Stunden-Wert (Bits 11..15) aus
        BFEXTU  d0{16:5},d1     ; Parameterdaten holen und
        move.b  d1,rtc_data     ; in STUNDEN-Reg. eintragen.
        move.b  #7,rtc_rnr      ; Tagesdatum (Bits 16..20) aus
        BFEXTU  d0{11:5},d1     ; Parameterdaten holen und
        move.b  d1,rtc_data     ; in TAGESDATUM-Reg. eintragen
        move.b  #8,rtc_rnr
        BFEXTU  d0{7:4},d1      ; Monatsdatum eintragen
        move.b  d1,rtc_data
        move.b  #9,rtc_rnr
        BFEXTU  d0{0:7},d1      ; Jahresdatum + 10 eintragen
        add.b   #$A,d1
        move.b  d1,rtc_data
        move.b  #$B,rtc_rnr
        move.b  #$E,rtc_data    ; RTC ist initialisiert!
        rts
;=============================================================
; Fehler (-1) in d0 zurückliefern!
;
seterr:
        moveq   #$FF,d0
        rts
;=============================================================
```

```
; Testroutine für Zugriff auf RTC-I/O-Bereich. Wenn RTC nicht vor-
; handen ist, wird eine eigene Bus-Error-Routine angesprungen und
; dort dann das Carry-Flag gesetzt (= keine RTC vorhanden!).
;
chkrtc:
        movea.l   a7,a0               ; Stackptr. in a0 merken
        movea.l   8,a1                ; Buserror-Ptr. nach a1
        move.l    #rtcberr,$00000008  ; Buserror-Ptr. auf Adr.
                                      ; $E02372 setzen
        move.b    #0,rtc_rnr          ; Zugriff auf Sekundenregister
        move.b    rtc_data,d0         ; Wenn die Routine bis hierhin
        move.l    a1,8                ; kommt, gibt es auch die RTC!
        andi      #$FE,ccr            ; Also Buserror-Ptr. restoren
        rts                           ; und mit Carry-Clear zurück!

; Die folgende Routine fängt eine Bus-error-Exception ab, wenn auf
; den RTC- I/O-Bereich zugegriffen wird, ohne daß eine RTC vorhan-
; den ist!
;
rtcberr:
        movea.l   a0,a7               ; Stackptr. restoren
        move.l    a1,8                ; Alten Buserror-Ptr. restoren
        ori       #1,ccr              ; Carry-Bit setzen (= No RTC!)
        rts                           ; Und raus hier!
```

Kapitel 11: Sonstiges zum TT

Auf die neuen Komponenten und Hardware-Eigenschaften des TT wurde ja in den vorigen Kapiteln bereits eingegangen. Dabei wurden jedoch nur jene Funktionsgruppen behandelt, die "anders" als beim "normalen" ST/STE sind. Natürlich finden sich bereits von der ST-Serie her bekannte (und beschriebene) Bausteine und Funktionsgruppen ebenfalls im TT wieder. Auf diese Funktionsgruppen soll deshalb nicht nochmal in besonderen Kapiteln eingegangen werden, sondern diese werden in diesem Kapitel noch kurz angesprochen. Informationen zu der prinzipiellen Funktionsweise finden sich in den entsprechenden ST-Hardware-Kapiteln.

Die ACIAs im TT

sind von der Funktion und Beschaltung her gleich mit denen des ST! So ist die Anbindung der MIDI-Schnittstellen und der Tastatur identisch mit der beim MEGA ST.

Die Tastatur mit ihrem Tastaturprozessor verhält sich genau wie die MEGA ST-Tastatur. So kann man ohne Probleme die MEGA ST-Tastatur am TT/MEGA STE betreiben und umgekehrt. Auch beim TT existiert keine Reset-Leitung mehr zur Tastatur hin. Die Folge ist dann auch hier, daß bei einem Tastatur-"Absturz" ein "echter Hardware-Reset" ausgeführt werden muß (Rechner aus- und nach einer kurzen Zeit wieder einschalten!), um die Tastatur wieder zur Arbeit zu bewegen. Auch der "Trick", bei gedrückter Maustaste den Rechner einzuschalten, führt (wie gewohnt) zu einem unmöglich konfigurierten Tastaturprozessor, mit dem ein Arbeiten nicht möglich ist (siehe auch im ST-Hardware-Teil "Die ACIAs im ST").

Der PSG im TT

Auch im TT/MEGA STE wird der YM 2149 (AY-5-8910 im MEGA STE) als Soundchip und Parallelport benutzt. Das davon erzeugte Soundsignal wird, wie beim STE erläutert, zu dem Soundsignal vom PCM-Soundmodul dazugemischt und läßt sich durch entsprechende Programmierung des Volume/Tone-Controllers LMC1992 auch bei Bedarf unterdrücken.

Was die Portanschlüsse des PSG im TT angeht, werden diese, wie vom ST her bereits bekannt, als Daten- und Steuerleitungen für den Parallel-Port (Druckerport) und zum Drive- und Side-select für die Floppy-Disk-Laufwerke benutzt. Beim TT sind die Steuersignale zur Floppy-Disk-Schnittstelle (Drive- und Side-Select) jedoch nochmal über eine Treiberschaltung gepuffert und werden erst freigeschaltet, wenn die Betriebsspannung stabil ist (über ein "Power Good"-Signal vom Netzteil!). Die beiden Portleitungen IOA7 und IO6 weisen eine neue Funktion auf.

Mit *IOA7* läßt sich beim TT/MEGA STE umschalten, ob der SCC-Kanal A auf dem LAN-Anschluß (Mini-DIN-Buchse an der Seite des Gerätes) oder auf dem SERIAL 2-Anschluß (9pol. SubD-Buchse an der Rückseite des Gerätes, wenn keine VME-Buskarte eingesteckt ist!) arbeitet. Setzt man mit der XBios-Funktion Ongibit($80) den Port-Anschluß IOA7 auf "Ein", ist SERIAL 2 eingeschaltet. Mit Offgibit($7F) arbeitet der SCC-Kanal A auf dem LAN-Anschluß.

Nach neuesten Informationen wird ATARI aber diesen Port-Anschluß des PSG nicht mehr für diese Umschaltung verwenden, sondern statt dessen am Parallelport zusätzlich zur Verfügung stellen, um das Anschließen von Scannern zu erleichtern. Die Umschaltung der beiden seriellen Anschlüsse LAN und SERIAL 2 erfolgt dann über einen Jumper auf der Hauptplatine unter der Festplatte.

IOA6 des PSG wird zum Ein- und Ausschalten des Lautsprechers im TT verwendet. Dabei wird das Soundsignal vom Volume/Tone-Controller LMC1992, das über eine Treiberstufe auf den Lautsprecher gelangt, durch ein Low am IOA6-Anschluß eingeschaltet.

Ongibit($40):	Lautsprecher im TT stummschalten.
Offgibit($BF):	Lautsprecher im TT einschalten.

Wichtig! Das Ein- und Ausschalten über IOA6 des PSG betrifft nur den internen Lautsprecher und nicht das über die Cinch-Buchsen (Audio out Links und Rechts) abgreifbare Tonsignal!

DMA-Sound im TT

Die Erzeugung und Kontrolle des PCM-Stereo-Sounds funktioniert im TT/MEGA STE genauso, wie beim STE beschrieben! Also können alle Adressen und Funktionsbeschreibungen (auch was das MMICROWIRE(TM)-Interface angeht!) dazu direkt im Hardware-Teil "Die Hardware des ATARI STE" nachgelesen werden.

Auch was die Verknüpfung vom "Monochrom-Detect"-Signal mit dem Frame-Ende-Signal an Port I7 des MFP (im TT ist das der ST-MFP!) angeht, ist alles so, wie beim STE beschrieben. Man muß nur beachten, daß mit Monochrombetriebsart beim TT eben die TT-High-Auflösung gemeint ist und nicht etwa ST-High!

Die etwas geänderte Schaltung (andere Unit-Nummern und zusätzliche Lautsprechertreiber-Stufe) für den TT ist nachfolgend aufgeführt!

Weil im TT kein SHIFTER der STE-Serie zum Einsatz kommt, mußte der DMA-Soundbetrieb über einen eigenen SOUNDSHIFTER realisiert werden. Dieser Custom-Chip dient zur

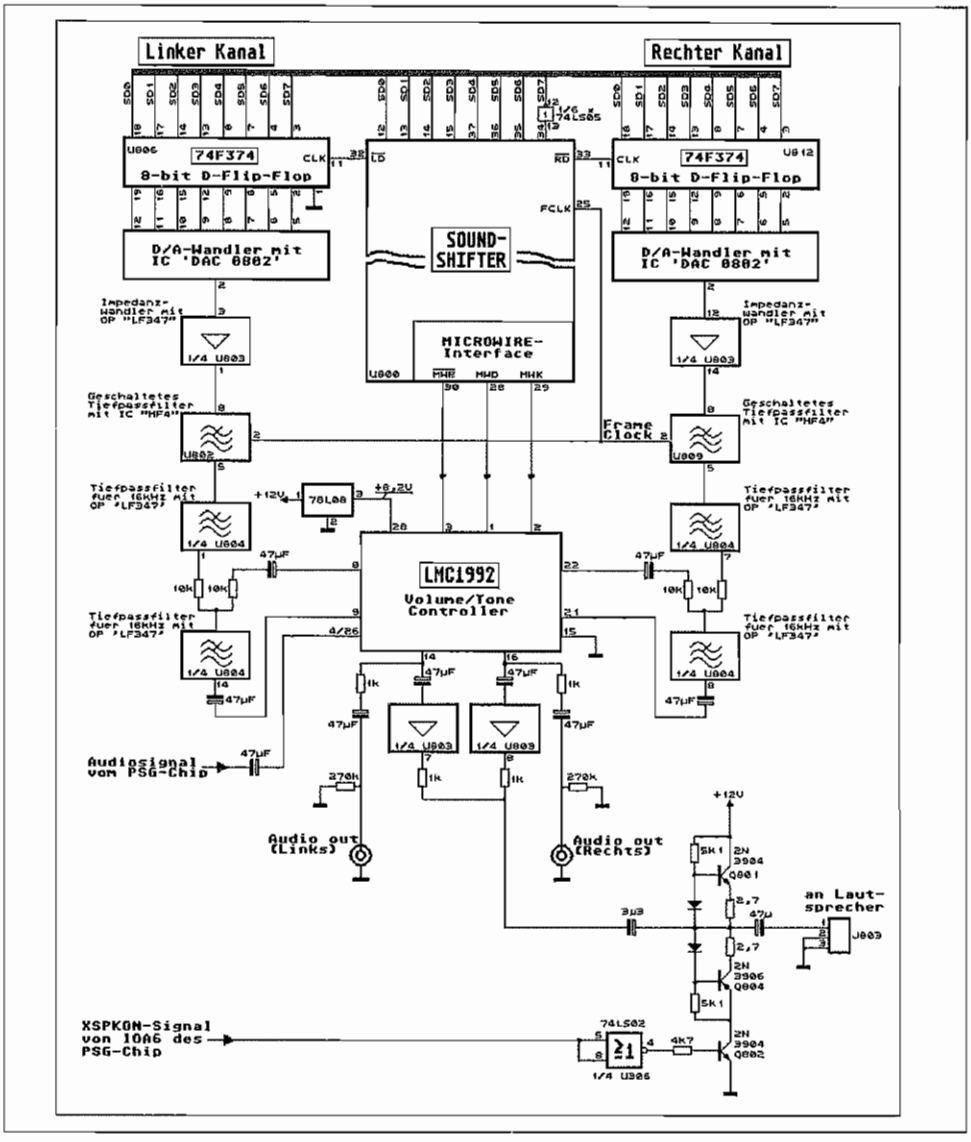


Abb. 11.1: Der TT-DMA-Soundteil

Erzeugung der Sound-DMA- und für die MICROWIRE(TM)-Steuersignale. Für den Programmierer verhält sich dieser Chip aber genauso wie der Sound-DMA-Teil des STE-SHIFTERS!

Das Floppy-Disk Interface

Was die Ansteuerung der Floppy-Disk-Laufwerke im TT/MEGA STE angeht, funktioniert (erstmal) alles genauso, wie bereits beim ST beschrieben.

Weil aber 720 KByte auf einer 3,5"-Disk wohl nicht mehr so ganz zeitgemäß sind, hat ATARI auf Abhilfe gesonnen und eine Möglichkeit vorgesehen, die Speicherkapazität zu verdoppeln. Diese erhöhte Kapazität erfordert spezielle Disketten (HD-Disks) und natürlich auch entsprechende HD-Disklaufwerke.

Um die doppelte Speicherkapazität zu erhalten, werden pro Spur bei HD-Betrieb nun die doppelte Anzahl an Sektoren pro Spur untergebracht. Das geht nur mit einer höheren Datentransferrate auf den Schreib- und Lesedatenleitungen zum Laufwerk. Um diese Verdopplung der Datenrate zu erzielen, wird bei HD-Betrieb der Systemtakt des FDC-Chips verdoppelt (statt 8 MHz dann 16 MHz!). Den Arbeitstakt für den FDC liefert im TT die MCU (U206) an Pin 44, die auch die Taktumschaltung vornimmt.

TT und MEGA STE sind (im Prinzip) HD-fähig

Die Verdopplung des Taktes "verkraften" aber nicht alle Exemplare des WD1772-FDC-Chips (schließlich ist dieser ja auch nur für 8 MHz ausgelegt!). ATARI will aber einen "eigenen", zum WD1772 aufwärtskompatiblen FDC-Chip liefern, der diese Taktumschaltung beherrscht. In meinem TT arbeitet allerdings auch bereits der "gute, alte WD1772" mit 16 MHz einwandfrei und bedient ein nachträglich eingebautes HD-Laufwerk! Allerdings gibt es bei Dauerbetrieb mit dieser hohen Taktfrequenz Probleme mit der Wärmeentwicklung des FDC! Zwar "stirbt" dieser nicht gerade, aber er arbeitet nicht mehr sicher.

Die Umschaltung des internen Laufwerks auf HD erfolgt über ein High-Signal an Pin 2 des Floppy-Steckverbinders. Dieser Anschluß existiert aber an der Anschlußbuchse zum externen Laufwerk nicht! Wenn man also mit einem externen, HD-fähigen Laufwerk arbeiten will, sollte dieses so konfiguriert werden, daß es die HD-Betriebsart aus der zusätzlichen Öffnung einer HD-Diskette erkennt und entsprechend umschaltet. Das TOS erkennt HD-formatierte Disks durch entsprechende Leseversuche mit den beiden möglichen Taktfrequenzen für den FDC.

Welche Density ein korrektes Lesen ermöglicht, wird im Drive-Status-Block gemerkt, der für diese Information erweitert wurde. Im TOS 3.01 findet sich der Drive-Status-Block (DSB) für das Laufwerk A ("dsb0") an Adresse $AFA. Der zweite DSB ("dsb1") liegt bei $B00! Vorsicht! Das sind Adressen von internen Datenstrukturen, die sich von TOS-Version zu TOS-Version ändern können! Also erst gar nicht benutzen! Für das Formatieren sollten die ent-

sprechenden Betriebssystem-Funktionen benutzt werden. Die behandeln die angeschlossene Hardware dann schon richtig!

Jeweils das zweite Word eines DSB wird für die aktuelle Density-Einstellung benutzt. Dabei ist das niedrigwertigste Bit für die Takteinstellung (Bit 0 = 1: 16 MHz-Takt) und das Bit 1 für die Density-Signalisierung (Pin 2 des Floppy-Steckverbinders) zum Laufwerk zuständig (Bit 1 = 1: Density-Leitung auf High!).

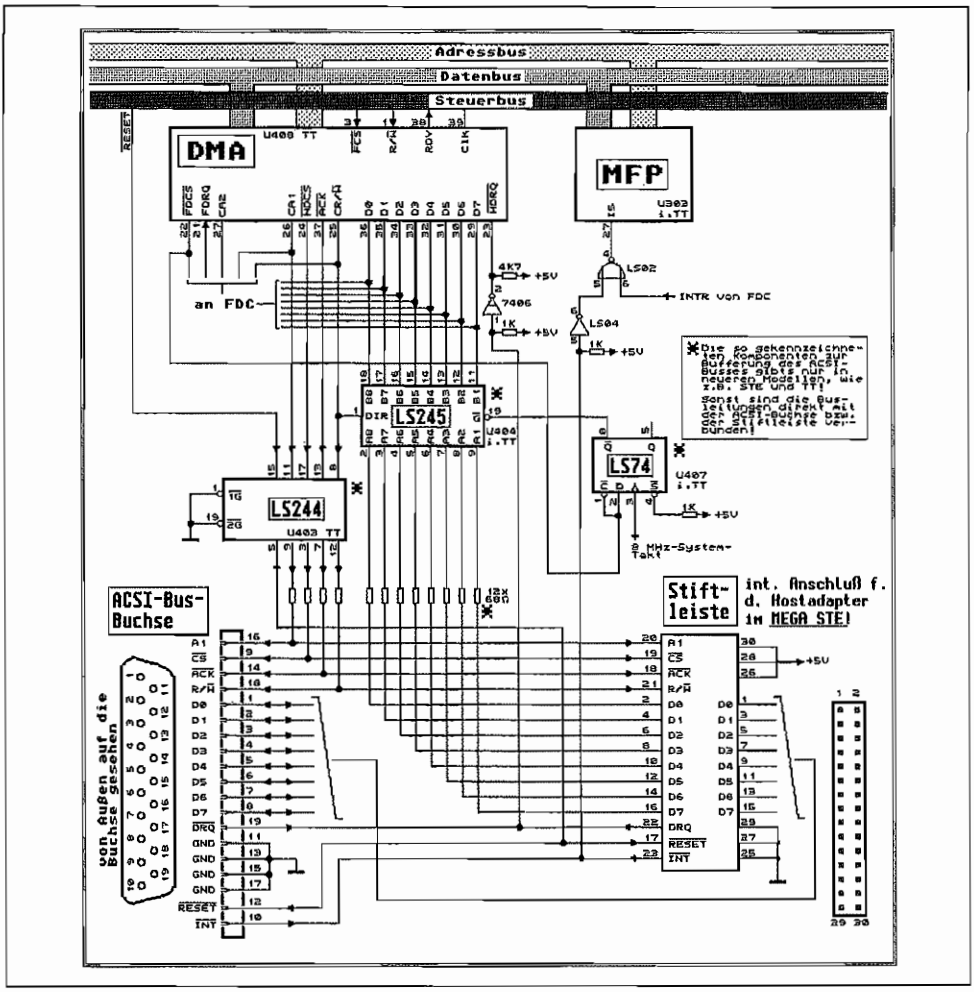


Abb. 11.2: Der TT/MEGA STE ACSI-Busanschluß

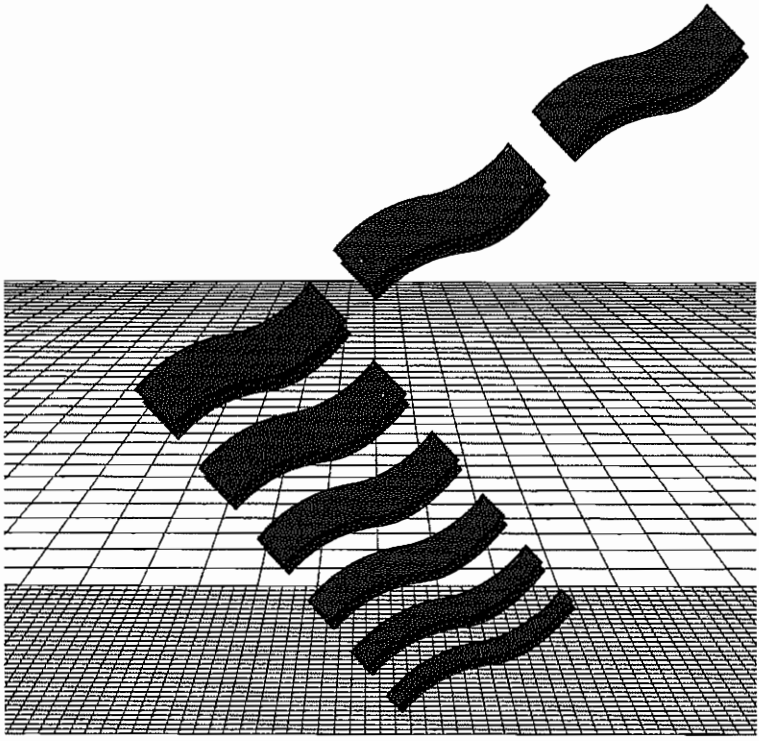
Das Hardware-Register (Word-Zugriffe, nur die beiden untersten Bits werden verwendet) für die Taktumschaltung und die Ansteuerung der HD-Signal-Leitung liegt bei Adresse $(FF)FF 860E! Auch diese Adresse ist nicht von ATARI dokumentiert und kann sich evtl. ändern. Also nicht direkt verwenden!

Der TT/MEGA STE und der ACSI-Bus

Auch hier gibt's gegenüber der Ausführung im ST keine gravierenden Änderungen. Die wesentlichste Abweichung besteht in der Verwendung von Treiberstufen für die ACSI-Bus-Signale zur Stabilisierung der Signale. Außerdem wird der DMA-Chip dadurch etwas weniger anfällig gegen Zerstörungen von "Außen".

Beim MEGA STE hat ATARI ja ebenfalls eine Festplatte (SCSI!) mit ins Gehäuse eingebaut. Für deren Betrieb ist dann natürlich ein entsprechender Hostadapter erforderlich, um die ACSI-Bussignale auf SCSI-Signale umzusetzen. Der Anschluß dieses Hostadapters erfolgt über eine 30pol. Pfostenfeldleiste auf dem Motherboard des MEGA STE. Die Datenbusleitungen werden dabei "durch" den Hostadapter geführt (eine Seite Input; andere Seite Output)! Wenn kein Hostadapter eingebaut ist, müssen deshalb die Datenbusleitungen an der Pfostenfeldleiste gebrückt sein, um die Signale auch am außenliegenden ACSI-Anschluß zur Verfügung zu haben! Die interne Platte hat standardmäßig die ACSI-Device-Nummer 0. Außen angeschlossene ACSI-Device sollten sich also daran entsprechend anschließen.

Anhänge



Anhang A: BIOS-, XBIOS- und GEMDOS-Fehlernummern

TOS-Fehlermeldungen

Beim Aufruf von BIOS-, XBIOS- und GEMDOS-Funktionen wird im allgemeinen im Register D0 ein 32-Bit-Statuswert zurückgeliefert, der darüber Auskunft gibt, ob, und wenn ja, was für ein Fehler bei der Ausführung der Funktion aufgetreten ist. Hier eine Übersicht:

0: E_OK ("OK (no error)")

Funktion erfolgreich ausgeführt (kein Fehler aufgetreten).

–1: ERROR ("Error")

Es ist ein Fehler aufgetreten, der nicht genauer spezifiziert werden kann.

–2: EDRVNR ("Drive not ready")

Angesprochenes Gerät ist nicht angeschlossen, nicht funktionsbereit oder reagiert nicht innerhalb der gesetzten Frist (Timeout).

–3: EUNCMD ("Unknown command")

Dem angesprochenen Peripheriegerät ist das gegebene Kommando unbekannt.

–4: E_CRC ("CRC error")

Beim Lesen eines Sektors ist ein Fehler aufgetreten (aufgetreten bei der CRC-Überprüfung).

–5: EBADRQ ("Bad request")

Das Peripheriegerät kann das Kommando nicht ausführen. Befehlsparameter und Kontext prüfen.

–6: E_SEEK ("Seek error")

Der angesprochene Track konnte vom Laufwerk nicht erreicht werden.

–7: EMEDIA ("Unknown media")

Leseversuch gescheitert, da das Medium keinen korrekten Bootsektor besitzt.

–8: ESECNF ("Sector not found")

Der betreffende Sektor wurde nicht gefunden.

–9: EPAPER ("Out of paper")

Drucker nicht betriebsbereit.

–10: EWRITF ("Write fault")

Fehler bei Schreiboperation aufgetreten.

–11: EREADF ("Read fault")

Fehler bei Leseoperation aufgetreten.

–12: EGENRL ("General error")

Allgemeiner Fehler (Kommentar im "Hitchhiker's guide to the Bios": 'Reserved for future catastrophes').

–13: EWRPRO ("Write on write-protected media")

Es wurde versucht, auf ein schreibgeschütztes Medium zu schreiben.

–14: E_CHNG ("Media change detected")

Seit der letzten Schreiboperation wurde das Medium gewechselt.

–15: EUNDEV ("Unknown device")

Das angesprochene Gerät ist dem Betriebssystem unbekannt.

–16: EBADSF ("Bad sectors on format")

Beim Formatiervorgang wurden defekte Sektoren entdeckt.

–17: EOTHER ("Insert other disk (request)")

Eine andere Diskette muß eingelegt werden. Tritt nur auf, wenn Laufwerk B: angesprochen wird, ohne angeschlossen zu sein.

In diesem Fall wird der Benutzer aufgefordert, "Diskette B:" in das erste Laufwerk einzulegen.

–18: EINSERT ("Insert disk")

Meta-DOS-Fehler: Medium einlegen!

–19: EDVNRSP ("Device not responding")

Meta-DOS-Fehler: Gerät antwortet nicht.

–32: EINVFN ("Invalid function number")

Unbekannte Funktionsnummer. Man erhält diese Meldung, wenn man eine undefinierte GEMDOS-Funktion aufruft (unter MS-DOS: Fehlercode 1).

–33: EFILNF ("File not found")

Datei nicht gefunden (unter MS-DOS: Fehlercode 2).

–34: EPTHNF ("Path not found")

Angesprochener Ordner nicht gefunden (unter MS-DOS: Fehlercode 3).

–35: ENHNDL ("Handle pool exhausted")

Keine Dateihandles mehr (zu viele Dateien geöffnet, unter MS-DOS: Fehlercode 4).

–36: EACCDN ("Access denied")

Zugriff nicht erlaubt (unter MS-DOS: Fehlercode 5).

–37: EIHNDL ("Invalid handle")

Das Dateihandle war nicht korrekt (unter MS-DOS: Fehlercode 6).

–39: ENSMEM ("Insufficient memory")

Nicht genügend Speicher vorhanden (unter MS-DOS: Fehlercode 8).

–40: EIMBA ("Invalid memory block address")

Die Adresse des Speicherblocks war nicht gültig (unter MS-DOS: Fehlercode 9).

–46: EDRIVE ("Invalid drive specification")

Die Laufwerksbezeichnung war ungültig (unter MS-DOS: Fehlercode 15).

–48: ENSAME ("Not the same drive")

Dateien sind auf verschiedenen logischen Laufwerken.

–49: ENMFIL ("No more files")

Es können keine Dateien mehr geöffnet werden (unter MS-DOS: Fehlercode 18).

–58: ELOCKED ("Record is locked")

Es wurde versucht, auf einen gelockten Bereich einer Datei zuzugreifen (nur im Zusammenhang mit einer Netzwerk-GEMDOS-Version).

–59: ENSLOCK ("Matching lock not found")

Es wurde versucht, einen nicht existierenden Lock (falscher Offset oder/und falsche Länge) zu entfernen (nur im Zusammenhang mit einer Netzwerk-GEMDOS-Version).

–64: ERANGE ("Range error")

Dateizeiger in ungültigem Bereich.

–65: EINTRN ("GEMDOS internal error")

Interner Fehler im GEMDOS (hoffentlich passiert's nie!).

–66: EPLFMT ("Invalid executable file format")

Das angesprochene Programm hat nicht das korrekte Format, um geladen zu werden.

–67: EGSBF ("Memory block growth failure")

Es wurde versucht, einen allozierten Speicherblock zu vergrößern.

Anhang B: Wichtige Betriebssystemstrukturen

AESPB (AES Parameter Block)

Offset	Struktur
	typedef struct
	{
$00	WORD *cb_pcontrol; /* Zeiger auf contrl[] */
$04	WORD *cb_pglobal; /* Zeiger auf global[] */
$08	WORD *cb_pintin; /* Zeiger auf int_in[] */
$0C	WORD *cb_pintout; /* Zeiger auf int_out[] */
$10	LONG *cb_padrin; /* Zeiger auf addr_in[] */
$14	LONG *cb_padrout; /* Zeiger auf addr_out[] */
	} AESPB;

ARHEADER (Archivheader)

Offset	Struktur
	typedef struct
	{
$00	char a_fname[14]; /* Dateiname */
$0E	LONG a_modti; /* Zeitpunkt des letzten Zugriffs */
$12	BYTE a_userid; /* nicht benutzt */
$13	BYTE a_gid; /* nicht benutzt */
$14	WORD a_fimode; /* Filemodus */
$16	LONG a_fsize; /* Dateilänge */
$1A	WORD reserved; /* Null */
	} ARHEADER;

BASEPAGE

Offset	Struktur
	`typedef struct basepage`
	`{`
$00	`void *p_lowtpa;  /* Anfangsadresse der TPA (Offset 0) */`
$04	`void *p_hitpa;  /* Erstes Byte nach dem Ende der TPA (Offset 4) */`
$08	`void *p_tbase;  /* Anfangsadresse des Programmcodes (TEXT-Abschnitt) (Offset 8) */`
$0C	`LONG p_tlen;  /* Länge des Programmcodes (Offset 12) */`
$10	`void *p_dbase;  /* Adresse des Bereichs für vorinitialisierte Daten (DATA-Abschnitt) (Offset 16) */`
$14	`LONG p_dlen;  /* Länge des Datenabschnitts (Offset 20) */`
$18	`void *p_bbase;  /* Adresse des Variablenbereichs (BSS-Abschnitt) (Offset 24) */`
$1C	`LONG p_blen;  /* Länge des Variablenbereichs (Offset 28) */`
$20	`DTA  *p_dta;  /* Zeiger auf Default-DTA (Vorsicht! Zeigt zunächst in die Kommandozeile) (Offset 32) */`
$24	`struct basepage *p_parent;` `/* Zeigt auf die Basepage (BASEPAGE) des aufrufenden Prozesses (Offset 36) */`
$28	`LONG p_resrvd0;  /* reserviert */`
$2C	`char *p_env;  /* Adresse der Environment-Strings (Offset 44) */`
$30	`CHAR p_resrvd1[80]; /* reserviert */`
$80	`char p_cmdlin[128]; /* Kommandozeile (dabei wird im ersten Byte die Anzahl der Zeichen eingesetzt. Die maximale Länge der Kommandozeile beläuft sich erstaunlicherweise nicht auf 127, sondern auf 124 Zeichen.) (Offset 128) */`
	`} BASEPAGE;`

BCB (Buffer Control Block)

Offset	Struktur

```
        typedef struct _bcb
        {
$00     struct _bcb     *b_link;  /* Zeiger auf nächsten BCB */
$04     WORD            b_neg1;   /* auf -1 initialisieren */
$06     WORD            b_private[5];
$10     void            *b_buf;   /* Zeiger auf den eigentlichen
                                     Puffer */
        } BCB;
```

BCONMAP (siehe 'Bconmap()')

Offset	Struktur

```
        typedef struct
        {
$00     MAPTAB    *maptab;      /* Zeiger auf Funktionstabelle */
$04     WORD      maptabsize;   /* Anzahl der Einträge */
        } BCONMAP;
```

BITBLK

Offset	Struktur

```
        typedef struct
        {
$00     UWORD     *bi_pdata;     /* Zeiger auf Image */
$04     UWORD     bi_wb;         /* Breite in Bytes */
$06     UWORD     bi_hl;         /* Höhe in Pixelzeilen */
$08     WORD      bi_x;          /* X-Position */
$0A     WORD      bi_y;          /* Y-Position */
$0C     WORD      bi_color;      /* Farbe */
        } BITBLK;
```

BPB (BIOS Parameter Block)

Offset	Struktur
	`typedef struct`
	`{`
$00	`WORD recsiz;  /* Bytes pro Sektor */`
$02	`WORD clsiz;  /* Sektoren pro Cluster (Einheit) */`
$04	`WORD clsizb;  /* Bytes pro Cluster */`
$06	`WORD rdlen;  /* Länge des Wurzelverzeichnisses in Sektoren */`
$08	`WORD fsiz;  /* Länge des File Allocation Table (FAT) */`
$0A	`WORD fatrec;  /* Startsektor der zweiten FAT */`
$0C	`WORD datrec;  /* Sektornummer des ersten freien Clusters */`
$0E	`WORD numcl;  /* Cluster-Gesamtzahl auf dem Medium */`
$10	`WORD bflags;  /* Bitvektor, zur Zeit nur Bit 0 belegt: 0 (12-Bit-FAT), 1 (16-Bit-FAT) */`
	`} BPB;`

CPXHEAD (CPX-Dateikopf)

Offset	Struktur
	`typedef struct`
	`{`
$00	`UWORD magic;  /* = 100 */`
$02	`struct`
	`{`
	`unsigned reserved : 13;  /* reserviert */`
	`unsigned resident : 1;  /* RAM-resident */`
	`unsigned bootinit : 1;  /* Boot-Initialisierung */`
	`unsigned setonly : 1;  /* Set-Only */`
	`} flags;`
$04	`LONG cpx_id;  /* eindeutige CPX-ID */`
$08	`UWORD cpx_version;  /* CPX-Versionsnummer */`
$0A	`char i_text[14];  /* Icontext */`
$18	`UWORD sm_icon[48];  /* Bitmap (32x24 Pixel) */`
$78	`UWORD i_color;  /* Iconfarbe */`

Offset	Struktur
$7A	char title_text[18]; /* Name */
$7C	UWORD t_color; /* Textfarbe */
$8E	char buffer[64]; /* nicht flüchtiger Puffer */
$CE	char reserved[306]; /* reserviert */
	} CPXHEAD;

CPXINFO

Offset	Struktur
	typedef struct
	{
$00	WORD cdecl (*cpx_call)(GRECT *work);
$04	void cdecl (*cpx_draw)(GRECT *clip);
$08	void cdecl (*cpx_wmove)(GRECT *work);
$0C	void cdecl (*cpx_timer)(WORD *event);
$10	void cdecl (*cpx_key)(WORD kstate, WORD key, WORD *event);
$14	void cdecl (*cpx_button)(MRETS *mrets, WORD *event);
$18	void cdecl (*cpx_m1)(MRETS *mrets, WORD *event);
$1C	void cdecl (*cpx_m2)(MRETS *mrets, WORD *event);
$20	WORD cdecl (*cpx_hook)(WORD event, WORD *msg, MRETS *mrets, WORD *key, WORD *nclicks);
$24	void cdecl (*cpx_close)(WORD flag);
	} CPXINFO;

DIR (Verzeichnis-Eintrag)

Offset	Struktur
	typedef struct
	{
$00	char dir_name[11]; /* Name inkl. Erweiterung ohne Punkt */
$0B	BYTE dir_attr; /* Attribut */

Offset	Struktur
$0C	BYTE dir_dummy[10]; /* unbenutzt */
$16	UWORD dir_time; /* Erstellungszeit */
$18	UWORD dir_date; /* Erstellungsdatum */
$1A	UWORD dir_stcl; /* erster Cluster */
$1C	ULONG dir_flen; /* Länge der Datei */
	} DIR;

DISKINFO

Offset	Struktur
	typedef struct
	{
$00	LONG b_free; /* Anzahl der freien Cluster */
$04	LONG b_total; /* Gesamtzahl der Cluster */
$08	LONG b_secsiz; /* Bytes pro Sektor */
$0C	LONG b_clsiz; /* Sektoren pro Cluster /*
	} DISKINFO;

DOSTIME

Offset	Struktur
	typedef struct
	{
$00	UWORD time; /* Zeit wie in "Tgettime" */
$02	UWORD date; /* Datum wie in "Tgetdate" */
	} DOSTIME;

DTA (Disk Transfer Area)

Offset	Struktur
	typedef struct
	{
$00	BYTE d_reserved[21]; /* fürs GEMDOS reserviert */

Offset	Struktur
$15	BYTE d_attrib; /* Datei-Attribut */
$16	UWORD d_time; /* Uhrzeit */
$18	UWORD d_date; /* Datum */
$1A	LONG d_length; /* Dateilänge */
$1E	char d_fname[14]; /* Dateiname */
	} DTA;

FONT_HDR

Offset	Struktur
	typedef struct
	{
$00	WORD font_id; /* Fontnummer */
$02	WORD point; /* Größe im Punktmaß */
$04	char name[32]; /* Name des Zeichensatzes */
$24	UWORD first_ade; /* Erstes Zeichen im Zeichensatz (nur die wenigsten Drucker-Schriften enthalten Control-Zeichen, also ASCII-Werte kleiner 32) */
$26	UWORD last_ade; /* Letztes Zeichen im Zeichensatz */
$28	UWORD top; /* Abstand Topline<->Baseline */
$2A	UWORD ascent; /* Abstand Ascentline<->Baseline */
$2C	UWORD half; /* Abstand Halfline<->Baseline */
$2E	UWORD descent; /* Abstand Descentline<->Baseline*/
$30	UWORD bottom; /* Abstand Bottomline<->Baseline */
$32	UWORD max_char_width; /* größte Zeichenbreite */$34
UWORD	max_cell_width; /* größte Zeichenzellenbreite */
$36	UWORD left_offset; /* linker Offset für Kursivschrift */
$38	UWORD right_offset; /* rechter Offset für Kursivschrift */
$3A	UWORD thicken; /* Verbreiterungsfaktor für Fettschrift (bei Fettschrift

Offset	Struktur		
			wird die Breite um genau so viele Pixel erhöht) */
$3C	UWORD	ul_size;	/* Dicke der Unterstreichung */
$3E	UWORD	lighten;	/* Maske für helle Schrift (wird mit Raster undiert, normalerweise 0x5555) */
$40	UWORD	skew;	/* Maske für Kursivschrift (beschreibt, an welchen Stellen die Rasterzeilen verschoben werden sollen, normalerweise 0x5555 */
$42	UWORD	flags;	/* Bit 0: gesetzt, wenn es der "Default system font" ist Bit 1: gesetzt, wenn die "Horizontal offset table" benutzt wird Bit 2: gesetzt, wenn Bytes nicht vertauscht werden müssen (Motorola-Format) Bit 3: gesetzt, wenn der Font nicht proportional ist */
$44	BYTE	*hor_table;	/* Zeiger auf "horizontal offset table"; in der Datei der Offset zum Dateibeginn */
$48	UWORD	*off_table;	/* Zeiger auf "character offset table" */
$4C	UWORD	*dat_table;	/* Zeiger auf Fontimage */
$50	UWORD	form_width;	/* Breite des Zeichensatzimage*/
$54	UWORD	form_height;	/* Höhe des Zeichensatzimage */
$58	FONT_HDR	*next_font;	/* Zeiger auf nächsten Fontheader (in der Fontdatei zunächst unbenutzt) */
	} FONT_HDR;		

GEM_MUPB (GEM Memory Usage Parameter Block)

Offset	Struktur

```
        typedef struct
        {
$00     LONG gm_magic;      /* muß 0x87654321 sein */
$04     void *gm_end;       /* Ende des von GEM benötigten
                               Speicherbereichs */
$08     void *gm_init;      /* Startadresse von GEM */
        } GEM_MUPB;
```

GRECT

Offset	Struktur

```
        typedef struct
        {
$00     WORD g_x;       /* X-Position */
$02     WORD g_y;       /* Y-Position */
$04     WORD g_w;       /* Breite */
$06     WORD g_h;       /* Höhe */
        } GRECT;
```

ICONBLK

Offset	Struktur

```
        typedef struct
        {
$00     UWORD     *ib_pmask;    /* Zeiger auf Icon-Maske */
$04     UWORD     *ib_pdata;    /* Zeiger auf Icon-Daten */
$08     char      *ib_ptext;    /* Zeiger auf Icon-Text */
$0C     UWORD     ib_char;      /* Zeichen, das im Icon erschei-
                                   nen soll, und Vorder- und
                                   Hintergrundfarbe des Icons */
```

Offset	Struktur
$0E	UWORD ib_xchar; /* dessen X-Position */
$10	UWORD ib_ychar; /* dessen Y-Position */
$12	UWORD ib_xicon; /* X-Position des Icons */
$14	UWORD ib_yicon; /* Y-Position des Icons */
$16	UWORD ib_wicon; /* Breite des Icons */
$18	UWORD ib_hicon; /* Höhe des Icons */
$1A	WORD ib_xtext; /* X-Position der Beschriftung */
$1C	WORD ib_ytext; /* Y-Position der Beschriftung */
$1E	UWORD ib_wtext; /* Textbreite in Pixel */
$20	UWORD ib_htext; /* Texthöhe in Pixel */
$22	UWORD ib_resvd; /* reserviert */
	} ICONBLK;

IMGHEADER

Offset	Struktur
	typedef struct
	{
$00	WORD im_version; /* Versionsnummer der Imagedatei*/
$02	UWORD im_headlength; /* Länge des Headers in Words */
$04	UWORD im_nplanes; /* Anzahl der Farbebenen */
$06	UWORD im_patlen; /* Musterlänge in Byte */
$08	UWORD im_pixwidth; /* Pixelbreite in mm/1000 */
$0A	UWORD im_pix_height; /* Pixelhöhe in mm/1000 */
$0C	UWORD im_scanwidth; /* Zeilenbreite in Pixel */
$0E	UWORD im_nlines; /* Anzahl der Zeilen */
	} IMGHEADER;

IOREC

Offset	Struktur
	typedef struct
	{
$00	LONG ibuf; /* Zeiger auf den Puffer */
$04	WORD ibufsiz; /* Länge des Puffers */

Offset	Struktur
$06	WORD ibufhd; /* nächste Schreibposition */
$08	WORD ibuftl; /* nächste Leseposition */
$0A	WORD ibuflow; /* "untere Wassermarke" */
$0C	WORD ibufhi; /* "obere Wassermarke" */
	} IOREC;

LINEA

Offset	Struktur
	typedef struct
	{
$00	WORD PLANES; /* Anzahl der Bildschirmebenen (+$00) */
$02	WORD WIDTH; /* Bytes pro Bildschirmzeile (+$02) */
$04	LONG CONTRL; /* Zeiger auf contrl[] (VDI) (+$04) */
$08	LONG INTIN; /* Zeiger auf int_in[] (VDI) (+$08) */
$0C	LONG PTSIN; /* Zeiger auf pts_in[] (VDI) (+$0C) */
$10	LONG INTOUT; /* Zeiger auf int_out[] (VDI) (+$10) */
$14	LONG PTSOUT; /* Zeiger auf pts_out[] (VDI) (+$14) */
$18	WORD COLBIT0; /* Farbwert für Plane 0 (+$18) */
$1A	WORD COLBIT1; /* Farbwert für Plane 1 (+$1A) */
$1C	WORD COLBIT2; /* Farbwert für Plane 2 (+$1C) */
$1E	WORD COLBIT3; /* Farbwert für Plane 3 (+$1E) */
$20	WORD LSTLIN; /* letzten Pixel einer Linie zeichnen (1) oder nicht zeichnen (0) +$20) */
$22	WORD LNMASK; /* Muster für Linien (+$22) */
$24	WORD WMODE; /* VDI-Schreibmodus (+$24) */
$26	WORD X1,Y1,X2,Y2; /* Koordinatenangaben (+$26) */
$2E	LONG PATPTR; /* Zeiger auf Füllmuster (+$2E) */
$32	WORD PATMSK; /* dazugehörige Maske (+$32) */
$34	WORD MFILL; /* Füllmuster monochr./farbig (+$34)*/
$36	WORD CLIP; /* Clipping aus/an (+$36) */
$38	WORD XMINCL,YMINCL; /* linke obere Ecke d. Clip-rect. (+$38)*/
$3C	WORD XMAXCL,YMAXCL; /* rechte untere Ecke des Clip-rect. (+$3C) */

Offset	Struktur

```
$40     WORD XDDA;      /* vor Textausgaben auf 0x8000 setzen
                           (+$40) */
$42     WORD DDAINC;    /* Vergrößerungsfaktor - bei Vergrößerung:
                           256*(Wunschgröße-Aktuelle)/Aktuelle;
                           bei Verkleinerung:
                           256*(Wunschgröße)/Aktuelle (+$42) */
$44     WORD SCALDIR;   /* Vergrößerungsrichtung (0: verkleinern,
                           1: vergrößern) (+$44) */
$46     WORD MONO;      /* Proportionalschrift ja/nein (+$46) */
$48     WORD SOURCEX,SOURCEY;
                        /* X,Y-Koordinate im Zeichensatz (+$48) */
$4C     WORD DESTX,DESTY;
                        /* Koordinate des Zeichens auf dem Bild-
                           schirm (+$4C) */
$50     WORD DELX,DELY; /* Breite und Höhe des Zeichens (+$50) */
$54     FONT_HDR  *FBASE;
                        /* Zeiger auf Zeichensatzimage (+$54) */
$58     WORD FWIDTH;    /* Breite des Zeichensatzimage (+$58) */
$5A     WORD STYLE;     /* Schreibstil (+$5A)
                           Bit 0:    Fettschrift
                           Bit 1:    Helle Schrift
                           Bit 2:    Kursivschrift
                           Bit 3:    Unterstrichene Schrift
                           Bit 4:    Umriß-Schrift */
$5C     WORD LITEMASK; /* Maske für das Schattieren (light)
                           (+$5C) */
$5E     WORD SKEWMASK; /* Maske für Italics (+$5E) */
$60     WORD WEIGHT;    /* zusätzliche Breite bei Bold (+$60) */
$62     WORD ROFF;      /* Kursiv-Offset rechts (+$62) */
$64     WORD LOFF;      /* Kursiv-Offset links (+$64) */
$66     WORD SCALE;     /* Vergrößerung ja/nein (+$66) */
$68     WORD CHUP;      /* Rotationswinkel * 10 (+$68) */
$6A     WORD TEXTFG;    /* Textfarbe (+$6A) */
$6C     LONG SCRTCHP;   /* Zeiger auf temp. Buffer für Texteffekte
                           (+$6C)                              */
$70     WORD SCRPT2;    /* Offset für denselben (+$70) */
$72     WORD TEXTBG;    /* Texthintergrundfarbe (+$72) */
$74     WORD COPYTRAN; /* Flag für 'COPY RASTER FORM'
                          (0: "opaque", 1: "transparent") (+$74) */
```

Offset	Struktur
$76	LONG SEEDABORT; /* Pointer auf Routine, die nach jeder Bildzeile eines "SEEDFILL" aufgerufen wird (+$76) */ } LINEA;

KBDVECS

Offset	Struktur
	typedef struct {
$00	void (*midivec)(); /* Midi-Eingabe */
$04	void (*vkbderr)(); /* Tastatur-Fehler */
$08	void (*vmiderr)(); /* MIDI-Fehler */
$0C	void (*statvec)(); /* Status von IKBD lesen */
$10	void (*mousevec)(); /* Mausabfrage */
$14	void (*clockvec)(); /* Uhrzeitabfrage */
$18	void (*joyvec)(); /* Joystickabfrage */
$1C	void (*midisys)(); /* MIDI-Systemvektor */
$20	void (*ikbdsys)(); /* IKBD-Systemvektor */
$24	WORD drvstat; /* IKBD-Treiberstatus */
	} KBDVECS;

KEYTAB

Offset	Struktur
	typedef struct {
$00	char *unshift; /* Tabelle für "normale" Tastendrücke */
$04	char *shift; /* Tabelle für Tastendrücke mit "Shift" */
$08	char *capslock; /* Tabelle für Tastendrücke bei "Capslock" */
	} KEYTAB;

MAPTAB (siehe 'Bconmap()')

Offset	Struktur
	`typedef struct`
	`{`
$00	`WORD (*Bconstat) (); /* Funktionszeiger */`
$04	`LONG (*Bconin) ();`
$08	`LONG (*Bcostat) ();`
$0C	`void (*Bconout) ();`
$10	`ULONG (*Rsconf) ();`
$14	`IOREC *iorec; /* Zeiger auf IOREC-Struktur */`
	`} MAPTAB;`

MD (Memory Descriptor)

Offset	Struktur
	`typedef struct md`
	`{`
$00	`struct md *m_link; /* Zeiger auf nächsten MD */`
$04	`LONG m_start; /* Anfangsadresse des Blocks */`
$08	`LONG m_length; /* Länge des Blocks */`
$0C	`BASEPAGE *m_own; /* Zeiger auf Prozeß-Beschreibungs-Struktur */`
	`} MD;`

METAHDR

Offset	Struktur
	`typedef struct`
	`{`
$00	`WORD mf_header; /* -1 (Metafile-Kennung) */`
$02	`WORD mf_hlength; /* Länge des Headers in Integers (24) */`

Offset	Struktur
$04	WORD mf_version; /* bei der aktuellen Version 101 (Version 1.01), Formel: 100*Hauptnummer+Unternummer */
$06	WORD mf_ndcrcfl; /* NDC/RC-Flag (0 oder 2) */
$08	WORD mf_extents[4]; /* optional - maximale Ausmaße der Grafik - können mit "v_meta_extents()" gesetzt werden (sonst mit Null gefüllt) */
$10	WORD mf_pagesz[2]; /* optional - Seitengröße in 1/10mm - kann mit "vm_pagesize()" gesetzt werden (sonst mit Nullen gefüllt) */
$14	WORD mf_coords[4]; /* optional - Koordinatensystem - kann mit "vm_coords()" gesetzt werden (sonst mit Nullen gefüllt) */
$1C	WORD mf_imgflag; /* falls Bit 0 = 1, so enthält die Datei Bitimages, sonst nicht. Die anderen Bits sind immer 0.*/
$1E	WORD mf_resvd[9]; /* zur Zeit unbenutzt */
	} METAHDR;

METAINFO

Offset	Struktur
	typedef struct
	{
$00	ULONG drivemap; /* Tabelle mit Bits für die Meta-DOS-Gerätetreiber "A".."Z" (Bit 0: "A") */
$04	char *version; /* Zeichenkette mit Namen und Versionsnummer von Meta-DOS */
$08	LONG reserved[2];
	} METAINFO;

MEVENT (Multi event)

Offset	Struktur
	typedef struct
	{
$00	UWORD e_flags;
$02	UWORD e_bclk;
$04	UWORD e_bmsk;
$06	UWORD e_bst;
$08	UWORD e_m1flags;
$0A	GRECT e_m1;
$12	UWORD e_m2flags;
$14	GRECT e_m2;
$1C	WORD *e_mepbuf;
$20	ULONG e_time;
$24	WORD e_mx;
$26	WORD e_my;
$28	UWORD e_mb;
$2A	UWORD e_ks;
$2C	UWORD e_kr;
$2E	UWORD e_br;
$30	UWORD e_m3flags;
$32	GRECT e_m3;
$3A	WORD e_xtra0;
$3C	WORD *e_smepbuf;
$40	ULONG e_xtra1;
$44	ULONG e_xtra2;
	} MEVENT;

MFDB (Memory Form Definition Block)

Offset	Struktur
	typedef struct
	{
$00	void *fd_addr; /* Zeiger auf Speicherblock (muß auf gerader Adresse liegen). Für den Bildspeicher muß man NULL übergeben, alle anderen Werte werden dann ignoriert/*

Offset	Struktur
$04	WORD fd_w; /* Breite des Speicherblocks in Punkten */
$06	WORD fd_h; /* Höhe des Speicherblocks in Punkten */
$08	WORD fd_wdwidth; /* Breite des Speicherblocks in 16-Bit-Worten */
$0A	WORD fd_stand; /* 0: geräteabhängiges Format, 1: Standardformat */
$0C	WORD fd_nplanes; /* Anzahl der Bildebenen */
$0E	WORD fd_r1,fd_r2,fd_r3; /* reserviert */
	} MFDB;

MFORM (Mouse form)

Offset	Struktur
	typedef struct
	{
$00	WORD mf_xhot; /* X-Pos. Aktionspunkt */
$02	WORD mf_yhot; /* Y-Pos. Aktionspunkt */
$04	WORD mf_nplanes; /* Anzahl planes (=1) */
$06	WORD mf_fg; /* Maskenfarbe (normal: 0) */
$08	WORD mf_bg; /* Zeigerfarbe (normal: 1) */
$0A	WORD mf_mask[16]; /* Maskenform */
$2A	WORD mf_data[16]; /* Zeigerform */
	} MFORM;

MPB (Memory Parameter Block)

Offset	Struktur
	typedef struct
	{
$00	MD *mp_mfl; /* Zeiger auf "Memory Free List" */
$04	MD *mp_mal; /* Zeiger auf "Memory Allocated List" */
$08	MD *mp_rover; /* roving pointer; nur intern benötigt */
	} MPB;

OBJECT

Offset	Struktur
	typedef struct
	{
$00	WORD ob_next; /* Nummer des nächsten Objekts*/
$02	WORD ob_head; /* Nummer des ersten Kinds */
$04	WORD ob_tail; /* Nummer des letzten Kinds */
$06	UWORD ob_type; /* Art des Objekts */
$08	UWORD ob_flags; /* Verschiedene Flags */
$0A	UWORD ob_state; /* Status des Objekts */
$0C	void *ob_spec; /* Typabhängig */
$10	WORD ob_x; /* X-Position (rel. zum Parent-Objekt) */
$12	WORD ob_y; /* Y-Position (rel. zum Parent-Objekt) */
$14	WORD ob_width; /* Breite */
$16	WORD ob_height; /* Höhe */
	} OBJECT;

OHEADER (Header einer DR-Objekdatei)

Offset	Struktur
	typedef struct
	{
$00	WORD magic; /* $601A */
$02	LONG tsize; /* Größe des Textsegments */
$06	LONG dsize; /* Größe des DATA-Segments */
$0A	LONG bsize; /* Größe der BSS */
$0E	LONG ssize; /* Größe der Symboltabelle */
$12	char reserved[10]; /* alle auf 0 setzen */
	} OHEADER;

OSHEADER

Offset	Struktur

```
        typedef struct _osheader
        {
$00     UWORD     os_entry;      /* BRAnch-Instruktion zum RESET-
                                    Handler, Offset $00 */
$02     UWORD     os_version;    /* TOS-Versionsnummer,
                                    Offset $02 */
$04     void      *reseth;       /* Zeiger auf RESET-Handler,
                                    Offset $04 */
$08     struct _osheader *os_beg;
                                 /* Basisadresse des Betriebs-
                                    systems, Offset $08 */
$0C     void      *os_end;       /* Erstes nicht vom OS benutztes
                                    Byte, Offset $0C */
$10     LONG      os_rsv1;       /* reserviert, Offset $10 */
$14     GEM_MUPB  *os_magic;     /* Zeiger auf "GEM memory usage
                                    parameter block",Offset $14*/
$18     LONG      os_date;       /* TOS-Herstellungsdatum im BCD-
                                    Format, etwa $02061986 für
                                    6. Febr. 1986, Offset $18 */
$1C     UWORD     os_conf;       /* verschiedene Konfigurations-
                                    bits, Offset $1C */
$1E     UWORD     os_dosdate;    /* TOS-Herstellungsdatum im
                                    GEMDOS-Format, Offset $1E */
        /* die folgenden Felder erst ab TOS-Version 1.02 */
$20     char      **p_root;      /* Basisadresse des GEMDOS-
                                    Pools, Offset $20 */
$24     BYTE      **pkbshift;    /* Zeiger auf BIOS-interne
                                    Variable für den aktuellen
                                    Wert von "Kbshift()",
                                    Offset $24 */
$28     BASEPAGE  **p_run;       /* Adresse der Variablen, die
                                    einen Zeiger auf den aktuel-
                                    len GEMDOS-Prozeß enthält,
                                    Offset $28 */
$2C     char      *p_rsv2;       /* reserviert, Offset $2C */
        } OSHEADER;
```

PARMBLK

Offset	Struktur

```
        typedef struct
        {
$00     OBJECT    *pb_tree;      /* Zeiger auf Objektbaum */
$04     WORD      pb_obj;        /* Objektnummer */
$06     WORD      pb_prevstate;  /* Vorheriger Status */
$08     WORD      pb_currstate;  /* Neuer Status */
$0A     WORD      pb_x;          /* X-Position des Objekts */
$0C     WORD      pb_y;          /* Y-Position des Objekts */
$0E     WORD      pb_w;          /* Breite */
$10     WORD      pb_h;          /* Höhe */
$12     WORD      pb_xc;         /* X-Position des Begrenzungs-
                                    rechtecks */
$14     WORD      pb_yc;         /* Y-Position des Begrenzungs-
                                    rechtecks */
$16     WORD      pb_wc;         /* Breite des Begrenzungs-
                                    rechtecks */
$18     WORD      pb_hc;         /* Höhe des Begrenzungs-
                                    rechtecks */
$1A     LONG      pb_parm;       /* Parameter aus USERBLK */
        } PARMBLK;
```

PBDEF (Printblk Definition)

Offset	Struktur

```
        typedef struct
        {
$00     LONG pb_scrptr;     /* Zeiger auf Bildschirmanfang */
$04     WORD pb_offset;     /* dazu zu addierender Offset */
$06     WORD pb_width;      /* Bildschirmbreite in Punkten */
$08     WORD pb_height;     /* Bildschirmhöhe in Punkten */
$0A     WORD pb_left;       /* Linker Rand in Punkten */
$0C     WORD pb_right;      /* Rechter Rand in Punkten */
$0E     WORD pb_screz;      /* Auflösung */
$10     WORD pb_prrez;      /* Druckertyp Atari/Epson */
```

Offset	Struktur	
$12	LONG pb_colptr;	/* Zeiger auf Farbpalette (zum Beispiel $FF8240) */
$16	WORD pb_prtype;	/* 0: Atari Matrix monochrom, 1: Atari Matrix farbig, 2: Atari Typenrad monochrom, 3: Epson Matrix monochrom */
$18	WORD pb_prport;	/* Schnittstelle Centronics/RS 232 */
$1A	LONG pb_mask;	/* Zeiger auf Halbtonmaske */
	} PBDEF;	

PH (Program Header)

Offset	Struktur	
	typedef struct	
	{	
$00	WORD ph_branch;	/* Branch an den Anfang des Programms (0x601A) */
$02	LONG ph_tlen;	/* Länge des TEXT-Abschnitts */
$06	LONG ph_dlen;	/* Länge des DATA-Abschnitts */
$0A	LONG ph_blen;	/* Länge des BSS-Abschnitts */
$0E	LONG ph_slen;	/* Länge der Symboltabelle */
$12	LONG ph_res1;	/* reserviert; muß 0 sein */
$16	LONG ph_prgflags;	/* spezielle Flags */
$1A	WORD ph_absflag;	/* 0: Relozierungsinformationen vorhanden */
	} PH;	

PUNINFO (Physical Unit Information)

Offset	Struktur		
	typedef struct		
	{		
$00	WORD	puns;	/* Anzahl der Geräte */
$02	BYTE	pun[16];	/* diverse Flags */
$12	LONG	part_start[16];	/* Partitionsanfänge */

Offset	Struktur
$52	LONG P_cookie; /* muß "AHDI" sein */
$56	LONG *P_cookptr; /* zeigt auf das vorherige Element */
$5A	UWORD P_version; /* 0x0300 oder größer */
$5C	UWORD P_max_sector; /* maximale Sektorgröße */
$5E	LONG reserved[16];
	} PUN_INFO;

RSHDR (Resource Header)

Offset	Struktur
	typedef struct
	{
$00	UWORD rsh_vrsn; /* null */
$02	UWORD rsh_object; /* Position des Objekt-Feldes */
$04	UWORD rsh_tedinfo; /* Position der TEDINFO-Strukturen */
$06	UWORD rsh_iconblk; /* Position der ICONBLK-Strukturen */
$08	UWORD rsh_bitblk; /* Position der BITBLK-Strukturen */
$0A	UWORD rsh_frstr; /* Position der freien Strings */
$0C	UWORD rsh_string; /* unbenutzt */
$0E	UWORD rsh_imdata; /* Position der Image-Daten */
$10	UWORD rsh_frimg; /* Position der freien Images */
$12	UWORD rsh_trindex; /* Position der Objektbaumtabelle */
$14	UWORD rsh_nobs; /* Gesamtzahl der Objekte */
$16	UWORD rsh_ntree; /* Gesamtzahl der Objektbäume */
$18	UWORD rsh_nted; /* Gesamtzahl der TEDINFO-Strukturen */
$1A	UWORD rsh_nib; /* Gesamtzahl der ICONBLK-Strukturen */
$1C	UWORD rsh_nbb; /* Gesamtzahl der BITBLK-Strukturen */
$1E	UWORD rsh_nstring; /* Gesamtzahl der Strings */
$20	UWORD rsh_nimages; /* Gesamtzahl der Images */
$22	UWORD rsh_rssize; /* Gesamtlänge der RSC-Datei */
	} RSHDR;

TEDINFO

Offset	Struktur
	typedef struct {
$00	char *te_ptext; /* Zeiger auf Text */
$04	char *te_ptmplt; /* Zeiger auf Textmaske */
$08	char *te_pvalid; /* Zeiger auf Texttypmaske */
$0C	WORD te_font; /* Zeichensatz */
$0E	WORD te_resvd1; /* reserviert */
$10	WORD te_just; /* Justifikation */
$12	WORD te_color; /* Farbe des betreff. Rechtecks */
$14	WORD te_resvd2; /* reserviert */
$16	WORD te_thickness; /* Rahmen */
$18	WORD te_txtlen; /* Länge des Textes */
$1A	WORD te_tmplen; /* Länge der Textmaske */
	} TEDINFO;

USERBLK

Offset	Struktur
	typedef struct {
$00	WORD (*ub_code)(PARMBLK *);/* Zeiger auf die eigene Funktion */
$04	LONG ub_parm; /* Ein optionaler Parameter */
	} USERBLK;

VDIESC

Offset	Struktur
	typedef struct {
-$38E	LONG RESERVED6; /* (-$38E) */
-$38A	FONT_HDR *CUR_FONT; /* Zeiger auf den Header des aktuel- len Zeichensatzes(-$38A) */

Offset	Struktur
-$386	WORD RESERVED5[23]; /* (-$386) */
-$358	WORD M_POS_HX; /* X-Koordinate des Maus-"Hot spot" (-$358) */
-$356	WORD M_POS_HY; /* Y-Koordinate des Maus-"Hot spot" (-$356) */
-$354	WORD M_PLANES; /* Maus wird im Replace-Modus (1) oder im XOR-Modus (-1) gezeichnet (-$354) */
-$352	WORD M_CDB_BG; /* Maus-Hintergrundfarbe (-$352) */
-$350	WORD M_CDB_FG; /* Maus-Vordergrundfarbe (-$350) */
-$34E	WORD MASK_FORM[32]; /* jeweils abwechselnd für Vordergrund und Maske 16 Words (-$34E) */
-$30E	WORD INQ_TAB[45]; /* Informationen, die bei "vq_extnd()" zurückgeliefert werden (-$30E) */
-$2B4	WORD DEV_TAB[45]; /* Informationen, die man bei "v_opnwk()" erhält (-$2B4) */
-$25A	WORD GCURX; /* Aktuelle X-Position der Maus (-$25A) */
-$258	WORD GCURY; /* Aktuelle Y-Position der Maus (-$258) */
-$256	WORD M_HID_CT; /* Anzahl der erfolgten "Hide Mouse"-Aufrufe (-$256) */
-$254	WORD MOUSE_BT; /* Aktueller Status der Mausknöpfe (-$254) */
-$252	WORD REQ_COL[48]; /* Interne Daten für "vq_color()" (-$252) */
-$1F2	WORD SIZ_TAB[15]; /* Informationen, die bei "v_opnwk()" zurückgeliefert werden (-$1F2) */
-$1D4	WORD RESERVED4[2];
-$1D0	LONG CUR_WORK; /* Zeiger auf Attributdaten der aktuellen virtuellen Workstation (-$1D0) */
-$1CC	FONT_HDR *DEF_FONT; /* Zeiger auf den Standard-Systemzeichensatz (-$1CC) */
-$1C8	LONG FONT_RING[4]; /* Drei Pointer auf Zeichensatzlisten (verkettete FONT_HDR-Strukturen).

Offset	Struktur
	Das letzte Element enthält eine 0 als Endezeichen (-$1C8) */
-$1B8	WORD FONT_COUNT; /* Anzahl der Zeichensätze in der "FONT_RING"-Liste (-$1B8) */
-$1B6	WORD RESERVED3[45];
-$15C	CHAR CUR_MS_STAT; /* Mausstatus: Bit 0: linker Knopf Bit 1: rechter Knopf Bit 2..4: reserviert Bit 5: Bewegungsflag (1: Maus wurde bewegt) Bit 6: gesetzt: Status des rechten Knopfes hat sich geändert Bit 7: gesetzt: Status des linken Knopfes hat sich geändert (-$15C) */
-$15B	CHAR RESERVED2;
-$15A	WORD V_HID_CNT; /* Anzahl der Hide-Cursor-Aufrufe */
-$158	WORD CUR_X; /* X-Position der Maus (-$158) */
-$156	WORD CUR_Y; /* Y-Position der Maus (-$156) */
-$154	CHAR CUR_FLAG; /* <>0, wenn Mauszeiger beim nächsten VBlank neu gezeichnet werden muß (-$154) */
-$153	CHAR MOUSE_FLAG; /* <>0, wenn Maus-Interruptbehandlung eingeschaltet ist (-$153) */
-$152	LONG RESERVED1;
-$14E	WORD V_SAV_XY[2]; /* gerettete X- und Y-Koordinate des Cursors (-$14E) */
-$14A	WORD SAVE_LEN; /* Anzahl der gebufferten Bildzeilen (-$14A) */
-$148	LONG SAVE_ADDR; /* Adresse des ersten gebufferten Bytes im Bildspeicher (-$148) */
-$144	WORD SAVE_STAT; /* Bit 0: Buffer ist gültig(1) oder ungültig (0), Bit 1: Longs (1) oder Words (0) gebuffert (-$144)*/
-$142	LONG SAVE_AREA[64]; /* Buffer für Bild unter Mauszeiger ($-142) */

Offset	Struktur	
-$42	LONG USER_TIM;	/* aktueller Timer-Interrupt-Vektor; sollte zur Beendigung zu "NEXT_TIM" springen (-$42) */
-$3E	LONG NEXT_TIM;	/* alter Timer-Interrupt-Vektor (-$3E) */
-$3A	LONG USER_BUT;	/* Maustastenvektor (-$3A) */
-$36	LONG USER_CUR;	/* Mausvektor (-$36) */
-$32	LONG USER_MOT;	/* Mausbewegungsvektor (-$32) */
-$2E	WORD V_CEL_HT;	/* Zeichenhöhe (-$2E) */
-$2C	WORD V_CEL_MX;	/* maximale Cursor-Spaltenposition (-$2C) */
-$2A	WORD V_CEL_MY;	/* maximale Cursor-Zeilenposition (-$2A) */
-$28	WORD V_CEL_WR;	/* Characterzeilenbreite in Bytes (Höhe * Breite einer Pixelzeile) (-$28) */
-$26	WORD V_COL_BG;	/* Hintergrundfarbe (-$26) */
-$24	WORD V_COL_FG;	/* Vordergrundfarbe (-$24) */
-$22	LONG V_CUR_AD;	/* Adresse der aktuellen Cursorposition auf dem Bildschirm (-$22) */
-$1E	WORD V_CUR_OF;	/* Vertikaler Offset vom physikalischen Bildschirmanfang (kann mit VDI ESC 101 verändert werden) (-$1E) */
-$1C	WORD V_CUR_XY[2];	/* X- und Y-Position des Cursors (-$1C) */
-$18	CHAR V_PERIOD;	/* Blinkgeschwindigkeit des Cursors (-$18) */
-$17	CHAR V_CUR_CT;	/* Zähler für Cursorblinken (-$17) */
-$16	LONG V_FNT_AD;	/* Zeiger auf Zeichensatzdaten des Systemzeichensatzes (siehe auch VDI ESC 102). Die Zeichenbreiten müssen jeweils 8 Bytes betragen! (-$16) */
-$12	WORD V_FNT_ND;	/* ASCII-Wert des letzten Zeichens im Zeichensatz (-$12) */
-$10	WORD V_FNT_ST;	/* ASCII-Wert des ersten Zeichens im Zeichensatz (-$10) */

Offset	Struktur
-$0E	WORD V_FNT_WD; /* Breite der Fontdaten (des Fontimage) in Bytes (-$E) */
-$0C	WORD V_REZ_HZ; /* Bildschirmbreite in Pixel (-$0C)*/
-$0A	LONG V_OFF_AD; /* Zeiger auf Zeichensatz-Offset-Tabelle (s. VDI ESC 102) (-$0A) */
-$06	WORD RESERVED; /* TOS 1.00: Cursorflag (-$06) */
-$04	WORD V_REZ_VT; /* Bildschirmhöhe in Pixel (-$04)*/
-$02	WORD BYTES_LIN; /* Bytes pro Pixelzeile (-$02) */
	} VDIESC;

XBRA

Offset	Struktur
	typedef struct
	{
$00	char xb_magic[4]; /* "XBRA" = 0x58425241 */
$04	char xb_id[4]; /* vier Buchstaben lange Kennung wie beim Cookie Jar */
$08	LONG xb_oldvec; /* ursprünglicher Wert des Vektors */
	} XBRA;

XCPB

Offset	Struktur
	typedef struct
	{
$00	WORD handle; /* aus graf_handle()-Aufruf von XControl. Wichtig für v_opnvwk()!! */
$02	WORD booting; /* ungleich 0: Initialisierung/ Bootvorgang */
$04	WORD reserved; /* reserviert */
$06	WORD SkipRshFix; /* ungleich: Resourcekoordinaten bereits transformiert */

Offset	Struktur
$08	`void *reserve1; /* reserviert */`
$0C	`void *reserve2; /* reserviert */`
$10	`void cdecl (*rsh_fix)(WORD num_objs, WORD num_frstr, WORD num_frimg, WORD num_tree, OBJECT *rs_object, TEDINFO *rs_tedinfo, char *rs_strings[], ICONBLK *rs_iconblk, BITBLK *rs_bitblk, long *rs_frstr, long *rs_frimg, long *rs_trindex, struct foobar *rs_imdope);`
$14	`void cdecl (*rsh_obfix)(OBJECT *tree, WORD curob);`
$18	`WORD cdecl (*Popup)(char *items[], WORD num_items, WORD default_item, WORD font_size, GRECT *button, GRECT *world);`
$1C	`void cdecl (*Sl_size)(OBJECT *tree, WORD base, WORD slider, WORD num_items, WORD visible, WORD direction, WORD min_size);`
$20	`void cdecl (*Sl_x)(OBJECT *tree, WORD base, WORD slider, WORD value, WORD num_min, WORD num_max, void (*foo)(void));`
$24	`void cdecl (*Sl_y)(OBJECT *tree, WORD base, WORD slider, WORD value, WORD num_min, WORD num_max, void (*foo)(void));`
$28	`void cdecl (*Sl_arrow)(OBJECT *tree, WORD base, WORD slider, WORD obj, WORD inc, WORD min, WORD max, WORD *numvar, WORD direction, void (*foo)(void));`
$2C	`void cdecl (*Sl_dragx)(OBJECT *tree, WORD base, WORD slider, WORD min, WORD max, WORD *numvar, void (*foo)(void));`

Offset	Struktur
$30	void cdecl (*Sl_dragy)(OBJECT *tree, WORD base, WORD slider, WORD min, WORD max, WORD *numvar, void (*foo)(void));
$34	WORD cdecl (*Xform_do)(OBJECT *tree, WORD start_field, WORD *puntmsg);
$38	GRECT * cdecl (*GetFirstRect)(GRECT *prect);
$3C	GRECT * cdecl (*GetNextRect)(void);
$40	void cdecl (*Set_Evnt_Mask)(WORD mask, MOBLK *m1, MOBLK *m2, long time);
$44	WORD cdecl (*XGen_Alert)(WORD id);
$48	WORD cdecl (*CPX_Save)(void *ptr, long num);
$4C	void * cdecl (*Get_Buffer)(void);
$50	WORD cdecl (*getcookie)(long cookie, long *p_value);
$54	WORD Country_Code; /* Länderkennung, analog zu der im OSHEADER - allerdings in fester Abhängigkeit von der XControl-Version */
$56	void cdecl (*MFsave)(WORD saveit, MFORM *mf); } XCPB;

Anhang C:
Tabelle der Tastatur-Scancodes

Vorbemerkung: Normalerweise ist es *nicht* nötig, mit Scancodes zu arbeiten – für Control-Tastenkombinationen kann man statt dessen den ASCII-Code verwenden, für Alternate-Kombinationen bietet sich die Abfrage der XBIOS-Tastaturtabellen via "Keytbl()" an. Die direkte Abfrage von Scancodes sollte also nur bei Sondertasten (wie den Cursortasten, Funktionstasten, UNDO etc.) nötig sein.

Scancode	Deutschland	USA	England	Frankreich	Standard-VDI-Code
1	ESC				
2	1	1	1	1	1
3	2	2	2	2	2
4	3	3	3	3	3
5	4	4	4	4	4
6	5	5	5	5	5
7	6	6	6	6	6
8	7	7	7	7	7
9	8	8	8	8	8
10	9	9	9	9	9
11	0	0	0	0	0
12	ß	–	–	)	–
13	'	=	=	–	=
14	Backspace				
15	TAB				
16	Q	Q	Q	A	Q
17	W	W	W	Z	W
18	E	E	E	E	E
19	R	R	R	R	R
20	T	T	T	T	T
21	Z	Y	Y	Y	Y
22	U	U	U	U	U
23	I	I	I	I	I
24	O	O	O	O	O
25	P	P	P	P	P
26	Ü	[	[	[	[
27	+	]	]	]	]
28	Return				

Scancode	Deutschland	USA	England	Frankreich	Standard-VDI-Code
29	Control				
30	A	A	A	Q	A
31	S	S	S	S	S
32	D	D	D	D	D
33	F	F	F	F	F
34	G	G	G	G	G
35	H	H	H	H	H
36	J	J	J	J	J
37	K	K	K	K	K
38	L	L	L	L	L
39	Ö	;	;	M	;
40	Ä	'	'	\	'
41	#	‘	‘	‘	‘
42	Shift links				
43	~	\	#	@	\
44	Y	Z	Z	W	Z
45	X	X	X	X	X
46	C	C	C	C	C
47	V	V	V	V	V
48	B	B	B	B	B
49	N	N	N	N	N
50	M	M	M	,	M
51	,	,	,	;	,
52	.	.	.	:	.
53	-	/	/	=	/
54	Shift rechts				
55	nicht da	nicht da	nicht da	nicht da	PRINT SCREEN
56	Alternate				
57	Leertaste				
58	CapsLock				
59	F1				
60	F2				
61	F3				
62	F4				
63	F5				
64	F6				
65	F7				

Scancode	Deutschland	USA	England	Frankreich	Standard-VDI-Code
66	F8				
67	F9				
68	F10				
71	ClrHome				
72	↑				
73	nicht da	nicht da	nicht da	nicht da	PAGE UP
74	– (Ziffernblock)				
75	←				
77	→				
78	+ (Ziffernblock)				
79	nicht da	nicht da	nicht da	nicht da	END
80	↓				
81	nicht da	nicht da	nicht da	nicht da	PAGE DOWN
82	Insert				
83	Delete				
84	Shift-F1	Shift-F1	Shift-F1	Shift-F1	F11
85	Shift-F2	Shift-F2	Shift-F2	Shift-F2	F12
86	Shift-F3	Shift-F3	Shift-F3	Shift-F3	F13
87	Shift-F4	Shift-F4	Shift-F4	Shift-F4	F14
88	Shift-F5	Shift-F5	Shift-F5	Shift-F5	F15
89	Shift-F6	Shift-F6	Shift-F6	Shift-F6	F16
90	Shift-F7	Shift-F7	Shift-F7	Shift-F7	F17
91	Shift-F8	Shift-F8	Shift-F8	Shift-F8	F18
92	Shift-F9	Shift-F9	Shift-F9	Shift-F9	F19
93	Shift-F10	Shift-F10	Shift-F10	Shift-F10	F20
94	nicht da	nicht da	nicht da	nicht da	F21
95	nicht da	nicht da	nicht da	nicht da	F22
96	<	nicht da	\	<	F23
97	Undo				F24
98	Help				F25
99	((Ziffernblock)				F26
100	) (Ziffernblock)				F27
101	/ (Ziffernblock)				F28
102	* (Ziffernblock)				F29
103	7 (Ziffernblock)				F30
104	8 (Ziffernblock)				F31
105	9 (Ziffernblock)				F32

Scancode	Deutschland	USA	England	Frankreich	Standard-VDI-Code
106	4 (Ziffernblock)				F33
107	5 (Ziffernblock)				F34
108	6 (Ziffernblock)				F35
109	1 (Ziffernblock)				F36
110	2 (Ziffernblock)				F37
111	3 (Ziffernblock)				F38
112	0 (Ziffernblock)				F39
113	. (Ziffernblock)				F40
114	Enter (Ziffernbl.)				CTRL PRINT SCREEN
115	nicht da	nicht da	nicht da	nicht da	CTRL ←
116	nicht da	nicht da	nicht da	nicht da	CTRL →
117	nicht da	nicht da	nicht da	nicht da	CTRL END
118	nicht da	nicht da	nicht da	nicht da	CTRL PAGE DOWN
119	nicht da	nicht da	nicht da	nicht da	CTRL HOME
120	ALT 1				ALT 1
121	ALT 2				ALT 2
122	ALT 3				ALT 3
123	ALT 4				ALT 4
124	ALT 5				ALT 5
125	ALT 6				ALT 6
126	ALT 7				ALT 7
127	ALT 8				ALT 8
128	ALT 9				ALT 9
129	ALT 0				ALT 0
130	ALT ß	ALT -	ALT -	ALT)	ALT -
131	ALT '	ALT =	ALT =	ALT -	ALT =
132	nicht da	nicht da	nicht da	nicht da	CTRL PAGE UP

Die Standard-VDI-Codes beziehen sich augenscheinlich auf eine amerikanische PC-Standardtastatur. Die Tastencodes ab 131 dürfen nur für Applikationen interessant sein, die auch unter MS-DOS laufen sollen. Die darunterliegenden Codes, die man bisher mit der Atari-Tastatur nicht erzeugen kann, könnten allerdings bei einer möglichen Änderung des Tastatur-Layouts wichtig werden!

Anhang D: ASCII-Zeichensatz

Der ASCII-Zeichensatz ("American Standard Code for Information Interchange") ist ein standardisierter 7-Bit-Zeichensatz, der folglich nur 128 Zeichen definiert. Das achte Bit kann (wie beim Atari-Zeichensatz geschehen) für Erweiterungen dienen, wobei der so entstandene erweiterte ASCII-Zeichensatz ab Zeichen 128 nicht mehr normiert ist. Neuere Normen sind "ISO-Latin'"(ein 8-Bit-Zeichensatz mit europäischen Sonderzeichen, der im UNIX-Bereich an Bedeutung gewinnt) und "UNICODE" (ein 16-Bit-Zeichensatz).

Code	Zeichen	Beschreibung (in Klammern: deutsch)
0	NUL	Null (Nil)
1	SOH (Ctrl-A)	Start of Heading (Anfang des Kopfes)
2	STX (Ctrl-B)	Start of Text (Anfang des Textes)
3	ETX (Ctrl-C)	End of Text (Ende des Textes)
4	EOT (Ctrl-D)	End of Transmission (Ende der Übertragung)
5	ENQ (Ctrl-E)	Enquiry (Stationsaufforderung)
6	ACK (Ctrl-F)	Acknowledge (Positive Rückmeldung)
7	BEL (Ctrl-G)	Bell (Klingel)
8	BS (Ctrl-H)	Backspace (Rückwärtsschritt)
9	HT (Ctrl-I)	Horizontal Tabulation (Horizontal-Tabulator)
10	LF (Ctrl-J)	Line Feed (Zeilenvorschub)
11	VT (Ctrl-K)	Vertical Tabulation (Vertikal-Tabulator)
12	FF (Ctrl-L)	Form Feed (Formularvorschub)
13	CR (Ctrl-M)	Carriage Return (Wagenrücklauf)
14	SO (Ctrl-N)	Shift-out (Dauerumschaltung)
15	SI (Ctrl-O)	Shift-in (Rückschaltung)
16	DLE (Ctrl-P)	Data Link Escape (Datenübertragungsumschaltung)
17	DC1 (Ctrl-Q)	Device Control 1 (Gerätesteuerung 1)
18	DC2 (Ctrl-R)	Device Control 2 (Gerätesteuerung 2)
19	DC3 (Ctrl-S)	Device Control 3 (Gerätesteuerung 3)
20	DC4 (Ctrl-T)	Device Control 4 (Gerätesteuerung 4)
21	NAK (Ctrl-U)	Negative Acknowledge (Negative Rückmeldung)
22	SYN (Ctrl-V)	Synchronous Idle (Synchronisierung)
23	ETB (Ctrl-W)	End of Transmission Block (Ende des Datenübertragungsblocks)
24	CAN (Ctrl-X)	Cancel (Ungültig)
25	EM (Ctrl-Y)	End of Medium (Ende der Aufzeichnung)

Code	Zeichen	Beschreibung (in Klammern: deutsch)
26	SUB (Ctrl-Z)	Substitute Character (Substitution)
27	ESC (Ctrl-[)	Escape (Fluchtzeichen)
28	FS (Ctrl-\)	File Separator (Hauptgruppentrennung)
29	GS (Ctrl-])	Group Separator (Gruppentrennung)
30	RS (Ctrl-^)	Record Separator (Untergruppentrennung)
31	US (Ctrl-_)	Unit Separator (Teilgruppentrennung)
32	SP	Space (Leerstelle)
127	DEL	Delete (Löschen)

Die Zeichen zwischen 32 und 127 entsprechen den Definitionen im Systemzeichensatz.

Hex.	Dez.	Oktal	Binär	Zeichensatz	Hex.	Dez.	Oktal	Binär	Zeichensatz
60	096	140	01100000	`	90	144	220	10010000	É
61	097	141	01100001	a	91	145	221	10010001	æ
62	098	142	01100010	b	92	146	222	10010010	Æ
63	099	143	01100011	c	93	147	223	10010011	ô
64	100	144	01100100	d	94	148	224	10010100	ö
65	101	145	01100101	e	95	149	225	10010101	ò
66	102	146	01100110	f	96	150	226	10010110	û
67	103	147	01100111	g	97	151	227	10010111	ù
68	104	150	01101000	h	98	152	230	10011000	ÿ
69	105	151	01101001	i	99	153	231	10011001	Ö
6A	106	152	01101010	j	9A	154	232	10011010	Ü
6B	107	153	01101011	k	9B	155	233	10011011	¢
6C	108	154	01101100	l	9C	156	234	10011100	£
6D	109	155	01101101	m	9D	157	235	10011101	¥
6E	110	156	01101110	n	9E	158	236	10011110	ß
6F	111	157	01101111	o	9F	159	237	10011111	ƒ
70	112	160	01110000	p	A0	160	240	10100000	á
71	113	161	01110001	q	A1	161	241	10100001	í
72	114	162	01110010	r	A2	162	242	10100010	ó
73	115	163	01110011	s	A3	163	243	10100011	ú
74	116	164	01110100	t	A4	164	244	10100100	ñ
75	117	165	01110101	u	A5	165	245	10100101	Ñ
76	118	166	01110110	v	A6	166	246	10100110	ª
77	119	167	01110111	w	A7	167	247	10100111	º
78	120	170	01111000	x	A8	168	250	10101000	¿
79	121	171	01111001	y	A9	169	251	10101001	⌐
7A	122	172	01111010	z	AA	170	252	10101010	¬
7B	123	173	01111011	{	AB	171	253	10101011	½
7C	124	174	01111100	|	AC	172	254	10101100	¼
7D	125	175	01111101	}	AD	173	255	10101101	¡
7E	126	176	01111110	~	AE	174	256	10101110	«
7F	127	177	01111111	∆	AF	175	257	10101111	»
80	128	200	10000000	Ç	B0	176	260	10110000	ã
81	129	201	10000001	ü	B1	177	261	10110001	õ
82	130	202	10000010	é	B2	178	262	10110010	Ø
83	131	203	10000011	â	B3	179	263	10110011	ø
84	132	204	10000100	ä	B4	180	264	10110100	œ
85	133	205	10000101	à	B5	181	265	10110101	Œ
86	134	206	10000110	å	B6	182	266	10110110	À
87	135	207	10000111	ç	B7	183	267	10110111	Ã
88	136	210	10001000	ê	B8	184	270	10111000	Õ
89	137	211	10001001	ë	B9	185	271	10111001	¨
8A	138	212	10001010	è	BA	186	272	10111010	´
8B	139	213	10001011	ï	BB	187	273	10111011	†
8C	140	214	10001100	î	BC	188	274	10111100	¶
8D	141	215	10001101	ì	BD	189	275	10111101	©
8E	142	216	10001110	Ä	BE	190	276	10111110	®
8F	143	217	10001111	Å	BF	191	277	10111111	™

Hex.	Dez.	Oktal	Binär	Zeichensatz
C0	192	300	11000000	ij
C1	193	301	11000001	IJ
C2	194	302	11000010	א
C3	195	303	11000011	ב
C4	196	304	11000100	ג
C5	197	305	11000101	ד
C6	198	306	11000110	ה
C7	199	307	11000111	ו
C8	200	310	11001000	ז
C9	201	311	11001001	ח
CA	202	312	11001010	ט
CB	203	313	11001011	י
CC	204	314	11001100	כ
CD	205	315	11001101	ל
CE	206	316	11001110	מ
CF	207	317	11001111	נ
D0	208	320	11010000	ס
D1	209	321	11010001	ע
D2	210	322	11010010	פ
D3	211	323	11010011	צ
D4	212	324	11010100	ק
D5	213	325	11010101	ר
D6	214	326	11010110	ש
D7	215	327	11010111	ת
D8	216	330	11011000	ן
D9	217	331	11011001	ך
DA	218	332	11011010	ם
DB	219	333	11011011	ף
DC	220	334	11011100	ץ
DD	221	335	11011101	§
DE	222	336	11011110	^
DF	223	337	11011111	∞
E0	224	340	11100000	α
E1	225	341	11100001	β
E2	226	342	11100010	Γ
E3	227	343	11100011	π
E4	228	344	11100100	Σ
E5	229	345	11100101	σ
E6	230	346	11100110	µ
E7	231	347	11100111	τ
E8	232	350	11101000	Φ
E9	233	351	11101001	Θ
EA	234	352	11101010	Ω
EB	235	353	11101011	δ
EC	236	354	11101100	ϕ
ED	237	355	11101101	φ
EE	238	356	11101110	∈
EF	239	357	11101111	∩
F0	240	360	11110000	≡
F1	241	361	11110001	±
F2	242	362	11110010	≥
F3	243	363	11110011	≤
F4	244	364	11110100	⌠
F5	245	365	11110101	⌡
F6	246	366	11110110	÷
F7	247	367	11110111	≈
F8	248	370	11111000	°
F9	249	371	11111001	•
FA	250	372	11111010	·
FB	251	373	11111011	√
FC	252	374	11111100	n
FD	253	375	11111101	2
FE	254	376	11111110	3
FF	255	377	11111111	¯

Anhang F: Patchvariablen im Festplattentreiber AHDI

Die Programmdatei des Atari-Festplattentreibers AHDI enthält ab Version 3.00 mehrere zum Verändern freigegebene Patchvariablen (zu finden in "AHDI.PRG" bzw. "SHDRIVER.SYS"). Die modifizierbaren Werte beziehen sich auf GEMDOS-Pool und maximale Sektorgröße (siehe GEMDOS-Einleitung) sowie die Reservierung von Laufwerkskennungen für wechselbare Medien (siehe BIOS-Einleitung, "Unterstützung von Wechselplatten").

Die angegebenen Offsets beziehen sich auf das erste Byte der Programmdatei:

Offset	Größe	Inhalt
$28	UWORD	"Magic number" $F0AD. Nur dann, wenn hier der richtige Wert steht, dürfen die folgenden Werte verändert werden.
$2A	UWORD	Versionsnummer, zum Beispiel $300 für AHDI 3.00.
$2C	UWORD	Anzahl der zusätzlichen Einträge für den GEMDOS-Pool (normalerweise 128, wird nur in GEMDOS-Versionen vor 0.15 benutzt, siehe Erläuterungen zu "FOLDR100.PRG')
$2E	UWORD	Maximale Sektorgröße (siehe in der GEMDOS-Einleitung: "GEMDOS-Puffer")
$30	UWORD	Anzahl der Einträge in der folgenden Tabelle (in Bytes)
$32	BYTE	Anzahl der zu reservierenden Laufwerksbuchstaben, falls es sich um ein wechselbares Medium handelt (siehe BIOS-Einleitung: "Unterstützung von Wechselplatten").

Bemerkungen

Neuere AHDI-Versionen (ab 4.00) haben für die SCSI-Geräte nicht etwa eine vergrößerte Tabelle, sondern eine zweite, zusätzliche.

Anhang G:
Das IMG-Format für Rasterbilder

Eine Datei mit Bildinformationen im IMG-Format trägt immer die Namenserweiterung "IMG". Der Header der Datei hat die Gestalt:

```
typedef struct
{
     UWORD     im_version;    /* Versionsnummer der Imagedatei */
     UWORD     im_headlength; /* Länge des Headers in Words */
     UWORD     im_nplanes;    /* Anzahl der Farbebenen */
     UWORD     im_patlen;     /* Musterlänge in Byte */
     UWORD     im_pixwidth;   /* Pixelbreite in mm/1000 */
     UWORD     im_pix_height; /* Pixelhöhe in mm/1000 */
     UWORD     im_scanwidth;  /* Zeilenbreite in Pixeln */
     UWORD     im_nlines;     /* Anzahl der Scanlines */
} IMGHEADER;
```

Die Länge des Headers muß immer überprüft werden, denn in künftigen GEM-Versionen könnte sich seine Länge (z. B. zur Aufnahme einer Farbpalette) ändern. Ältere Programme, die diese Mehrinformation nicht verarbeiten können, überspringen dann einfach den Rest der Struktur. Die Länge der Muster liegt zwischen 1 und 8; bei den meisten Bildschirmen beträgt die Musterlänge zwei Bytes. Jede Scanline-Information setzt sich aus zwei Komponenten zusammen:

- Wiederholungsteil

```
typedef struct
{
     WORD sc_zero;  /* immer 0 */
     BYTE sc_ff;    /* immer 255 (=0xFF) */
     BYTE sc_cnt;   /* Anzahl der kodierten Bildzeilen */
} SCANLINE;
```

- eigentliche Bildinformation

  Die Bildinformationen werden in drei verschiedenen Kodierungen zeilenweise abgespeichert:

- Einfarbige Pixelfolgen ("Solid Run") werden als einfaches Byte gespeichert, wobei das oberste Bit den Status des Punktes bestimmt (also "an" oder "aus"). Die restlichen sieben Bits teilen mit, wie viele Bytes ausgegeben werden sollen.

- 48 gesetzte Pixel würden also als $86 ($80 + $6) kodiert, acht nicht gesetzte Pixel als $06.

- Musterfolgen ("Pattern Run") haben als erstes Byte die 0 und als zweites die Anzahl der Musterwiederholungen. Die Musterlänge steht im Header unter "im_patlen". Es folgen den ersten beiden Bytes also genau so viele Bytes, wie benötigt werden, um das Muster darzustellen.

```
typedef struct
{
     BYTE pr_zero;        /* immer 0 */
     BYTE pr_cnt;         /* Anzahl der Bytes */
     BYTE pr_data[...];   /* Musterdaten */
} PATTERNRUN;
```

- Schlecht oder nicht verkürzbare Bildinformation ("Bit String'" wird unverändert abgespeichert. Hier steht als erstes Byte eine $80 und als zweites die Anzahl der Bytes. Danach folgen dann die entsprechenden Bytes mit der unkomprimierten Bildinformation.

```
typedef struct
{
     BYTE bs_first;       /* immer 0x80 */
     BYTE bs_cnt;         /* Anzahl der Bytes */
     BYTE bs_data[..];    /* Bilddaten */
} BITSTRING;
```

Es folgen jeweils die Bildinformationen für jede kodierte Farbebene direkt hintereinander. Man sollte auch darauf achten, daß immer eine Zeile mit voller Bytezahl codiert wird, auch wenn das Bild effektiv schmaler ist.

Es können also bis zu sieben Bits "überflüssiger" Information vorliegen. Die Pixelbreite des Bildes steht im Dateikopf unter "im_scanwidth".

Leider ist das IMG-Format für die Arbeit mit Farbgrafiken nur sehr bedingt geeignet. Die GEM-Dokumentation spricht zwar von einer separaten Speicherung von Rot-, Grün-, Blau- und Grau-Informationen – wie das aber genau aussehen soll, ist unklar.

Mögliche Abhilfen:

1. Farbpaletteninformation in einer begleitenden Metadatei (siehe VDI-Funktion "v_bit_image()") ablegen.

2. Anderes, leistungsfähigeres Bildformat verwenden (die PC- und Macintosh-Welt scheinen darüber übereinzustimmen, daß TIFF das Format der Wahl ist).

3. Einige Programme unterstützen auch das sogenannte "XIMG"-Format, das von Dieter und Jürgen Geiß vorgeschlagen und bei dem der Dateikopf um Paletteninformationen erweitert worden ist:

Offset	Größe	Belegung
$10	LONG	Konstante "XIMG" als ASCII-Zeichen
$14	WORD	Farbmodell (0: RGB, 1: CMY, 2: Pantone)
$16ff	WORD	jeweils drei 16-Bit-Werte pro Farbregister, im RGB-Modell mit der Standard-VDI-Farbbeschreibung (Rot-, Grün- und Blau-Wert in Promille)

Das Feld "im_headlength" muß natürlich angesichts des längeren Dateikopfes entsprechend verändert werden.

Anhang H: Kurzeinführung in die Syntax der Programmiersprache C

Diese Kurzeinführung soll es Nicht-C-Programmierern erleichtern, die abgedruckten Beispiele und Bindings zu lesen. Sie erhebt keinen Anspruch auf Vollständigkeit und kann natürlich ein "richtiges" C-Buch nicht ersetzen – ebensowenig, wie man in sieben Tagen alle Sehenswürdigkeiten Europas besichtigen kann. Alle Beispiele beziehen sich auf die relativ neue ANSI-Norm, wie sie weitestgehend von 'Turbo-C' und vollständig von GNU-CC implementiert wird.

Operatoren

Beispiel	Ergebnis
-a	Negation von a
*a	Inhalt des Zeigers a
&a	Adresse der Variablen a
~a	logische Negation von a
++a	inkrementiere zunächst a und betrachte a dann
a++	betrachte a und inkrementiere a anschließend
--a	dekrementiere zunächst a und betrachte a dann
a--	betrachte a und dekrementiere a anschließend
sizeof (TYP)	Größe des Datentyps TYP in Bytes
sizeof (EXP)	Größe des Ausdrucks EXP in Bytes
(TYP) EXP	Wert des Ausdrucks EXP wird nach Datentyp TYP konvertiert
a + b	a plus b
a – b	a minus b
a * b	a mal b
a / b	a durch b
a % b	a modulo b
a >> b	a um b Bits nach rechts verschoben
a << b	a um b Bits nach links verschoben
a < b	ungleich Null, falls a < b
a > b	ungleich Null, falls a > b
a <= b	ungleich Null, falls a <= b
a >= b	ungleich Null, falls a >= b
a == b	ungleich Null, falls a = b

Beispiel	Ergebnis
a != b	ungleich Null, falls a ungleich b
a & b	a bitweise undiert mit b
a | b	a bitweise oderiert mit b
a ^ b	a bitweise xoderiert mit b (excl. oder)
a && b	a logisch undiert mit b
a || b	a logisch oderiert mit b
!a	NOT a
a ? EXP1:EXP2	EXP1 falls a ungleich 0, sonst EXP2
a = b	a wird der Wert von b zugewiesen
a += b	zu a wird b addiert (*)
a -= b	von a wird b subtrahiert (*)
a *= b	a wird mit b multipliziert (*)
a /= b	a wird durch b geteilt (*)
a %= b	a wird der Rest von (a/b) zugewiesen (*)
a >>= b	a wird um b Bits nach rechts geschoben (*)
a <<= b	a wird um b Bits nach links geschoben (*)
a &= b	a wird mit b undiert (*)
a |= b	a wird mit b oderiert (*)
a ^= b	a wird mit b x-oderiert (*)
EXP1 , EXP2	erst wird EXP1, dann EXP2 ausgewertet; Resultat ist der Wert von EXP2
a[]	Feld-Selektor
a->b	Element b der Struktur, auf die a zeigt
a.b	Element b der Struktur, die bei a beginnt

(*) *Ergebnis ist jeweils der (neue) Wert von a*

Datentypen

Folgende grundlegende Datentypen werden in diesem Buch verwendet:

char	hier paßt ein Zeichen hinein (Größe: 8 Bit)
BYTE	ganzzahliger Wert zwischen 0 und 255 (Größe: 8 Bit)
WORD	ganzzahliger Wert zwischen –32768 und 32767 (Größe: 16 Bit)
UWORD	ganzzahliger Wert zwischen 0 und 65535 (Größe: 16 Bit)
LONG	ganzzahliger Wert zwischen –2147483648 und 2147483647 (Größe: 32 Bit)
ULONG	ganzzahliger Wert zwischen 0 und 4294967295 (Größe: 32 Bit)

Aus diesen "skalaren" Datentypen kann man neue Typen konstruieren. Das einfachste Beispiel ist ein Array in der Form:

```
TYPNAME Variablenname[Größe]; /* Kommentar: ein Array */
```

Ein Beispiel:

```
WORD int_in[16];
```

deklariert also ein Feld mit Platz für 16 WORD-Variablen. Die Numerierung der Feldelemente beginnt im Gegensatz zu Pascal immer bei Null – die definierten Feldelemente liegen also zwischen int_in[0] und int_in[15].

C-Zeichenketten sind einfach nur Felder von "char"-Variablen, wobei das Ende durch ein Nullbyte gekennzeichnet ist.

```
char frage[] = "Was soll das?";
```

reserviert also Platz für 14 Zeichen – 13 für die Zeichen und eines für das Nullbyte.

Wichtig sind auch die Pointer, also Zeigervariablen. Bei vielen Betriebssystemaufrufen werden nicht die Daten selbst, sondern ein Zeiger darauf übergeben. Dies passiert meist dann, wenn die Information nicht in einen skalaren Datentyp paßt – zum Beispiel ein Dateiname. Eine Zeigerdeklaration sieht so aus:

```
TYPNAME *Variablenname; /* Ein Zeiger */
```

Ein Beispiel:

```
WORD *zeiger; /* Ein Zeiger auf ein WORD */
```

Zur Arbeit mit Zeigern braucht man zwei Operatoren, den "*" zum Derefenzieren und "&" zum Referenzieren.

Ein Beispiel:

```
WORD zahl;
WORD *zeiger;

zahl = 5;           /* zahl enthält den Wert 5 */
zeiger = &zahl;     /* zeiger enthält die Adresse der Variablen
                       zahl */
*zeiger = 7;        /* die Variable, auf die zeiger zeigt (also
                       zahl) wird auf 7 gesetzt */
```

Intern behandelt ein C-Compiler Felder und Zeigervariablen (fast) identisch. Dies äußert sich besonders anschaulich in der Gleichheit von

"name[index]" und "*(name+index)"

Damit stehen alle folgenden Ausdrücke für das gleiche:

"name[0]", "*name" und "0[name]"

Es genügt zu wissen, daß man in der Praxis Zeiger als Felder und umgekehrt betrachten kann.

Komplizierter wird es da schon bei den Datenstrukturen, bei denen man mehrere einfachere Datentypen zu einer gemeinsamen Struktur zusammenfaßt:

```
typedef struct
{
     WORD      ob_next;       /* Nummer des nächsten Objekts */
     WORD      ob_head;       /* Nummer des ersten Kinds */
     WORD      ob_tail;       /* Nummer des letzten Kinds */
     UWORD     ob_type;       /* Art des Objekts */
     UWORD     ob_flags;      /* Verschiedene Flags */
     UWORD     ob_state;      /* Status des Objekts */
     ULONG     ob_spec;       /* Typabhängig */
     WORD      ob_x;          /* X-Position (rel. zum Parent-
                                 Objekt) */
     WORD      ob_y;          /* Y-Position (rel. zum Parent-
                                 Objekt) */
     WORD      ob_width;      /* Breite */
     WORD      ob_height;     /* Höhe */
} OBJECT;
```

Dies würde in Pascal folgendermaßen aussehen:

```
TYPE
  OBJECT = RECORD
     ob_next   : integer;
     ob_head   : integer;
     ob_tail   : integer;
     ob_type   : integer;
     ob_flags  : integer;
     ob_state  : integer;
```

```
    ob_spec   : long_integer;
    ob_x      : integer;
    ob_y      : integer;
    ob_width  : integer;
    ob_height : integer;
  END;
```

Wer in keiner Hochsprache programmiert, wird sich mehr dafür interessieren, wie so eine Struktur im Speicher aussieht. Trifft ein Compiler auf eine Strukturdeklaration, dann reserviert er zunächst einmal einen Speicherbereich mit der Gesamtgröße der Struktur, die sich aus der Summe der Strukturkomponenten berechnet. Beispiel:

```
OBJECT box;
```

Damit ist ein 24 Bytes großer Speicherbereich für die Datenstruktur reserviert. Was passiert, wenn man auf eine Strukturkomponente zugreift?

```
box.ob_width = 10;
```

Der Compiler überlegt sich nun, wie groß alle in der Struktur davor liegenden Komponenten sind. Die Anwort ist – nicht überraschend – 20. Erzeugt wird also ein Assemblerbefehl, der ungefähr so aussieht:

```
move.w    #10, box+20
```

Das heißt, daß jede Strukturkomponente einen festen Offset innerhalb der Objektstruktur hat, die wir auch im Anhang "Wichtige Betriebssystemstrukturen" angegeben haben.

Komplizierter wird es schon bei so einer Anweisung:

```
OBJECT *boxpnt;
boxpnt->ob_width=10;
```

Hier haben wir in "boxpnt" einen Zeiger auf ein Objekt. Erzeugt wird in diesem Fall ungefähr Assembler-Code der Form:

```
move.l    boxpnt, a0
move.w    #10, 20(a0)
```

Es wird also 20 zu der Adresse addiert, die in "boxpnt" steht und das Ergebnis als Zeigervariable verwendet.

Für die vom Betriebssystem benutzten Strukturen sind die Offsets der einzelnen Komponenten natürlich fest definiert. Dies gilt allerdings im allgemeinen für C-Strukturen nicht. Ein C-Compiler kann durchaus zwischen den einzelnen Strukturkomponenten unbenutzte Bytes einfügen, um Laufzeitoptimierungen zu erreichen. Dies ist aber normalerweise bei Compilern auf Motorola-Systemen nicht der Fall – außer, es werden Füllbytes eingefügt, um einen WORD- oder LONG-Wert auf eine gerade Adresse zu verschieben.

Bitfelder sind enge Verwandte der normalen C-Strukturen – nur dieses Mal werden den einzelnen Bits eigene Namen zugewiesen:

```
typedef struct
{
     unsigned character   : 8; /* Wert zw. 0 und 255 (8 Bits) */
     signed framesize     : 8; /* Wert zw. -128 und 127 (8 Bits) */
     unsigned framecol    : 4; /* Wert zwischen 0 und 15 (4 Bits)*/
     unsigned textcol     : 4;
     unsigned textmode    : 1; /* 0 oder 1 */
     unsigned fillpattern : 3; /* Wert zwischen 0 und 7 (3 Bits) */
     unsigned interiorcol : 4; /* Wert zwischen 0 und 15 (4 Bits)*/
} bfobspec;
```

Die Art und Weise, wie ein C-Compiler diese Bits anordnet, ist vom Compiler anhängig. Überall, wo im Profibuch Bitfelder auftauchen, gelten die Angaben für "Turbo-C", das in diesem Fall folgende Aufteilung benutzen würde:

Bits	Feldelement
24..31	character
16..23	framesize
12..15	framecol
8..11	textcol
7	textmode
4..6	fillpattern
0..3	interiorcol

Funktionen

Eine C-Funktion ist – grob vereinfacht ausgedrückt – eine Reihe von Maschineninstruktionen, denen man einige Parameter übergibt und ein Ergebnis zurückerhält. Dazu ein Beispiel für eine Funktionsdeklaration in ANSI-C:

```
LONG Bconin (WORD dev);
```

Eingabeparameter ist also ein WORD-Parameter namens "dev", und zurückgeliefert wird von der Funktion ein LONG-Returnwert. Anstelle einfacher skalarer Datentypen können natürlich auch kompliziertere Datenstrukturen auftreten:

```
BPB *Getbpb (WORD dev);  /* Funktion liefert einen Zeiger auf eine
                            BPB-Struktur */
```

Im folgenden Beispiel erhält die Funktion als ersten Parameter einen Zeiger auf eine Zeichenkette:

```
WORD Fcreate (char *fname, WORD attribs);
```

Auf welche Art und Weise die Parameterübergabe erfolgt, kann für jeden C-Compiler unterschiedlich sein. Die im TOS benutzten Übergabemechanismen wurden natürlich durch den bei der Entwicklung von TOS benutzten Compiler ("Alcyon-C") geprägt:

- Parameter werden von rechts nach links nacheinander auf dem Stack abgelegt.
- Der Rückgabewert wird im Datenregister D0 übergeben.

"Turbo-C" hingegen übergibt normalerweise mehrere Parameter in Daten- und Adreßregistern, was natürlich zu kürzeren und schnelleren Maschinencodes führt. Normalerweise kümmern sich die Standardbibliotheken um die notwendigen Anpassungen. Immer dann, wenn man direkt Betriebssystemfunktionen aufruft (zum Beispiel bei der Benutzung von "shell_p" oder bei der Programmierung von XControl-Erweiterungen), muß man 'Turbo-C' durch das Schlüsselwort "cdecl" anweisen, die "alte" Form der Parameterübergabe zu benutzen. Besitzer anderer Compiler können dieses Schlüsselwort gefahrlos ignorieren.

"void" und "const"

Bei vielen Funktionen taucht auch das Schlüsselwort "void" auf. Je nach Kontext hat es eine geringfügig andere Bedeutung:

- Allein vor einer Funktionsdeklaration stehend, gibt es an, daß die Funktion keinen sinnvollen Wert zurückliefert. Beispiel:

  ```
  void Bconout (WORD dev, WORD c);
  ```

- Als einziger Parameter einer Funktion legt es fest, daß die Funktion tatsächlich gar keine Parameter erhält. Beispiel:

```
DTA *Fgetdta (void);
```

- Im Zusammenhang mit einer Zeigervariablen gibt es an, daß der Zeiger auf keinen bestimmten Datentyp zeigt. Solche "void"-Zeiger können jeder anderen Zeigervariablen zugewiesen werden. Beispiel:

```
void *Malloc (LONG amount);
char *zeichenkette;
OBJECT *ein_objekt;


zeichenkette = Malloc (20 * sizeof (char)); /* Platz für 20
                                               Zeichen */
ein_objekt = Malloc (sizeof (OBJECT));      /* Platz für
                                               OBJECT-Struktur*/
```

Anmerkung: Die Syntax in den ersten beiden Beispielen ist tatsächlich etwas merkwürdig. Logischere Schreibweisen wie

```
Bconout (WORD dev, WORD c);
```

bzw.

```
DTA *Fgetdta ();
```

waren allerdings aus Gründen der Rückwärtskompatibilität zum "alten" C nicht möglich.

Das Wörtchen "const" kommt meist bei Parametern von Funktionen vor und legt fest, daß die aufgerufene Funktion den Inhalt des Parameters nicht verändert. Beispiel:

```
void print_it (const char *zeichen);
```

Hier bliebe also die Zeichenkette, auf die "zeichen" zeigt, garantiert intakt. Aus Zeitgründen und wegen fehlender Dokumentation war es uns leider nicht möglich, diese Eigenschaft für jede einzelne Betriebssystemfunktion zu verifizieren. Dort, wo das Wort auftaucht, darf man aber davon ausgehen, daß die Daten tatsächlich nicht verändert werden.

Kontrollstrukturen

Und schließlich noch eine kurze Übersicht über die Kontrollstrukturen der Sprache C:

```
if (Ausdruck)
     Anweisung;
```

Wenn der Ausdruck wahr (also ungleich Null) ist, wird die Anweisung ausgeführt. Überall dort, wo eine Anweisung stehen darf, kann auch ein ganzer Anweisungsblock eingefügt werden:

```
if (Ausdruck)
{
     Anweisung1;
     Anweisung2;
}
```

Zusätzlich kann ein "else"-Zweig angefügt werden:

```
if (Ausdruck)
     Anweisung1;
else
     Anweisung2;
```

Für Fallunterscheidungen gibt es "switch":

```
switch (Ausdruck)
{
     case Konstante1:    Anweisungen...
     case Konstante2:    Anweisungen...

     default:            Anweisungen...
}
```

Zunächst wird der Wert des Ausdrucks ermittelt. Stimmt das Ergebnis mit einer der angegebenen Konstanten überein, dann werden alle folgenden Anweisungen bis zum ersten "break" bzw. bis zum Ende der switch-Struktur ausgeführt. Ist der errechnete Wert mit keiner der Konstanten identisch, wird bei "default" fortgefahren. Üblicherweise sehen daher switch-Anweisungen so aus:

```
switch (wert)
{
     case KONSTANTE1:
     case KONSTANTE2:
          Anweisungen;
          break;
```

```
     case KONSTANTE3:
          Anweisungen;
          break;

     default:
          /* Fehlerbehandlung */
          break;
}
```

C kennt mehrere Typen von Schleifen, die einfachste davon ist die while-Schleife:

```
while (Ausdruck)
     Anweisung;
```

Die Anweisung (oder der Anweisungsblock) wird also so lange ausgeführt, bis der Ausdruck nicht mehr wahr ist. Diese Form der Schleife ist abweisend, die Anweisung wird also dann gar nicht ausgeführt, wenn der Ausdruck gleich zu Beginn falsch ist. Anders bei:

```
do
     Anweisung;
while (Ausdruck);
```

In diesem Fall erfolgt der erste Test erst nach dem ersten Schleifendurchlauf. Sowohl leistungsfähig als auch schwer lesbar ist die for-Schleife:

```
for (Ausdruck1; Ausdruck2; Ausdruck3)
     Anweisung;
```

ist eine Abkürzung für:

```
Ausdruck1;          /* Initialisierung */
while (Ausdruck2)   /* Abbruchbedingung */
{
     Anweisung;
     Ausdruck3;     /* Schleifenzähler weiterführen */
}
```

Dazu ein Beispiel:

```
for (i = 0; i < 10; i++)
     workin[i] = 0;
```

ist identisch mit:

```
i = 0;
while (i < 10)
{
     workin[i] = 0;
     i++;
}
```

Es sei noch erwähnt, daß jeder der Ausdrücke und auch die Schleifenanweisung leer sein können:

```
     for (;;)                          /* Endlosschleife */
          Anweisung;
```

oder

```
     for ( ; i < 10; workin[i++] = 0) /* leere Anweisung */;
```

Zum Abschluß noch "break" und "continue":

break: Ausführung der Schleife beenden und bei der nächsten auf die Schleife folgenden Anweisung fortsetzen.

continue: Beginn des nächsten Schleifendurchlaufs erzwingen.

Anhang I:
Nützliche Werkzeuge für Programmierer

Neben den "großen" Entwicklungssystemen gibt es eine ganze Reihe von nützlichen Tools, die bei der Programmentwicklung unter TOS helfen können. Die folgende Auswahl erhebt keinen Anspruch auf Vollständigkeit, sondern spiegelt nur die Vorlieben der Autoren wieder. Speziell die Programme "SysMon" und "TempleMon" seien jedem engagierten Programmierer ans Herz gelegt – diese Werkzeuge machen das Debuggen von systemnahen Programmen eigentlich erst möglich!

BigScreen

BigScreen vom Profibuch-Autor Julian Reschke ermöglicht es, auf herkömmlichen ST-, STE- oder TT-Systemen eine größere Bildfläche zu simulieren. Damit ist es möglich, Programme auf Auflösungsunabhängigkeit zu testen oder auch ohne Grafikkarte Versuche in Farbauflösungen zu unternehmen. BigScreen läuft in allen Standardauflösungen des ST, STE und TT und erfordert eine TOS-Version größer gleich 1.04. Man bekommt es (zusammen mit Stefan Eissings Drucker-Spooler "SPEX") bei der

SciLab GmbH
Isestraße 57
D-W2000 Hamburg 13

SysMon

SysMon ist ein System-Monitor, der genau an der Schnittstelle zwischen Betriebssystem und anderen Programmen sitzt, die mit diesem Betriebssystem arbeiten. Der SysMon ermöglicht es, die Aufrufe an das Betriebssystem mitzuprotokollieren. Er ist auch in der Lage, den Zustand einzelner Systemresourcen – wie Speicherblöcke und Systemvektoren – anzuzeigen. Monitor bedeutet aber auch, daß der Zustand des Systems nur beobachtet und nicht verändert werden kann. SysMon gliedert sich in zwei generelle Teile. Der erste Teil kann die Aufrufe von Systemfunktionen mitprotokollieren (der Tracer), und der zweite Teil ist für die Anzeige von Systemresourcen zuständig (der Monitor).

Der Tracer unterstützt alle dokumentierten Systemfunktionen und ist in weiten Bereichen konfigurierbar. Er kann alle Aufrufe von Systemfunktionen durch das Betriebssystem oder

von Programmen mitprotokollieren. Einzelne Systemaufrufe können bei der Ausgabe unterdrückt und einzelne Programme vom Tracen ausgenommen werden. Die Ausgaben können über verschiedene Einstellungen den jeweiligen Bedürfnissen angepaßt werden. Es sind sehr viele Erweiterungen in den Tracer eingebaut. Es werden zum Beispiel alle bekannten Message-Protokolle, die zur Kommunikation zwischen Applikationen und den quasiparallelen Accessories dienen, unterstützt. Auch werden alle definierten Konstanten in Textform ausgegeben. Die Ausgaben des Tracers können auf einer zweiten Bildschirmseite geschehen, auf die RS-232-, Centronics- oder MIDI-Schnittstelle ausgegeben oder in einer Datei mitprotokolliert werden.

Der Monitor zeigt mehrere Informationen wie Speicherblöcke, Systemvektoren und Strukturen, die teilweise nur systemintern dokumentiert sind. Die Systemvektoren bestimmen, welche Programme auf einzelne Funktionen des Systems Einfluß nehmen. Die Speicheranzeige zeigt nicht nur die Länge der einzelnen Bereiche, sondern macht auch Aussagen über deren Inhalt und das Programm, das den Bereich angefordert hat. Außerdem werden sämtliche im Profibuch dokumentierten Systemvariablen und Strukturen angezeigt.

Beim Starten installiert sich SysMon in den Systemvektoren und untersucht den Speicher nach vorher geladenen Programmen. Wird SysMon dann über eine bestimmte Tastenkombination aktiviert, kann man alle folgenden Systemaufrufe dokumentieren. Da SysMon ziemlich viel Speicher benötigt (ca 200 KB) und somit auf Rechnern mit wenig Speicher nicht permanent geladen werden kann, ist er in der Lage, sich selbst wieder aus dem System zu entfernen.

Durch diese Möglichkeiten eignet sich SysMon vorzüglich zum Debuggen von eigenen Programmen oder zur Analyse von fremden Programmen. Auch früher besonders schwer zu debuggende Probleme, wie die Kommunikation zwischen verschiedenen Prozessen, sind mit SysMon schnell und einfach zu bewerkstelligen.

SysMon arbeitet auf allen Atari-Rechnern der ST-, STE- und T-Baureihe und unter allen bislang veröffentlichten TOS-Versionen. Da ein Monitor selber nicht Systemfunktionen, die er beobachtet, zur Ausgabe benutzen kann, werden eigene Funktionen benutzt. Diese unterstützen alle Standard-Auflösungen, wobei auch Anpassungen an einige zusätzliche Grafikkarten vorgenommen wurden.

Es existiert eine Schnittstelle zum Maschinenspache-Monitor TempleMon, die es ermöglicht, die Analyse auch auf die Assembler-Ebene auszudehnen und einzelne Aufrufe schrittweise abzuarbeiten. SysMon kostet 149,– DM zzgl. Porto und Verpackung. Er ist zu beziehen bei

OverScan Gbr
Säntisstraße 166
D-W1000 Berlin 48

Das umfangreiche Manual erklärt die Benutzung und Möglichkeiten des SysMon und bietet neben einer Einführung in den Funktionsablauf des Atari-Betriebssystems und einer Anleitung zum "Auflösungsunabhängigen Programmieren" auch ein größeres Kapitel über Tips, Tricks und Fallstricke bei der Programmierung der Atari-Rechner. Für registrierte Nutzer der Versionen 1.0.x gibt es eine günstige Updatemöglichkeit.

TempleMon

TempleMon ist ein speicherresidentes Monitorprogramm, das Fehlermeldungen (Exceptions) der CPU abfängt, und die Möglichkeit der Betrachtung und Modifikation des CPU-Zustands zum Fehlerzeitpunkt bietet. Bei geeigneter Änderung kann oft das unterbrochene Programm fortgesetzt werden, um z. B. noch eine Sicherung des gerade bearbeiteten Dokumentes vorzunehmen. Ferner ist auch das Disassemblieren von Speicherbereichen und das Tracen von Programmen vorgesehen. Bei der Verwendung von TempleMon sind Kenntnisse über Aufbau und Programmierung der CPU empfehlenswert.

TempleMon ist ein "Public Domain"-Programm und kann über verschiedene Mailboxen oder PD-Vertriebe bezogen werden. Die Autoren Thomas Tempelmann und Johannes Hill bieten weiterhin die Möglichkeit an, sich als Anwender registrieren zu lassen. Dazu schicken Sie bitte DM 30.– mit ausreichend frankiertem (für einen Brief ca. 100g) und vollständig beschriftetem Rückumschlag Format C5 an

Johannes Hill
Alicenstraße 30
D-W6100 Darmstadt 1.

Sie bekommen dann ein Handbuch zurückgeschickt. Haben Sie zusätzlich noch eine formatierte Diskette beigelegt, dann gibt es auch noch die neueste Programmversion.

Wega-Library

Das "WEGA" Developer Toolkit ist eine C-Library, die Funktionen zur komfortblen Handhabung von Dialogboxen (auch in Fenstern) enthält. Nähere Informationen gibt es beim Profibuch-Autor

Dietmar Rabich
Koppelbusch 37 (bis 31.12.1991: Dövelingsweg 2)
D-W4408 Dülmen

Anhang J:
Die Hardware des ATARI STE

Auf der Atari-Messe in Düsseldorf stellte Atari im September 1989 einen neuen Computer der ST-Serie vor. Er befindet sich im gleichen Gehäuse wie der "1040STF(M)" und fällt damit ebenfalls unter die Kategorie Tastaturcomputer.

Als Produktbezeichnung hat Atari die Modellbezeichnung "STE" gewählt, was darauf hindeuten soll, daß es sich hier im Prinzip um einen "normalen ST" handelt, der jedoch in einigen Punkten erweitert wurde (dafür das E in STE!).

Bevor es um Details zu den einzelnen Erweiterungen geht, soll hier in Kurzfassung aufgeführt werden, was denn beim STE anders als bei seinen "Brüdern" ist (siehe hierzu auch das Blockschaltbild Abbildung 0.2 im Kapitel "Einführung" des Teils "Die Hardware des ST").

Die Erweiterungen in Kurzfassung

- Leider nicht anders als bei den "normalen STs" (ausgenommen seien hier die MEGA STs) ist die Tastatur des STE! Sie ist die gleiche wie die der Vorgänger.

Wesentliche Erweiterungen und Änderungen sind jedoch bei der Video-Hardware vorgenommen worden:

- Der BLITTER ist im STE serienmäßig vorhanden!

- Für jede der drei Primärfarben Rot, Grün und Blau existieren in den 16 Farbregistern nun *vier* statt bisher drei Bits zur Farbauswahl. Damit stehen (ohne besondere Programmiertricks) in der niedrigen Auflösung dann 16 Farben aus $2^{(4+4+4)} = 2^{12} = 4096$ Farbtönen zur Verfügung!

- Für GENLOCK-Anwendungen (Mischen von Videosignalen von z. B. Videokamera oder -Recorder mit dem ST-Videosignal) ist der STE von außen nun voll synchronisierbar.

- Ein zusätzliches STE-Video-Base-Register für das Low-Byte der Anfangsadresse des Video-RAMs erlaubt nun die Positionierung des Bildschirmspeichers im RAM auf Word-Grenzen (bei den "normalen STs" fehlte das Register für das Low-Adreßbyte, und somit konnte der Bildschirmspeicher im RAM immer nur an einer 256-Byte-Grenze beginnen!). Im STE ist damit nun vertikales Finescrolling möglich.

- Zur Unterstützung von horizontalem Finescrolling besitzt der STE ein eigenes HSCROLL-Register.

Außerdem ist der STE nun "zugänglicher" geworden.

- Zusätzlich zum vom ST bekannten Maus- und Joystickanschluß über den Tastaturprozessor hat Atari dem STE zwei neue Portanschlüsse mitgegeben. Über Adapter lassen sich daran bis zu *vier* weitere Joysticks anschließen. Außerdem kann an diese Ports ein Light-Pen bzw. eine Light-Gun für Spiele angeschlossen werden. Für Hardware-Bastler natürlich besonders interessant: Acht dieser Portleitungen an den neuen Joystickbuchsen lassen sich auch als Ausgangsports "mißbrauchen"!

- Bisher gab es bei den STs nur digitale Ein- und Ausgänge. Beim STE stehen nun erstmals an den zwei neuen Portanschlüssen auch analoge Eingänge zur Verfügung, an die je zwei Paddles (Drehregler) angeschlossen werden können.

Auch für die Ohren hat der STE einiges zu bieten:

- Zur Tonerzeugung stehen zusätzlich zum ST-Soundchip (PSG) nun zwei PCM-Soundkanäle (Stereo-Sound möglich) mit je 8-Bit Auflösung und eigenen Ausgängen zur Verfügung. Die Sounddaten können im DMA-Betrieb abgespielt werden.

- Die Steuerung der Lautstärke, Balance und die Klangbeeinflussung der neuen Soundkanäle erfolgt über einen eigenen Chip. Dieser Chip wird über eine zusätzliche serielle Schnittstelle im STE, das sogenannte MICROWIRE™-Interface angesteuert. Die Datenübertragungsgeschwindigkeit auf dieser seriellen Schnittstelle beträgt 1 MBit/s!

Um alle diese neuen Komponenten entsprechend ansteuern und einsetzen zu können, waren natürlich Änderungen am Betriebssystem erforderlich:

- Die zusätzlichen Routinen belegen mehr ROM-Speicherplatz. Deshalb kommt man mit 192 KByte ROM im STE nicht mehr aus. Der bisher benutzte Adreßraum für das TOS von $FC 0000 – $FE FFFF ist somit ebenfalls zu klein geworden! Das STE-TOS befindet sich deshalb in zwei 1 MBit-EPROMS ab Adresse $E0 0000 und reicht bis $E3 FFFF!

Zusätzliche Verbindungen zur Außenwelt

Wie ja schon zuvor erwähnt, besitzt der STE einige zusätzliche Anschlußmöglichkeiten. Als augenfälligste Ergänzungen bei den neu hinzugekommenen Anschlüssen sind die zwei zusätzlichen Buchsen für Joystick/Paddles/Light-Gun bzw. Light-Pen zu nennen.

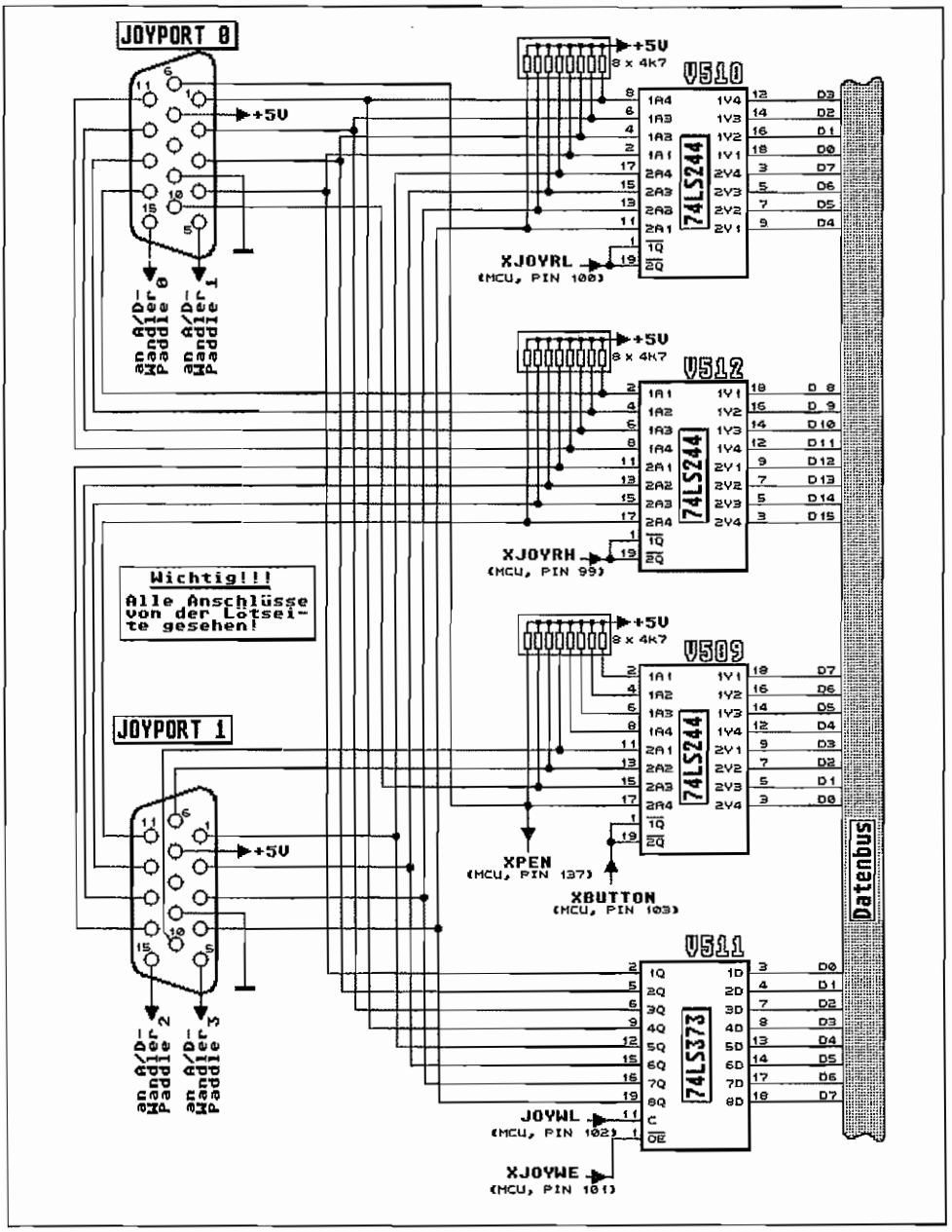


Abb. J.1: Schaltung der zusätzlichen Joystickports im STE

Zwei Buchsen für vier Joysticks – und noch einiges mehr

Nur drei zusätzliche Chips in der STE-Hardware erlauben auf einfache Weise den Anschluß von vier zusätzlichen Joysticks (auf die Paddles gehen wir später noch ein!). Dabei lassen sich an jede der beiden 15poligen Subminiatur-Buchsen zwei vollwertige Joysticks über einen entsprechenden Adapter anschließen. Die Pinbelegung der beiden Buchsen und die Anschaltung der Joysticks über die zusätzlichen Chips zeigt die Abbildung J.1.

Und nun zur Funktionsweise:

Die Richtungskontakte der Joysticks und die Kontakte der Fire-Buttons müssen gegen Masse schließen, um eine Reaktion auszulösen. Wenn die Kontakte nicht geschlossen sind, liegen die Anschlüsse über Pull-up-Widerstände an +5V.

Die Portanschlüsse sind auf die Eingänge der drei Bus-Treiberbausteine V509/V510/V512 des Typs 74LS244 geführt. Dabei werden die Kontakte für die vier Fire-Buttons vom Bustreiber V509 "bedient", während die Richtungskontakte der vier Joysticks an V510 und V512 auflaufen.

Die Bus-Treiberbausteine arbeiten nach folgendem einfachen Prinzip: Liegt am _1Q- bzw. _2Q-Anschluß ein High-Pegel, sind die Ausgänge 1Q – 1Q4 bzw. 2Q1 – 2Q4 im hochohmigen Zustand.

Geht der _1Q- bzw. _2Q-Anschluß des Bustreibers auf Low, so wird der Logik-Pegel der Eingangsanschlüsse 1A1 – 1A4 bzw. 2A1 – 2A4 an den entsprechenden 1Q1 – 1Q4 bzw. 2Q1 – 2Q4-Ausgängen präsentiert. In unserem Fall gelangen die Logikpegel der Joysticks also über die Bustreiber auf den Datenbus!

Dabei werden die Richtungsinformationen von Joystick 0 (Joyport 0) und Joystick 2 (Joyport 1) auf die unteren acht Datenleitungen des Datenbusses abgebildet. Die Joysticks 1 (Joyport 0) und 3 (Joyport 1) legen die Richtungsinformationen auf den oberen acht Datenleitungen des 16 Bit-Datenbusses ab.

Das "Einlesen" der Joystick-Richtungsinformation erfolgt also mit einem Low-Impuls auf den Leitungen XJOYRL und XJOYRH. Diese Leitungen kommen im STE vom MCU-Baustein, in dem die Funktionen von GLUE und MMU zusammengefaßt worden sind. Durch einen Word-Lesezugriff auf die Adresse $FF 9202 wird dieser Low-Impuls auf den Leitungen XJOYRL und XJOYRH erzeugt und somit der Zustand der Joystick-Richtungstasten direkt über die Bustreiber auf den Datenbus gegeben. Da die CPU ja gerade einen Lesezugriff durchführt, werden so die Richtungsinformationen der vier Joysticks mit einem Wordzugriff direkt eingelesen!

Das Einlesen der Fire-Button-Informationen erfolgt nach dem gleichen Schema. Jedoch muß dazu auf der XBUTTON-Leitung ein Low-Impuls erzeugt werden. Das geschieht durch entsprechende Dekodierung der Adreßleitungen der CPU durch den MCU-Baustein bei einem Word-Lesezugriff auf die Adresse $FF 9000. Die unteren vier Bits des Datenwords an Adresse $FF 9000 geben damit den Zustand der vier Fire-Buttons wieder.

Somit ergeben sich im Adreßraum des STE zwei Adressen, an denen die Zustände der vier Joysticks direkt abgefragt werden können!

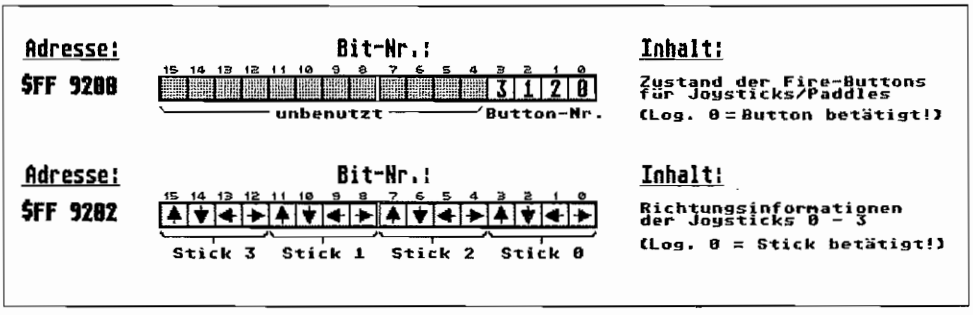


Abb. J.2: Die Adressen der neuen Joystickports

Wie im Schaltungsauszug in Abbildung J.1 dargestellt ist, wird vom Joyport 1 die Fire-Button-Leitung des Joysticks 0 (Pin 6 der Buchse) noch zusätzlich über die XPEN-Leitung zur MCU geführt. Über diese Leitung gibt der Light-Pen bzw. die Light-Gun einen Low-Impuls ab, wenn der Elektronenstrahl beim Beschreiben der Bildschirmfläche jene Stelle erreicht, die vom Light-Pen/Light-Gun gerade abgetastet wird.

Atari gibt für die Genauigkeit der abgetasteten Bildschirmposition bei Betrieb mit Light-Pen/-Gun folgende Toleranzen an:

- In vertikaler Richtung ist die Position auf ein Pixel genau.
- In horizontaler Richtung ergeben sich je nach Betriebsart unterschiedliche Toleranzwerte:

Modus	Toleranz
320 x 200	4 Pixel
640 x 200	8 Pixel
640 x 400	16 Pixel

Diese Toleranzen gelten unabhängig von der Qualität des verwendeten Light-Pens/-Gun. Auch ein hochwertigerer Light-Pen wird die angegebenen Toleranzwerte nicht weiter drücken können.

Die X- und Y-Position des Lightpens ist aus folgenden Registern mit einem Wordzugriff (als 16 Bit Wert) zu "erfahren":

Adresse	Label	Inhalt
$FF 9220	XPEN	X-Position des Lightpens
$FF 9222	YPEN	Y-Position des Lightpens

Dabei ist zu beachten, daß die Positionswerte im XPEN-Register in Pixel für den Lowres-Modus (320x200 Pixel) geliefert werden. Um korrekte Werte im Midres-Modus (640x200 Pixel) zu erhalten, ist der Wert im XPEN-Register um eine Bitposition nach links zu shiften. Bei Highres-Betrieb (640x400 Pixel) ist der XPEN-Wert um zwei Bits nach links zu shiften, um die korrekte Positionsangabe zu erhalten!

Keine Einbahnstraße für Daten – Joystick-Ports als Ausgänge

Bis jetzt war immer nur von drei der vier zusätzlichen Chips an den Joystickports des STE die Rede. Im Schaltungsauszug befindet sich jedoch noch ein weiterer Chip im Joystickinterface. Der Baustein V511 ist ein sogenanntes Oktal-Latch.

Die Funktionsweise ist folgende:

Mit der ansteigenden Flanke eines Signals am C-Anschluß werden die Logikpegel der Eingänge 1D – 8D in dem Baustein gespeichert. Bei Low-Pegel am _OE-Anschluß erscheinen die gespeicherten Informationen an den Ausgängen 1Q – 8Q. Ist der _OE-Anschluß des Bausteins jedoch auf High-Pegel, so gehen alle Ausgänge 1Q – 8Q in den hochohmigen Zustand.

Im STE sind die acht Eingänge des Oktal-Latches an die acht niederwertigen Leitungen des Datenbusses angeschlossen, und die acht Ausgänge des Bausteins V511 führen an die Anschlüsse der Joysticks 0 (Joyport 0) und 2 (Joyport 1). Vom MCU-Baustein steuern die beiden Leitungen JOYWL und XJOYWE die Anschlüsse C und _OE des Latches nun so an, daß bei einem *Schreibzugriff* auf die Adresse $FF 9202 (Label JOYDIR) die unteren acht Bits der geschriebenen Information als Logikpegel an den Joystickanschlüssen 0 und 2 ausgegeben werden. Die Pegel bleiben so lange "stehen", bis ein erneuter Schreibzugriff auf diese Adresse erfolgt oder bis auf Adresse $FF 9202 lesend zugegriffen wird!

Damit lassen sich also acht Bits an Informationen blitzschnell (so schnell die CPU kann) parallel ausgeben!

Von Analog nach Digital – Paddles am STE

Schon bei den 8-Bit-Ataris (Serie 800/800XL/130XE) war es möglich, nicht nur Schalterzustände von z. B. Joysticks abzufragen, sondern stufenlose Richtungs- oder sonstige Informationen an den Computer weiterzugeben. So lassen sich mit sogenannten Paddles, das sind Drehregler, stufenlos Richtungsinformationen o. ä. einstellen. Je weiter der Regler aufgedreht ist, desto höher ist der übermittelte Wert. Die Drehregler bestehen aus veränderbaren Drehwiderständen mit einem Widerstandswert von >470 kOhm (Atari-Paddles haben 680 kOhm-Drehwiderstände im Vollausschlag). Durch eine Elektronik werden diese analogen Widerstandswerte der Drehregler in digitale Zahlenwerte umgewandelt.

Im STE hat Atari wieder auf diese Steuerungsmöglichkeit zurückgegriffen und erlaubt den Anschluß von je zwei Paddles an jedem Joyport. Damit besteht sogar die Möglichkeit, sogenannte analoge Joysticks zu verwenden, die statt Richtungskontakten über veränderbare Widerstände verfügen und so eine stufenlose Richtungsangabe erlauben!

Wie der Schaltungsauszug in Abbildung M.3 zeigt, existiert für jeden der vier möglichen Paddleanschlüsse eine eigene Schaltung. Dabei handelt es sich um sogenannte monostabile Multivibratoren oder auch Monoflops. Im STE werden dazu sogenannte Timer-ICs vom Typ "LM 556" verwendet, wobei je zwei Monoflops in einem IC-Gehäuse untergebracht sind.

Monoflops haben die "Angewohnheit", durch einen Impuls von ihrer Ruhelage (Ausgangspegel in unserem Fall auf Low) für eine bestimmte Zeit in die Arbeitslage (Ausgangspegel in unserem Fall dann High) "umgekippt" zu werden. Die Dauer, in der das Monoflop in der Arbeitslage "verharrt", ist abhängig von einem Zeitglied, das aus einem Widerstand besteht, über den ein Kondensator aufgeladen wird. Erreicht die Ladespannung an dem Kondensator einen bestimmten Schwellwert, so wird das vom Monoflop-Baustein über einen Fühleranschluß "bemerkt", und das Monoflop kippt zurück in seine Ruhelage.

Die Dauer, in der das Monoflop in Arbeitslage bleibt, ist zum einen abhängig vom Kondensatorwert (C_T im Schaltungsauszug) und zum anderen vom Wert des Ladewiderstands. Beim STE wird der Ladewiderstand im Prinzip durch den Widerstand eines Paddles gebildet. Da dieser Widerstandswert veränderbar ist, läßt sich so auch die Zeitdauer der Arbeitslage des Monoflops direkt beeinflussen. Das Prinzip der Umwandlung von Widerstandswerten (der Paddles) in im STE verwendbare Zahlenwerte ist nun folgendes (siehe auch den Schaltungsauszug in Abbildung J.3):

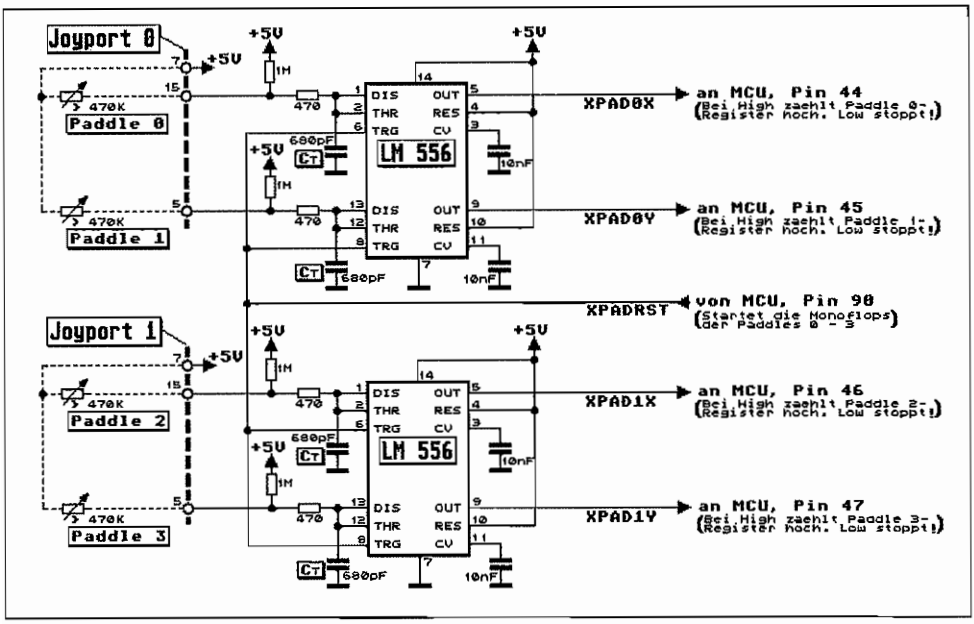


Abb. J.3: Einbindung der Paddles beim STE

Für jeden Paddle existiert ein Zähler in der STE-MCU. In regelmäßigen Abständen (ca. alle 0,52 ms) werden alle vier Zähler auf Null gesetzt und gleichzeitig die vier Monoflops mit einem Impuls über die XPADRST-Leitung in Arbeitslage "gekippt". Die vier Zähler werden in einem bestimmten Takt weitergezählt (ca. 500 KHz im STE). Die Monoflops kippen je nach Widerstandswert der Paddles nach gewisser Zeit wieder in ihren Ruhezustand zurück und halten durch diesen Signalwechsel an ihren Ausgängen (Leitungen XPAD0X – XPAD1Y im Schaltplanauszug) "ihren" Zähler in der MCU an.

Der so gewonnene Zählerstand ist damit direkt abhängig vom eingestellten Widerstandswert des Paddles und damit von dessen Position! So setzt man analoge Widerstandswerte in Zahlenwerte um. Die vier Paddle-Zähler-Register sind unter den folgenden Adressen im Speicherraum des STE abfragbar:

Adresse	Label	Inhalt
$FF 9210	PADDL0	Position des Paddle 0
$FF 9212	PADDL1	Position des Paddle 1
$FF 9214	PADDL2	Position des Paddle 2
$FF 9216	PADDL3	Position des Paddle 3

Die Register sind zwar 16 Bit breit. Es ist jedoch nur das Low-Byte signifikant (es reicht also aus, das entsprechende Low-Byte zu lesen!). Damit ergeben sich mögliche Paddle-Werte zwischen 0 und 255, wobei aber Werte über 200 nur mit Drehwiderständen mit einem maximalen Endwert von ca. 1 MOhm erreicht werden.

Mit nachfolgender Faustformel kann für einen gegebenen Paddle-Widerstandswert (in der Formel mit R_p bezeichnet und in kOhm einzusetzen) der entsprechende Paddle-Registerwert berechnet werden:

$$\text{RegWert} = \frac{394}{1 + 1000/R_p} \quad [\ R_p \text{ in kOhm! }]$$

Für die Fire-Buttons der Paddles werden die Fire-Button-Anschlüsse der Joysticks verwendet und können somit im Register JOYBUTT an Adresse $FF 9200 abgefragt werden. Nachfolgend noch ein Verdrahtungsvorschlag für die Adaption von zwei handelsüblichen Joysticks mit 9poligen Anschlußbuchsen an einen STE-Joyport:

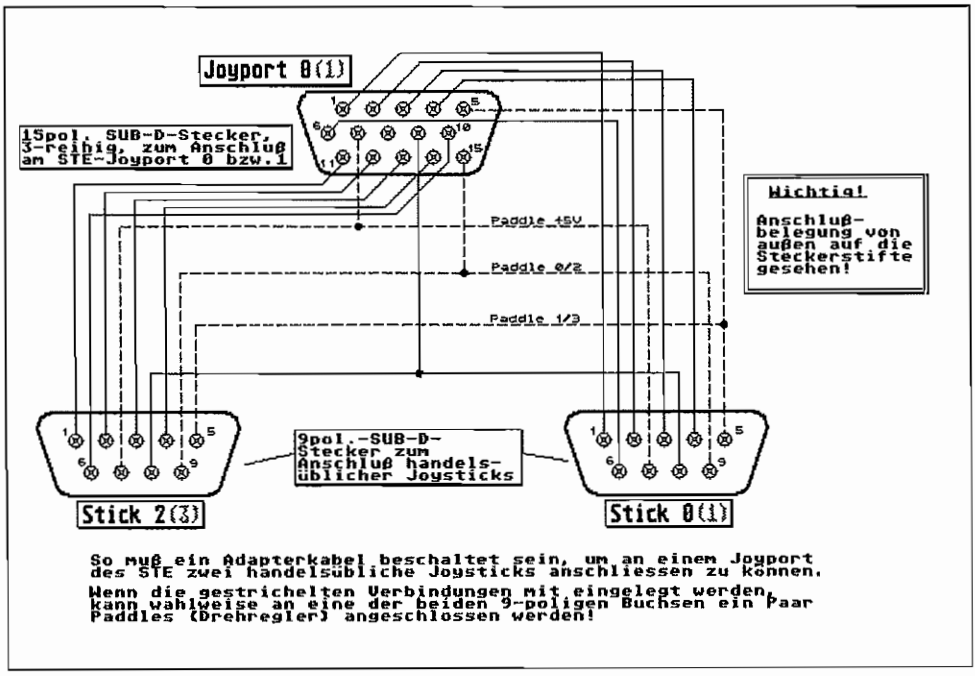


Abb. J.4: Adapter zum Anschluß von Joysticks und Paddles an die neuen STE-Joystickports

Wenn die gestrichelten Verbindungen ebenfalls eingelegt werden, so lassen sich an einem der beiden 9poligen Anschlüsse des Adapters zwei Paddles anschließen.

Gutes für die Ohren – Sound in neuer Qualität

Gegenüber den eher eingeschränkten Möglichkeiten der Sounderzeugung mit Hilfe des PSG im "normalen ST" hat sich beim STE doch einiges getan. So besitzt der STE jetzt zwei Soundkanäle mit je 8 Bit Auflösung, herausgeführt auf entsprechende Cinch-Buchsen zum Anschluß an handelsübliche Verstärker!

8-Bit-Auflösung heißt dabei nichts anderes, als daß sich die Amplitude (der Ausgangswert) eines beliebigen Tonsignals in maximal 256 Stufen unterteilen und damit nachbilden läßt. Jedes Byte mit seinen 8 Bit stellt dabei einen sogenannten Sample dar, also einen Wert, den das Ausgangssignal zu einem bestimmten Zeitpunkt annimmt. Dabei ist der 8-Bit-Wert als vorzeichenbehaftete Zahl zu betrachten.

$80	=	−128	= maximale negative Amplitude
$00	=	0	= minimale Amplitude (Ruhestellung)
$7F	=	+127	= maximale positive Amplitude

Zur Tonerzeugung werden diese Samples nacheinander zum Abspielen an einen Digital/Analog-Wandler (D/A-Wandler) ausgegeben, der diese in entsprechende Spannungswerte umwandelt. Nach entsprechender Aufbereitung (Filterung) steht dann das Signal an den Ausgangsanschlüssen zur Verfügung.

Messungen im mir zur Verfügung stehenden STE ergaben jedoch, daß sich nicht der komplette 8-Bit-Bereich auflösen läßt. Zwar soll der komplette Spannungsbereich zwischen –5V..+5V (Betriebsspannungen der D/A-Wandler) in 256 Stufen dargestellt werden können (je Stufe also ein Spannungshub von ca. 39mV). Die eingebauten D/A-Wandler hatten jedoch Probleme, Samples mit Werten > |80| in entsprechende Spannungswerte umzusetzen. Man sollte sich also, um Verzerrungen zu vermeiden, mit den Samplewerten im Bereich zwischen –80 und +80 bewegen.

Aus der Aneinanderreihung vieler dieser Samples läßt sich also ein beliebiges Tonsignal "zusammenbauen". Die Ausgabe dieser im Speicher als 8 Bit-Werte abgelegten Samples an die D/A-Wandler kann mit vier verschiedenen Geschwindigkeiten erfolgen. Im STE stehen die Samplefrequenzen 6,258 kHz, 12,517 kHz, 25,033 kHz und 50,066 kHz zur Verfügung.

Mit der niedrigsten Samplefrequenz werden also 6258 Samples pro Sekunde abgespielt! Oder anders ausgedrückt: Für eine Sekunde Sound sind 6258 Samples (= Bytes) erforderlich!

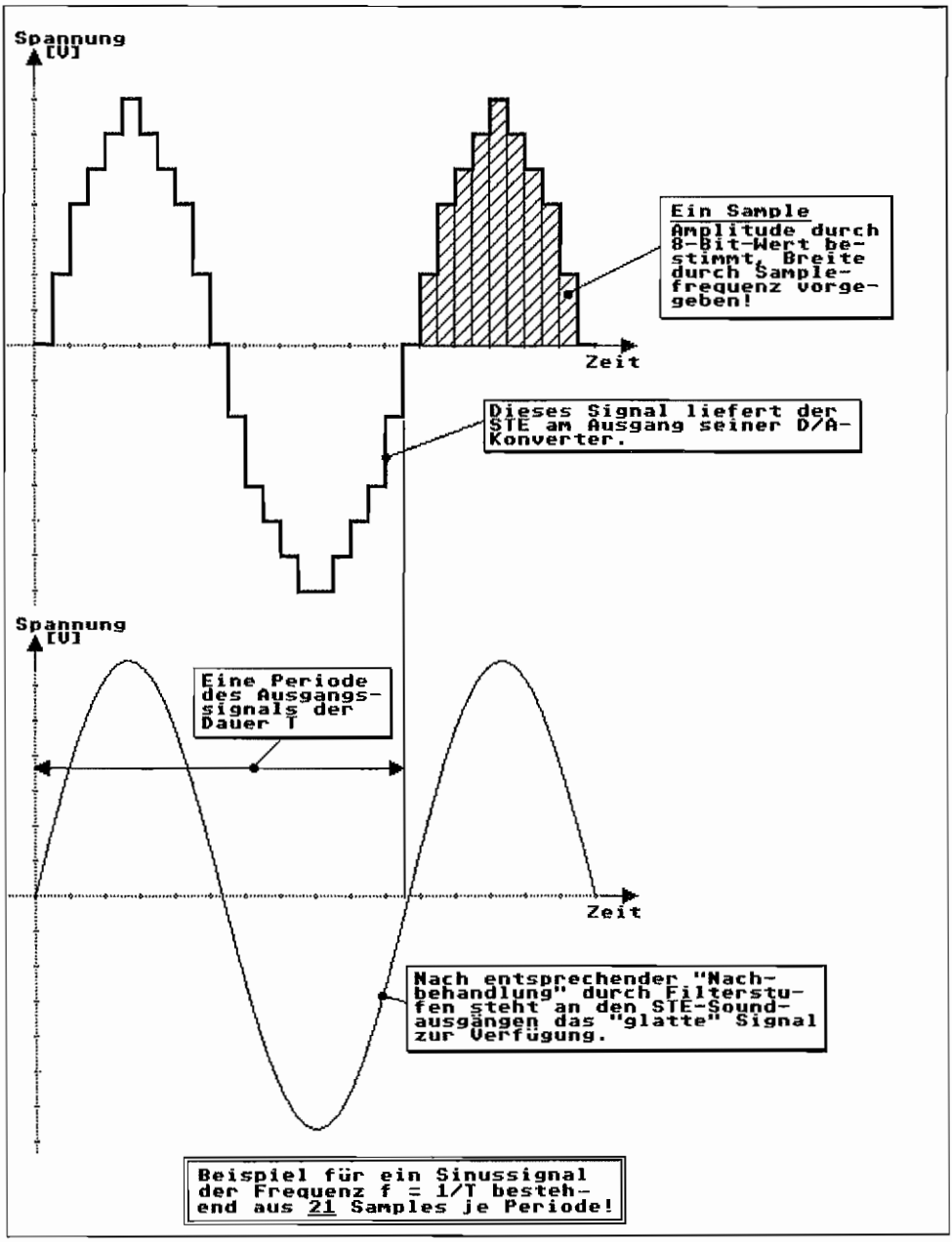


Abb. J.5: Aus Samples wird ein Tonsignal

Die Qualität des Sounds hängt wesentlich von der Samplefrequenz ab. So gilt die Faustregel, daß die Samplefrequenz mindestens doppelt so groß wie die höchste im Sound vorkommende Tonfrequenz sein soll. Um also auch hohe Signalfrequenzen im Sound realisieren zu können, sind entsprechend hohe Samplefrequenzen erforderlich. Für das menschliche Hörvermögen wird Musik in hoher Wiedergabequalität empfunden, wenn der Frequenzbereich zwischen ca. 20 Hz und 20 kHz möglichst unverfälscht wiedergegeben wird. Dazu sind also bei digitaler Signalverarbeitung Samplefrequenzen > 40kHz erforderlich. In der professionellen Technik arbeitet man deshalb mit Sampleraten von ca. 44 kHz, wobei jeder Sample als 16 Bit-Wert vorliegt (Signalwerte in max. 65536 Stufen darstellbar)!

Musik "von ganz alleine" – DMA auch für Sounddaten

Um nun die einzelnen Samplewerte nicht Byte für Byte durch die CPU an die D/A-Wandler ausgeben zu müssen, bedient sich der STE zum Abspielen ganzer Blöcke von Samples (auch als Frame bezeichnet) der DMA-Technik. Beim STE wird dazu im HBlank-Interrupt jeweils ein Word (also zwei Samples) durch das im STE-Shifter mitintegrierte Sound-DMA-Modul an die D/A-Wandler ausgegeben. Dabei kann in zwei Modi gearbeitet werden:

Stereo-Modus: Von dem vom Sound-DMA-Modul aus dem Speicher geholten Word wird das High-Byte als Samplewert für den linken Tonkanal (D/A-Wandler) verwendet und das Low-Byte für den rechten Kanal (D/A-Wandler).

Mono-Modus: Ein Word enthält zwei Samplewerte, die der Reihe nach (erst High-Byte, dann Low-Byte) an beide D/A-Wandler gleichzeitig ausgegeben werden. Somit steht an beiden Soundausgängen das gleiche Signal zur Verfügung.

Um also einen Frame (eine Gruppe von Samples) abzuspielen, muß das Sound-DMA-Modul "wissen", wo ein Frame beginnt (Anfangsadresse) und wo dieser endet (Endadresse). Des weiteren läßt sich bei der Programmierung des Sound-DMA-Moduls festlegen, mit welcher Geschwindigkeit die Samples ausgegeben werden sollen (Samplefrequenz). Samples lassen sich außerdem mehr als einmal abspielen. Das Sound-DMA-Modul läßt sich dazu auf Dauerbetrieb einstellen und spielt so den Frame immer wieder von neuem ab.

Da vom Sound-DMA-Modul immer Words aus dem Speicher gelesen und zur D/A-Wandlung gegeben werden, läßt sich immer nur eine gerade Anzahl von Samples pro Frame abspielen (auch bei Mono-Mode!). Entsprechend ist das Low-Bit der Frame-Start- und Frame-End-Adresse in den entsprechenden DMA-Registern immer Null! Die in der folgenden Tabelle mit "*" versehenen Voreinstellungen sind nach Einschalten des STE (Reset) gültig!

Adresse Label	R/W	Bits	Bedeutung
$FF 8900 sndmactl	R/W	----- ----- ----- --RE ||	Sound-DMA-Control-Reg.
		00	DMA-Sound aus (*)
		01	DMA-Sound ein
		11	DMA-Sound ein, Frame ständig wiederholen
$FF 8902 sndbashi	R/W	---- ---- 00HH HHHH	Frame-Start-Adresse (High-Byte)
$FF 8904 sndbasmi	R/W	---- ---- MMMM MMMM	Frame-Start-Adresse (Middle-Byte)
$FF 8906 sndbaslo	R/W	---- ---- LLLL LLLL	Frame-Start-Adresse (Low-Byte)
$FF 8908 sndadrhi	R	---- ---- 00HH HHHH	Frame-Adress-Counter (High-Byte)
$FF 890A sndadrmi	R R	---- ---- MMMM MMMM	Frame-Adress-Counter (Middle-Byte)
$FF 890C sndadrlo	R	---- ---- LLLL LLLL	Frame-Adress-Counter (Low-Byte)
$FF 890E sndendhi	R/W	---- ---- 00HH HHHH	Frame-End-Adresse (High-Byte)
$FF 8910 sndendmi	R/W	---- ---- MMMM MMMM	Frame-End-Adresse (Middle-Byte)
$FF 8912 sndendlo	R/W	---- ---- LLLL LLLL	Frame-End-Adresse (Low-Byte)
$FF 8920 sndmode	R/W	---- ---- M000 00RR ||	Sound-Mode-Control
		00	6258 Hz Samplerate (*)
		01	12517 Hz Samplerate
		10	25033 Hz Samplerate
		11	50066 Hz Samplerate
		0	Stereo-Modus (*)
		1	Mono-Modus

Wie ja schon vom DMA-Baustein des ST bekannt, kann man auch hier wieder nicht mittels Langwort-Zugriff direkt die Start- und End-Adressen setzen. Man muß auf Assemblerebene entweder in einem Datenregister die Adressen "zurechtschieben" und dann, schön der Reihe nach, Low-, Middle- und High-Byte der Frameadressen im entsprechenden Sound-DMA-Register setzen.

```
START:
      move.l    #fstart,d0      ; Frame-Startadresse -> D0.
      move.b    d0,sndbaslo     ; Low-Byte paßt, also einsetzen.
      ror.l     #8,d0           ; Jetzt Mid-Byte passend schieben.
      move.b    d0,sndbasmi     ; Middle-Byte wegbringen.
      ror.l     #8,d0           ; Jetzt High-Byte passend schieben.
      move.b    d0,sndbashi     ; Und ab ins Register damit.
```

Diese Version hat den Nachteil, daß der Adreßwert in einem Datenregister (z. B. D0) stehen muß und nach den Verschiebungen nicht mehr in seiner ursprünglichen Form vorliegt. Durch eine weitere Verschiebung um 8 Bits nach rechts ("ror.l #8,d0") ist der Adreßwert zwar wieder korrekt in D0 enthalten, aber das kostet natürlich auch alles CPU-Zeit!

Die nachfolgende Version benutzt den Stack für das "Aufbröseln" der Adressen in registergerechte Form.

```
START:
      pea       fstart,-(sp)    ; Frame-Start-Adr. auf Stack.
      tst.l     (sp)+           ; Stackptr. wieder korrigieren.
      move.b    -3(sp),sndbashi ; Nun die Bytes der Frame-Start-
      move.b    -2(sp),sndbasmi ; Adresse wie sie benötigt werden
      move.b    -1(sp),sndbaslo ; vom Stack holen und wegbringen!
```

Nachdem die Anfangs- und Endadresse auf die eine oder andere Weise dem Sound-DMA-Modul mitgeteilt worden sind, wird im Modus-Kontrollregister ("sndmode", Adr. $FF 8920) die gewünschte Samplefrequenz und die Betriebsweise (Stereo/Mono) gewählt.

Nach Setzen von Bit 0 im Sound-DMA-Control-Register ("sndmactl", Adr. $FF 8900) wird der Frame abgespielt. Ist Bit 1 im Sound-DMA-Control-Register ebenfalls gesetzt, wird der Frame ständig wiederholt. Eine Null im Sound-DMA-Control-Register stoppt den DMA-Sound sofort.

Wer wissen will, welcher Sample als nächster gespielt wird, kann jederzeit die drei Sound-Adress-Register auslesen. Sie bilden zusammen den Zeiger auf das nächste Sample-Word.

Soundsteuerung mit und ohne Timer-A-Interrupts

Um bei der DMA-Soundkontrolle flexibel zu sein, hat Atari beim STE ein wenig zusätzliche Hardware spendiert.

Die Register für Frame-Start-Adresse und Frame-End-Adresse sind gebuffert. Das bedeutet, daß eine dort hineingeschriebene Adresse zunächst zwischengespeichert wird. "Benutzt" wird diese Adresse erst dann (die Adreßwerte werden in die entsprechenden Arbeitsregister transportiert), wenn der momentan gespielte Frame beendet wurde. In der Praxis kann man deshalb schon während des Abspielens eines Frames die Adressen des nächsten Frames eintragen. Der nächste Frame wird dann "nahtlos" an den vorigen "angehängt"!

Mit der Information, wann ein Frame zu Ende ist, besteht die Möglichkeit, eine ganze Serie von Frames hintereinander zu ketten! Dazu liefert das DMA-Sound-Modul ein Signal über den augenblicklichen "Stand der Dinge". Von Pin 105 der MCU über einen Inverter (74LS04) geführt, steht ein Signal ("XSINT") zur Verfügung, das auf High liegt, wenn gerade ein Frame gespielt wird (siehe auch Abbildung J.6).

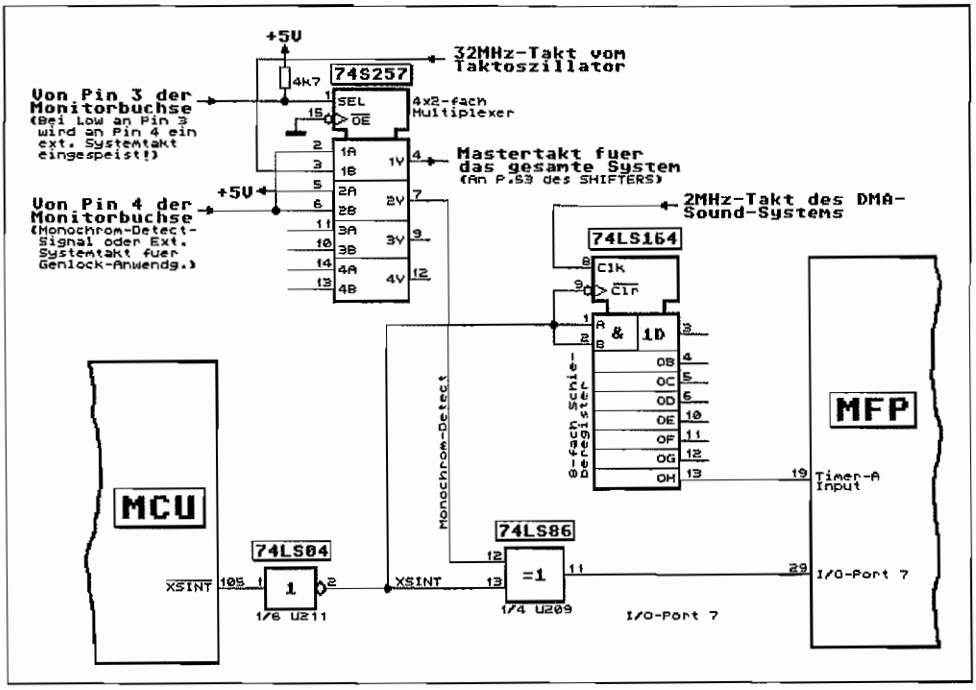


Abb. J.6: Signallauf des XSINT-Signals im STE

Mit Einlesen des letzten Words eines Frames geht das Signal auf Low und erst wieder auf High, wenn wieder ein Frame begonnen wird. Das geschieht auch, wenn das Sound-DMA-Modul auf "Dauerbetrieb" (ständiges Abspielen eines Frames) eingestellt ist. Da das Sound-DMA-Modul einen 4-Word-FIFO-Buffer besitzt, geht das Signal XSINT also bereits vier Words (= acht Samples) vor Frameende auf log. 0!

Beim STE wird das XSINT-Signal sinnvollerweise (nach einer Aufbereitung durch ein mit 2MHz getaktetes 8fach-Schieberegister des Typs 74LS164) auf den Eingang des Timers A vom MFP geführt und kann damit Interrupts auslösen, wenn jeweils ein oder mehrere Frames beendet wurden (je nach Einstellung des Timers A).

Um Impulse am Eingang des Timers A erfassen zu können, muß im Aktive-Edge-Register, Bit 4 (AER, Adresse $FF FA03) des MFP eingestellt werden, auf welche Flanke der XSINT-Impulse der Eingang des Timers A reagieren soll.

Mit diesem AER-Bit wird aber auch eingestellt, auf welche Flanke der Interrupt am Port IO4 des MFP (Interruptsignal von Midi- und Tastatur-ACIA) auslöst. Atari hat das XSINT-Signal deshalb so ausgelegt, daß bei beiden Signalen jeweils auf fallende Flanken reagiert wird (Bit 4 im AER auf 0!).

Man braucht sich deshalb um die Programmierung des AER-Bit 4 nicht mehr zu kümmern, denn das wird vom Betriebssystem zur Bedienung der Midi- und Tastatur-Interrupts schon richtig voreingestellt!

Um nun also verschiedene Frames direkt hintereinander abspielen und evtl. einzelne Frames unterschiedlich oft wiederholen zu können, wäre die folgende Vorgehensweise angebracht:

Beispiel: Frame A (eine Sinusschwingung) soll 100mal gespielt werden, direkt anschließend folgt 150mal eine Dreieckschwingung und zum Schluß noch 250mal der Frame C für eine Rechteckschwingung.

1. Start- und Endadressen für Frame A im Sound-DMA-Modul setzen.

2. Timer-A auf Ereigniszählung programmieren (es sollen die XSINT-Impulse (sprich: Frames) gezählt werden). Interruptvektor für Timer-A auf eigene Interruptroutine setzen. Timer-A mit der Zahl der Framewiederholungen –1 laden.

3. Samplerate und Modus (Stereo oder Mono) im Sound-Mode-Register einstellen. Bit 0 und 1 im Sound-DMA-Control-Register (sndmactl, Adresse $FF 8900) setzen. Frame 1 wird gespielt, Wiederholbetrieb ist eingeschaltet. [Am Ende der 99sten Wiederholung von Frame A löst der Timer-A einen Interrupt aus!]

4. In der Interruptroutine Adressen für Frame B setzen. Ebenso die Anzahl der Wiederholungen für Frame B (150) in Timer-A einstellen. Interruptroutine verlassen.

   [Sobald die letzte Wiederholung von Frame A beendet ist, wird der zweite Frame begonnen (Bufferung!), und der Timer-A zählt von 150 auf 149. Am Ende der 149sten Wiederholung von Frame B wird wieder ein Interrupt ausgelöst.]

5. In der Interruptroutine Adressen für Frame C setzen. Ebenso die Anzahl der Wiederholungen für Frame C (250) in Timer-A einstellen. Interruptroutine verlassen.

   [Am Ende der letzten Wiederholung von Frame B wird der dritte Frame begonnen, und der Timer-A zählt von 250 auf 249. Zum Ende der 249sten Wiederholung von Frame C wird wieder ein Interrupt ausgelöst.]

6. Im Interrupt Bit 1 des Sound-DMA-Control-Registers zurücksetzen. Damit ist die ständige Wiederholung eines Frames ausgeschaltet, und der letzte Frame wird noch zu Ende gespielt, dann hält das Sound-DMA-Modul an. Timer-A wird gestoppt, und die Timer-A-Interrupts werden disabled und maskiert. Das war's!

Nachfolgend ist ein Beispiellisting für den gerade beschriebenen Fall dargestellt.

```
;-------------------------------------------------------------------
; Definition der verwendeten Adressen und Konstanten

sndmactl    equ   $FF8900    ; Sound-DMA-Control-Register
sndbashi    equ   $FF8903    ; Frame-Start-Address (High-Byte)
sndbasmi    equ   $FF8905    ; Frame-Start-Address (Middle-Byte)
sndbaslo    equ   $FF8907    ; Frame-Start-Address (Low-Byte)
sndendhi    equ   $FF890F    ; Frame-End-Address (High-Byte)
sndendmi    equ   $FF8911    ; Frame-End-Address (Middle-Byte)
sndendlo    equ   $FF8913    ; Frame-End-Address (Low-Byte)
sndmode     equ   $FF8920    ; Sound-DMA-Mode-Register

MWDATA      equ   $FF8922    ; MICROWIRE-Data-Register
MWMASK      equ   $FF8924    ; MICROWIRE-Mask-Register

iera        equ   $FFFA07    ; Interrupt-Enable-Register A
isra        equ   $FFFA0F    ; Interrupt-in-Service-Register A
imra        equ   $FFFA13    ; Interrupt-Mask-Register A

tadr        equ   $FFFA1F    ; Timer-A-Data-Register
tacr        equ   $FFFA19    ; Timer-A-Control-Register
```

```
;-------------------------------------------------------------------
; Beispielprogramm für die Programmierung des STE-DMA-Sounds unter
; Ausnutzung der Frame-Ende-Signalisierung (Signal XSINT) durch das
; Sound-DMA-Modul mit auf Ereigniszählung programmiertem Timer-A.
; (Das Programm muß im Supervisor-Modus ausgeführt werden!)

start:
     lea       FRMDAT,a2  ;Ab FRMDAT steht eine Tabelle mit Zeigern
     move.l    (a2)+,a0   ; auf Frame-Start/End-Adressen und Werten
     move.l    (a2)+,a1   ; für die Wiederholzahl der einzelnen
                          ; Frames.
     bsr       setframe   ; Frameadressen setzen.

     move.w    (a2)+,d0   ; Timer-A-Wert nach Register D0 holen.
     subi.w    #1,d0      ; Passend machen.
     move.l    a2,pointer ; Zeiger für nächstes Mal verwahren.

;  Timer-A wird jetzt auf Ereigniszählung programmiert. Außerdem
;  wird der zugehörige Timer-A-Interruptvektor auf die Routine
;   "t_a_int" umgeleitet.
;  Timer-A wird mit der Anzahl der Wiederholungen-1 für den ersten
;  Frame geladen. Nach entsprechend häufigem Wiederholen des Frames
;  A wird der Timer-A einen Interrupt auslösen, und die t_a_int-
;  Routine tritt in Aktion.

     pea       t_a_int    ; Zeiger auf Timer-A-Interruptroutine.
     move.w    d0,-(sp)   ; Wert für Timer-A-Data-Register.
     move.w    #$8,-(sp)  ; Wert f. T-A-Ctrl-Register
                            (Ereigniszähl.!).
     clr.w     -(sp)      ; Timer 0 (= Timer-A) ist gemeint.
     move.w    #31,-(sp)  ; Xbios-Aufruf Nr. 31 (= Xbtimer)
     trap      #14
     lea       12(sp),sp  ; Stack korrigieren.

;  MFP-Interrupt #13 (Timer-A-Interrupt) wird freigegeben.

     move.w    #13,-(sp)  ; Interrupt #13 des MFP einschalten.
     move.w    #27,-(sp)  ; Xbios-Aufruf Nr. 27 (= Jenabint)
     trap      #14
     addq.l    #4,sp      ; Stack korrigieren.
```

```
;  Sound-DMA-Modul einstellen

     move.w     #$80,sndmode     ; Mono-Modus mit 6258Hz Samplerate
                                 ; einstellen.
     move.w     #3,sndmactl      ; Bis auf ewig abspielen!

;  Über MICROWIRE-Interface den Volume/Tone-Control-Chip vor-
;  einstellen.

     move.w     #$7ff,MWMASK          ; MICROWIRE-Mask-Register
                                      ; einstellen.
     move.w     #%10011101000,d0      ; Lautstärke des LMC1992 auf
                                      ; Maximum!
     bsr        mw_write   ; Befehl ü. MICROWIRE-Interf. ausgeb.
the_end:
     rts

;-------------------------------------------------------------------
; Interruptroutine, die vom Sound-DMA-Modul am Ende eines Frames
; jeweils aufgerufen wird. Sie sorgt dafür, daß das Sound-DMA-Modul
; immer mit den nächsten Framedaten nachgeladen wird. Außerdem wird
; der Timer-A jeweils mit der Anzahl der Wiederholungen für den
; nächsten Frame versehen. Wenn der letzte Eintrag in der Tabelle
; ab FRMDAT abgearbeitet ist und der Interrupt zu Beginn der letz-
; ten Wiederholung des letzten Frames auftritt, wird das Sound-DMA-
; Modul auf Einzelframe-Wiedergabe geschaltet (letzten Frame noch
; zu Ende spielen und dann stoppen) und Timer-A "abgeschaltet"!

t_a_int:
     movem.l    d0/a0-a2,-(sp)  ; Benutzte Register auf Stack
                                ; retten.
     move.l     pointer,a2      ; Zeiger auf Frame-Daten holen.
     tst.l      (a2)            ; Tabelleneintrag testen.
     beq.s      t_a_int1   ; Letzte Wiederholung des letzten Frames?
     move.l     (a2)+,a0   ; Nein! Aus Tabelle die nächsten
     move.l     (a2)+,a1   ; Frame-Adressen holen.
     bsr        setframe   ; Und im DMA-Modul setzen.
     move.w     (a2)+,d0   ; Dann neuen Wert für Timer-A holen.
     move.b     d0,tadr    ; Und Timer-A entsprechend setzen.
     move.l     a2,pointer ; Neuen Zeiger für nächstes Mal merken.
     bra.s      t_a_end    ; Zum Ende springen.
```

```
t_a_int1:
     move.b     #0,tacr          ; Ja! Timer-A stoppen!
     and.b      #$DF,iera        ; Timer-A-Int. des MFP disablen.
     and.b      #$DF,imra        ;      "       des MFP maskieren.
     move.w     #1,sndmactl      ; Frame noch beenden und dann
                                 ; Schluß!
t_a_end:
     move.b     #$DF,isra        ; MFP über Int.-Ende informieren!
     movem.l    (sp)+,d0/a0-a2   ; Registerinhalte restaurieren.
     rte

;-------------------------------------------------------------------
; Unterprogramm zum Setzen der Start- und Endadresse des abzuspie-
; lenden Frames.
;    Parameter:       A0 = Startadresse des Frames
;                     A1 = Endadresse des Frames
;    Benutzte Reg.:   Keine
;
setframe:
     move.l     a0,-(sp)         ; Startadresse des Frames in
     tst.l      (sp)+            ; die Frame-Start-Address-Register
     move.b     -3(sp),sndbashi  ; schreiben.
     move.b     -2(sp),sndbasmi
     move.b     -1(sp),sndbaslo
     move.l     a1,-(sp)         ; Endadresse des Frames in
     tst.l      (sp)+            ; die Frame-End-Address-Register
     move.b     -3(sp),sndendhi  ; schreiben.
     move.b     -2(sp),sndendmi
     move.b     -1(sp),sndendlo
sfrmend:
     rts

;-------------------------------------------------------------------
; Unterprogramm zum Schreiben über MICROWIRE-Interface
;    Parameter:       D0 = zu schreibende Daten
;    Benutzte Reg:    Keine
;
mw_write:
     cmp.w      #$7ff,MWMASK     ; Warten, bis letzter
     bne.s      mw_write         ; Schreibvorgang beendet ist.
```

```
      move.w     d0,MWDATA        ; Nun aber raus mit den Daten!
      rts

;-------------------------------------------------------------------------
      .data
pointer:
      dc.l FRMDAT      ; Zeiger auf die nächsten Framedaten.
FRMDAT:
      dc.l FRAMEA,FRAMEB    ; Adressen für ersten Frame + Timer-A-
      dc.w 100              ; Wert

      dc.l FRAMEB,FRAMEC    ; Adressen für zweiten Frame + Timer-A-
      dc.w 150              ; Wert

      dc.l FRAMEC,FRMENDE   ; Adressen für dritten Frame + Timer-A-
      dc.w 250              ; Wert

      dc.l 0                ; Null signalisiert das Tabellenende!

FRAMEA:                     ; Daten für eine Sinusschwingung
      dc.b 0,14,27,40,51,61,69,75,79,80
      dc.b 79,75,69,61,51,40,27,14,0
      dc.b -14,-27,-40,-51,-61,-69,-75,-79,-80
      dc.b -79,-75,-69,-61,-51,-40,-27,-14
FRAMEB:                     ; Daten für eine Dreieckschwingung
      dc.b 0,10,20,30,40,50,70,80
      dc.b 70,60,50,40,30,20,10,0
      dc.b -10,-20,-30,-40,-50,-60,-70,-80
      dc.b -70,-60,-50,-40,-30,-20,-10
FRAMEC:                     ; Daten für eine Rechteckschwingung
      dc.b 80,80,80,80,80,80,80,80
      dc.b -80,-80,-80,-80,-80,-80,-80,-80
FRMENDE:
```

Um nun nicht unbedingt den letzten frei verfügbaren Timer für die DMA-Soundsteuerung benutzen zu müssen, hat Atari im STE das XSINT-Signal noch an anderer Stelle für Programmierer verfügbar gemacht.

So wird das Frame-Ende-Signal noch zusätzlich mit dem Monochrom-Detect-Signal der Monitorbuchse über ein Exklusiv-Oder (EXOR) zusammengefaßt und dann auf den Port IO7 des MFP geführt (siehe hierzu auch Abbildung J.6!).

Um dieses Signal jedoch benutzen zu können, muß man ein paar Gegebenheiten berücksichtigen.

1. Der Interrupt tritt an jedem Frame-Ende (eigentlich vier Sample-Words vor Frameende, wegen des FIFO-Buffers im Sound-DMA-Modul) auf, und nicht nach einer bestimmten Anzahl von Framewiederholungen.

2. Die mittels eines Interrupts auswertbare Flanke, zu der das DMA-Modul von "Abspielen" auf "Frameende" umschaltet, ist durch die EXOR-Verknüpfung mit dem Monochrom-detect-Signal abhängig vom verwendeten Monitor! Hat man einen Monochrom-Monitor angeschlossen, so ist der 1->0-Flankenwechsel für die Interruptauslösung zu programmieren. Ist kein Monochrom-Monitor angeschlossen, so ist die 0->1-Flanke zu benutzen. Um nun zu ermitteln, welche Flanke für die Interruptauslösung im AER, Bit 7, denn nun zu verwenden ist, ist folgender Weg zu beschreiten:

   - Das Sound-DMA-Modul abschalten (0 ins Sound-DMA-Control-Register schreiben), dadurch ist das XSINT-Signal auf 0! Der Zustand des Ports I7 des MFP wird jetzt nur noch wegen der EXOR-Verknüpfung durch das Monochrom-Detect-Signal beeinflußt!

   - Port I7 des MFP auslesen. Wenn eine log. 1 gelesen wird, hat man es mit einem Color-Monitor zu tun, und das Bit 7 im AER des MFP ist für die Interruptauslösung auf 1 zu setzen. Wird an Port I7 eine log. 0 erkannt, ist ein Monochrom-Monitor angeschlossen, und Bit 7 im AER ist zur Interruptauslösung mittels des XSINT-Signals auf 0 zu setzen.

Man kann also die Sound-DMA-Steuerung auch über diesen MFP-Port I7-Interrupt abwikkeln. Es steht halt nur kein Timer für das Zählen der Frames zur Verfügung; das muß man dann in der Interruptroutine schon selbst machen!

Von Digital nach Analog – Der Weg des Tonsignals im STE

Die Abbildung J.7 zeigt das Zusammenspiel des STE-DMA-Soundteils im Überblick.

Es ist deutlich die Trennung in Linker und Rechter Kanal zu erkennen. Die vom SHIFTER im DMA-Betrieb aus dem RAM gelesenen Sound-Daten (Samples) gelangen als 8-Bit-Werte zunächst in ein 8-Bit D-Flip-Flop und werden dort "eingefroren", bis der nächste Sample folgt! Das "Einfrieren" wird durch die Signale _LD (Daten für linken Kanal) und _RD (Daten für rechten Kanal) für jeden Kanal getrennt bewirkt. Bei Mono-Betrieb werden beide 8-Bit-Latches gleichzeitig (_LD und _RD sind synchron) bedient.

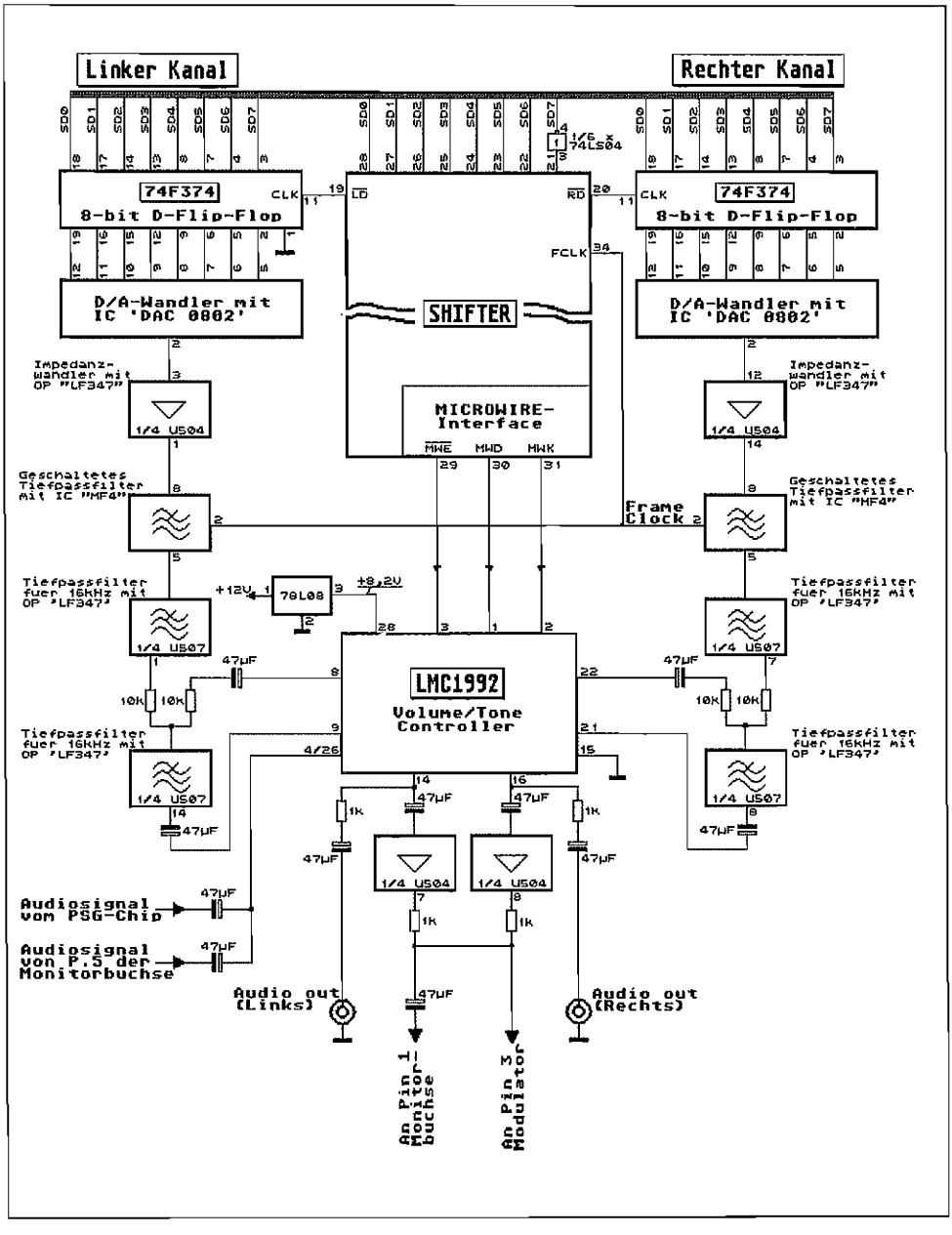


Abb. J.7: Das Zusammenspiel des STE-DMA-Soundteils

An den Ausgängen der 8-Bit-Latches steht also jeweils für die Dauer eines Samples eine 8-Bit-Information an. Diese wird durch die nachgeschalteten Digital/Analog-Wandler in einen entsprechenden Spannungswert umgesetzt. Der Ausgangswert wird hochohmig über einen Impedanzwandler (1fach Verstärker mit hohem Eingangs- und niedrigem Ausgangswiderstand) abgegriffen und einem geschalteten Tiefpaßfilter zugeführt. Dieser geschaltete Tiefpaßfilter vierter Ordnung wird direkt durch die Samplefrequenz (Frameclock) gesteuert und läßt nur noch Frequenzen unterhalb 40% der Samplefrequenz durch!

Anschließend gelangt das Signal über zwei 16kHz-Tiefpässe zweiter Ordnung. Das bedeutet also, daß in einem DMA-Sound-Ausgangssignal nur Frequenzen bis zu max. 16kHz "vertreten" sein können. Alles was darüber liegt, wird ausgefiltert. Am Ausgang des zweiten 16kHz-Filters liegt das Signal dann zur Bearbeitung durch den Volume/Tone-Controller LMC1992 bereit. Die Pegelbeeinflussung (Volume) und Balancesteuerung erfolgt erst im LMC1992. Für die Anhebung/Absenkung der Bässe bzw. Höhen kann der LMC1992 einen Teil des Signals (in den Bässen/Höhen angehoben oder abgesenkt) zwischen dem ersten und zweiten 16kHz-Tiefpaßfilter einkoppeln.

An den Ausgängen des LMC1992 wird das Tonsignal für den rechten und linken Kanal direkt abgegriffen und über einen Schutzwiderstand (gegen Kurzschlüsse am Ausgang) und einen Kondensator gleichspannungsfrei auf die Cinch-Buchsen geführt.

Damit über den Lautsprecher eines angeschlossenen Monitors auch etwas zu hören ist, werden beide Kanäle über eine Mischstufe (bestehend aus zwei Verstärkerstufen und zwei Mischwiderständen) zusammengefaßt. Von dort gelangt das Tonsignal (in Mono) an die Monitorbuchse und an den HF-Modulator.

Um aber das Tonsignal vom "alten" Soundchip PSG nicht zu vergessen (woher sonst den "schönen" Tastaturklick nehmen), hat Atari dessen Audiosignal ebenfalls über den LMC1992 geführt. Zusammen mit einem von außen zuführbaren externen Tonsignal wird das PSG-Audiosignal ebenfalls einer Bearbeitung durch den LMC1992 unterworfen.

Kontrolle über den Sound – Das MICROWIRE(TM)-Interface

Für die Beeinflussung des vom Sound-DMA-Modul gelieferten Tonsignals hat Atari einen eigenen Chip von National Semiconductor eingesetzt. Mit diesem speziellen Chip ist es möglich, per Softwarekontrolle Einstellung der Lautstärke (Volume), Balance (zwischen den beiden Kanälen) vorzunehmen. Außerdem ist eine Bass-/Höhenregelung per Software realisierbar.

Da dieser Chip nicht speziell für den STE ausgelegt wurde, sondern auch in Geräten der Unterhaltungselektronik (z. B. Fernseher) Verwendung findet, wird für die Übergabe der Parameter wie z. B. Lautstärke, Balance usw. ein einfaches, serielles Bussystem verwendet.

National Semiconductors spricht hierbei vom MICROWIRE(TM)-Bussystem. Sowohl der Volume/Tone-Controller Chip LMC1992 als auch der STE besitzen dafür ein eigenes MICROWIRE(TM)-Interface. Im STE wurde das Interface mit im SHIFTER-Chip untergebracht!

Noch eine serielle Schnittstelle im STE

Wie bereits angesprochen, wird der Parameteraustausch (Lautstärke, Balance usw.) mit dem LMC1992 über einen seriellen Bus abgewickelt. Der Bus besteht nur aus drei Leitungen, wobei die MWD-Leitung (MICROWIRE(TM)-Data) zum Transport der eigentlichen Daten benutzt wird.

Dabei ist der Datentransport nur in eine Richtung, nämlich zum LMC1992 hin, vorgesehen. Das _MWE-Signal (MICROWIRE(TM)-Enable) geht so lange auf Low, wie Daten gesendet werden, und über die MWK-Leitung (MICROWIRE(TM)-Clock) erfährt der Empfänger (in unserem Fall der LMC1992), in welchem Takt die Daten übermittelt werden. Die Datenübertragungsgeschwindigkeit beträgt dabei im STE immerhin 1 MBit/s!

Das Bussystem und Übertragungsprotokoll ist dabei sehr flexibel ausgelegt, so daß mehrere verschiedene MICROWIRE(TM)-Devices angesprochen werden können.

Jedes Device kann individuell vom Controller (STE) adressiert werden. Die Länge des zu übermittelnden Datenstroms ist dabei abhängig vom adressierten Device. Damit jedes Device "mitbekommt", wann es gemeint ist und welche Daten übertragen werden, hat der serielle Bitstrom folgendes Format:

Zunächst werden N-Adrßsbits (beim LMC1992 im STE sind das zwei Bits) zur Festlegung des gewünschten Devices gesendet. Darauf folgen eventuell einige "Don't care"-Bits und dann die eigentlichen Datenbits.

Für die Bedienung des seriellen Interfaces werden im STE-SHIFTER zwei 16-Bit-Schieberegister benutzt. Dazu werden im MICROWIRE(TM)-Data-Register die Bits für den Bitstrom (Adreß- und Datenbits) abgelegt (Bit 15 wird zuerst ausgegeben, Bit 0 zum Schluß), der über die MWD-Leitung ausgegeben werden soll. Das MICROWIRE(TM)-Mask-Register bestimmt, welche Bits im MW-Data-Register gültige Bits sind (also Adreß- bzw. Datenbits) und welche als "Don't care"-Bits fungieren.

Adresse Label	R/W	Bits	Bedeutung
$FF 8922 MWDATA	R/W	DDDD DDDD DDDD DDDD	Adreß-/Datenbits zur Ausgabe ü. MWD-Leitung
$FF 8924 MWMASK	R/W	MMMM MMMM MMMM MMMM	MICROWIRE(TM)-Mask-Register

Um also an z. B. den LMC1992 des STE eine Information abzusetzen, sind im MWDATA-Register zunächst die Adreßbits korrekt zu setzen. Der LMC1992 im STE hat die Adresse #2 entsprechend binär "10". Danach folgen die zu sendenden Datenbits. Wollte man z. B. an den LMC1992 den Binärwert "11011" senden, so könnte das mit folgenden MWMASK- und MWDATA-Register-Einstellungen geschehen:

<——— Schieberichtung

```
1 0 1 1   0 1 1 X   X X X X   X X X X   <- MWDATA
1 1 1 1   1 1 1 0   0 0 0 0   0 0 0 0   <- MWMASK
```

oder

```
X X X X   1 0 X X   X X 1 1   0 1 1 X   <- MWDATA
0 0 0 0   1 1 0 0   0 0 1 1   1 1 1 0   <- MWMASK
```

oder

```
X X X X   X X X X   X 1 0 1   1 0 1 1   <- MWDATA
0 0 0 0   0 0 0 0   0 1 1 1   1 1 1 1   <- MWMASK
```

oder

```
1 0 X X   X X X X   X X X 1   1 0 1 1   <- MWDATA
1 1 0 0   0 0 0 0   0 0 0 1   1 1 1 1   <- MWMASK
```

(X = Don't care)

Wie in den Beispielen zu sehen ist, kommt es also nur darauf an, daß Adreß- und Datenbits gleichzeitig im MWDATA-Register stehen und die zugehörigen Mask-Bits gesetzt sind. Adreßinformation und Daten dürfen durch mehrere Bitpositionen getrennt werden, können aber auch direkt hintereinander anschließen.

Bei der Bedienung der beiden Register und damit des MICROWIRE(TM)-Interfaces im STE ist zu beachten, daß zuerst das MWMASK-Register gesetzt wird. Denn sobald das MWDATA-Register beschrieben wurde, gehen die Informationen auf den Bus!

Während der Registerinhalt auf den Bus ausgegeben wird, ist ein Beschreiben der MWMASK und MWDATA-Register nicht möglich. Um also die nächste Ausgabe starten zu können, muß gewartet werden, bis die gerade laufende Ausgabe beendet ist.

Die beiden Schieberegister werden zur Informationsausgabe dabei Bit für Bit nach links *rotiert!* Das bedeutet, daß nach insgesamt 16 Bit-Shifts (also bei 1MBits/s Datenübertragungsgeschwindigkeit nach ca. 16µsec) die Register durch eine komplette Rotation des eingeschriebenen Bitmusters wieder in Ausgangsposition angekommen sind! Das Auslesen der Register während der laufenden Ausgabe ist möglich und liefert jeweils eine "Momentaufnahme" über den Zustand der Schieberegister.

Diese Eigenschaften kann man sich zunutze machen, indem man durch ständiges Auslesen des MWMASK-Registers nach Starten der Ausgabe den Zeitpunkt feststellt, wann das MWMASK-Register wieder im Ausgangszustand angelangt ist. Dann ist nämlich die letzte Ausgabe abgeschlossen, und der nächste Wert für das MWDATA-Register kann eingebracht werden.

Da man es im STE selten mit einem anderen Device als dem LMC1992 zu tun haben wird, kann man das MWMASK-Register nach dem erstmaligen Setzen der "Valid"-Bits unverändert lassen. Alle weiteren Kommandos an den LMC1992 haben ja das gleiche Format bezüglich der Anordnung der "Valid"-Bits! Man braucht also nur das MWDATA-Register mit neuen Werten zu versehen.

Computer Controlled Volume/Tone – Der LMC 1992

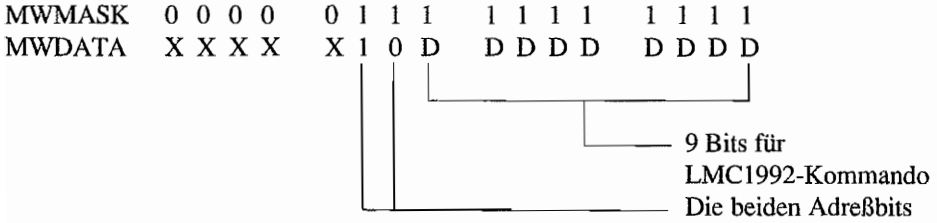
Beim LMC1992 handelt es sich um einen relativ leistungsfähigen Baustein zur Kontrolle von Audiosignalen. Im STE werden die im folgenden dargestellten Einstellmöglichkeiten benutzt:

LMC1992 Device Adresse: $10_{binär}$

Datenbits								**Befehl**
011	V	V	V	V	V	V		Lautstärke einstellen
	0	0	0	0	0	0		–80 dB (Einstellung in Schritten
	0	1	1	0	0	0		–40 dB (zu je 2 dB möglich!)
	1	0	1	X	X	X		0 dB = max. Lautstärke

Datenbits											Befehl
101	X	L	L	L	L	L					Lautstärke linker Kanal (2 dB/Schritt)
		0	0	0	0	0					–40 dB
		0	1	0	1	0					–20 dB
		1	0	1	X	X					0 dB
100	X	R	R	R	R	R					Lautstärke rechter Kanal (2 dB/Schritt)
		0	0	0	0	0					–40 dB
		0	0	1	0	1					–30 dB
		1	0	1	X	X					0 dB
010	X	X	H	H	H	H					Höhenanhebung bei ca. 15kHz um 2 dB/Schritt
			0	0	0	0					–12 dB Absenkung
			0	1	1	0					0 db (Linear)
			1	1	0	0					+12 dB Anhebung
001	X	X	B	B	B	B					Baßanhebung bei ca. 50Hz um 2 dB/Schritt
			0	0	0	0					–12 dB Absenkung
			0	1	1	0					0 dB (Linear)
			1	1	0	0					+12 dB Anhebung
000	X	X	X	X	M	M					Mischen mit PSG-Soundsignal/Ext. Soundsignal
					0	0					PSG-Signal mit –12 dB Absenkung zumischen
					0	1					PSG-Signal 1:1 zumischen
					1	0					PSG-Signal nicht zumischen
					1	1					reserviert

Alle Kommandos sind beim LMC1992 also neun Bits breit. Mit den jeweils zwei Bits ($10_{binär}$) für die LMC1992-Adresse ergeben sich also immer insgesamt elf Bits, die über den Bus auszugeben sind. Setzt man die Adresse und Daten unmittelbar aneinander und rechtsbündig ins MWDATA-Register, so ist als Wert für das MWMASK-Register das Bitmuster $07FF zu wählen. MWMASK- und MWDATA-Register müßten dann also folgendermaßen aussehen:



Um im STE den Volume/Tone-Control-Chip auf max. Lautstärke einzustellen, müßte also folgende Einstellung in den beiden Schieberegistern benutzt werden:

MWMASK	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1
MWDATA	X	X	X	X	X	1	0	0	1	1	1	0	1	0	0	0

Die Bedienung der MWMASK- und MWDATA-Register in einem Programm ist ebenfalls aus dem Beispiellisting ersichtlich.

Mehr Kontrollmöglichkeiten für den Videoteil

Atari hat dem STE noch einiges an zusätzlicher Hardware im Videoteil "spendiert". Damit lassen sich vor allen Dingen in der Colorbetriebsart einige interessante Effekte realisieren. Hauptsächliche Anwendungen dieser zusätzlichen Effektmöglichkeiten dürften im Spielebereich zu finden sein.

Aber auch für einen anderen Hobbybereich dürfte der STE nun interessanter werden. Bisher war es nur mit direkten Manipulationen in der ST-Hardware möglich, sogenannte GENLOCK-Anwendungen unter Einbeziehung von STs zu realisieren. Gedacht ist hierbei an das Mischen von ST-Videosignalen mit anderen Videosignalquellen (z. B. Videokamera), um so beispielsweise Titelbild- oder Hintergrundgestaltung mittels Computer zu unterstützen.

Um Videosignale unterschiedlicher Quellen zu mischen, ist es erforderlich, daß alle Signale zueinander synchron sind. Das bedeutet nichts anderes, als daß ein Videosignal als Mastersignal verwendet wird, auf welches sich alle anderen Videosignale einzustellen haben. Für den ST/STE folgt daraus, daß nicht die intern erzeugten H- und V-Synchronisationssignale verwendet werden, sondern extern zugeführte Signale zu verwenden sind. Das war ja schon bei den STs möglich (Bit 0 im Sync-Mode-Register (Adr. $FF 820A) auf log. 1 setzen und an der Monitorbuchse HSync- und VSync-Signale einspeisen)! Aber damit war der Beginn des Bildinhalts in jeder Zeile immer noch vom *internen* ST-Takt abhängig und führte deshalb zu unerwünschtem "Zeilenreißen" (das Bild begann von Zeile zu Zeile mehr oder weniger stark waagerecht versetzt und erschien deshalb ausgefranst und unruhig).

Im STE ist es jetzt möglich, auch einen externen Mastertakt als Systemtakt einzuspeisen, der von einer Mastersignalquelle abgeleitet wird. Diese Mastersignalquelle muß in der Lage sein, an alle zu mischenden Videosignalquellen die aus einem Mastertakt abgeleiteten HSync- und VSync-Signale zu liefern und dem STE einen Mastertakt von 32,084988 MHz (= PAL-Anwendungen; bei NTSC-Anwendungen soll der ext. Takt bei 32,215905 MHz liegen) zur Verfügung zu stellen. Damit läuft der STE also völlig synchron zur Mastersignalquelle, und das Zeilenreißen ist vorbei.

Wie kommt der Mastertakt nun in den STE hinein? Dafür hat Atari die Beschaltung der STE-Monitorbuchse gegenüber den STs etwas verändert. An Pin 3 der Monitorbuchse kann nicht länger das General-Purpose-Output-Signal (GPO-Signal) vom PSG-Port IOA 6 abgegriffen werden. Vielmehr wird jetzt durch das Anlegen einer log. 0 an diesem Anschluß der "Weg frei gemacht" für den ext. 32 MHz-Mastertakt. Dieser Mastertakt wird dann an Pin 4 der Monitorbuchse eingespeist. Die Zusammenhänge zwischen diesen beiden Anschlüssen zeigt auch der Schaltungsauszug in Abbildung J.6. Bei High an Pin 3 arbeitet Pin 4 als Monochrom-Detect-Eingang, und bei Low wird der Mastertakt an Pin 4 erwartet.

Wichtig! Nicht während des laufenden Betriebs die Taktumschaltung mit Pin 3 vornehmen. Also immer bei ausgeschaltetem STE Pin 3 gegen Masse legen und ext. Mastertakt an Pin 4 zuführen.

Mehr Farben im Zugriff – Erweiterungen bei der Farbpalette

Eine weitere Hardware-Änderung betrifft den SHIFTER. Genauer gesagt die Palette-Register, denen Atari im STE nun für jede der drei Primärfarben Rot, Grün und Blau 4 Bit (statt bisher 3 Bit im ST) für die Verschlüsselung der Farbintensität zugedacht hat. Somit sind nun 3 x 4 Bit zur Farbauswahl möglich, und man kann aus $2^{4+4+4} = 2^{12} = 4096$ Farben wählen. Dabei sind aber ohne Tricks gleichzeitig auch künftig nur 16 verschiedene Farben darstellbar (wegen der weiterhin nur 16 Farbpalette-Register)!

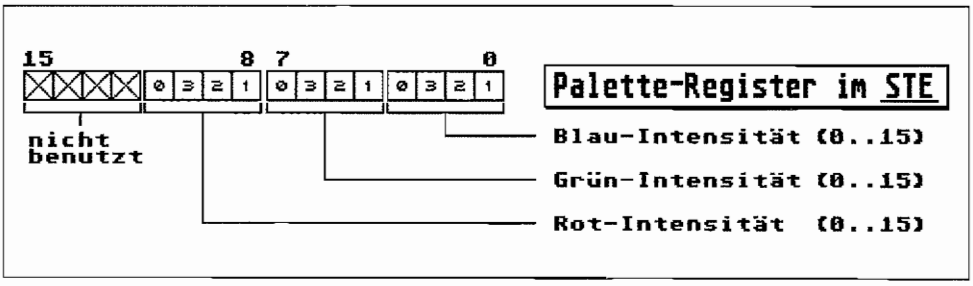


Abb. J.8: Palette-Register im STE

Um mit dem STE trotz der erweiterten Farb-Palette möglichst kompatibel zu bleiben, ist die Gewichtung (Intensität), mit der die einzelnen Primärfarben im Gesamtsignal auftauchen, für die unteren 3 Bits jeder Primärfarbe beibehalten worden. Die Voreinstellung für ein giftiges Grün ist auf ST und STE mit dem gleichen Wert im entsprechenden Palette-Register möglich. Beim STE kann man dieses Grün durch das zusätzliche 4. Bit halt noch ein wenig "giftiger" machen.

Entsprechend den zusätzlichen Bits in den Palette-Registern, wurde natürlich der D/A-Wandler am SHIFTER-Ausgang um je ein Bit auf 4 Bit-Auflösung je Primärfarbe erweitert.

Finescrolling mit Hardware-Unterstützung

Durch einige Hardware-Änderungen ist es mit dem STE jetzt einfacher möglich, Finescrolling (Bildverschiebungen im Pixel-Raster) sowohl in senkrechter als auch in waagerechter Richtung vorzunehmen. Dazu wurden zwei neue Register eingeführt und drei bereits bekannte Register des Videoteils in ihrer Funktion erweitert.

Senkrechtes Finescrolling

läßt sich einfach dadurch realisieren, daß man während der Dunkeltastung des Elektronenstrahls (im Vertical-Blank) die Anfangsadresse des Bildschirmspeichers um eine Bildschirmzeile (das sind in der Regel 80 Bytes) herauf- oder heruntersetzt. Dadurch ergibt sich der Eindruck, als wäre das gesamte Bild etwas nach unten bzw. nach oben verschoben worden. Natürlich muß man im Speicher schon etwas mehr als nur einen Bildschirminhalt zur Verfügung haben (z. B. mehrere komplette "Bildschirme" direkt hintereinander in den Speicher laden). "Oben" und "unten" vom gerade dargestellten Bildschirminhalt muß sich im Speicher ja ebenfalls sinnvolle Bildinformation befinden, die angezeigt wird, wenn nach oben oder unten gescrollt wird.

Man kann sich das Ganze also so vorstellen, als würde man auf dem Monitor ein Fenster sehen, das in senkrechter Richtung über ein großes Bild (im Speicher) verschoben werden kann.

Das Problem mit dem senkrechten Finescrolling im ST besteht darin, daß man den Anfang des Bildschirmspeichers mittels des Video-Base-Registers (Video-Base-Reg. High-Byte bei Adr. $FF 8201, Video-Base-Reg. Mid-Byte bei Adr. $FF 8203) nur in 256 Byte-Schritten versetzen kann. Es "fehlt" ja im Video-Base-Register ein Registerplatz für das Low-Byte der Anfangsadresse des Bildschirmspeichers.

Im STE hat man dieses Register zusätzlich "eingebaut", so daß senkrechtes Finescrolling jetzt problemlos möglich ist. Der vollständige Registersatz für die Anfangsadresse der Bildschirmadresse sieht damit im STE folgendermaßen aus:

Adresse Label	R/W	Bits	Bedeutung
$FF 8201 dbaseh	R/W	XXHH HHHH	Video-Base-Register, High-Byte

Adresse Label	R/W	Bits	Bedeutung
$FF 8203 dbasel	R/W	MMMM MMMM	Video-Base-Register, Medium-Byte
$FF 820C dbaselow	R/W	L L L L L L L 0	Video-Base-Register, Low-Byte

Diese drei Register bilden zusammen den Pointer auf den Anfang des Bildschirmspeichers. Durch das zusätzliche "dbaselow"-Register läßt sich die Anfangsadresse des Bildschirmspeichers im STE auf jede beliebige Word-Grenze im RAM legen.

Diese Anfangsadresse kann jederzeit ausgelesen werden. Auch das Neusetzen der Bildschirmspeicher-Anfangsadresse kann jederzeit durchgeführt werden; Auswirkungen sind aber erst nach dem nächsten Vertical-Blank (also beim Neubeginn des nächsten Bildes) zu erwarten. Denn was im Moment dargestellt wird, ist von der Adresse im Video-Address-Counter abhängig, und dieser wird jedesmal zu Beginn eines neuen Bildes mit dem Wert im Video-Base-Register initialisiert!

Nachfolgend ist ein einfaches Demoprogramm für senkrechtes Finescrolling im Monochrom-Modus aufgeführt. Es werden zwei "normale" Doodle-Bilder hintereinander in einen Buffer geladen. Die aktuelle Bildschirmspeicher-Basisadresse wird abgefragt und verwahrt. Danach wird die Basisadresse des Bildschirmspeichers auf den Anfang des Buffers mit den beiden Bildern gelegt. Anschließend wird schrittweise der sichtbare Bildschirmausschnitt durch ständiges Erhöhen der Bildschirmspeicher-Basisadresse um 80 Bytes (= eine Zeile) darüber "hinweggefahren". Das ergibt den Effekt, als würde das Bild von unten nach oben durchrollen.

Zum guten Schluß wird der Videocontroller wieder auf die alte Bildschirmbasisadresse zurückgesetzt.

```
1000 ' Vertikales Finscrolling mit STE (Einfaches Demo im
     ' Monochrom-Modus!)
1010 '
```

```
1020 ' Es werden zwei Bilddateien im Doodle-Format eingeladen und
1030 ' zu einem Großbild "hintereinander" kopiert. Dieses Großbild
1040 ' wird dann per vertikalem Scrolling einmal von oben nach
     ' unten "durchfahren"!
1050 '

1060 '
```

```
1070 MEMORY_BLOCK 00,64000,Buffer%L:'          Platz für zwei
                                               ".DOO"-Bilder
1080 BLOAD "BILD_1.DOO",Buffer%L:'             Bilder einladen.
1090 BLOAD "BILD_2.DOO",Buffer%L+32000
1100 '
1110 XBIOS (Physbase%L,2):'                    Alte Bildschirm-
                                               adresse merken.
1120 '
1130 XBIOS (,5,L -1,L Buffer%L,-1):'           Auf "Großbild" um-
                                               schalten!
1140 '
1150 FOR Ypos%L=0 TO 399:'                     Vertikale Pos.
                                               schrittweise ändern.
1160    Scrnadr%L=Buffer%L+Ypos%L*80:'         Neue Screenadresse
                                               berechnen
1170    XBIOS (,5,L -1,L Scrnadr%L,-1):'       und einstellen.
1180    WAIT .01:'                             Kleine Pause!
1190 NEXT Ypos%L
1200 '
1210 XBIOS (,5,L -1,L Physbase%L,-1):'         Auf alten Screen
                                               zurückschalten.
1220 FRE (Buffer%L):'                          Benutzten Speicher
                                               wieder freigeben!
```

Um unmittelbare Reaktionen zu erzielen (nach einer bestimmten Zahl von Bildschirmzeilen auf einen anderen Bildschirmspeicherbereich umzuschalten o. ä.), müßte man den Video-Address-Counter manipulieren können. Dieser enthält nämlich den Pointer auf das nächste Word im Bildschirmspeicher, das geholt und in Bildinformation umgesetzt werden soll. *Beim ST* ist der Video-Address-Counter ein 3-Byte-*Read-only*-Register.

Im STE hat Atari dieses Register ebenfalls für Schreibzugriffe zugänglich gemacht.

Adresse Label	**R/W**	**Bits**	**Bedeutung**
$FF 8205 vcounthi	*R/W*	XXHH HHHH	Video-Address-Counter, High-Byte
$FF 8207 vcountmid	*R/W*	MMMM MMMM	Video-Address-Counter, Med.-Byte
$FF 8209 vcountlow	*R/W*	L L L L L L L 0	Video-Address-Counter, Low-Byte

Beim Umgang mit diesem Register ist Vorsicht geboten, da das Resultat unmittelbar sichtbar wird. Eine Bedienung des Video-Address-Counters erscheint deshalb nur während des Horizontal-Blank sinnvoll. Dann ist der Elektronenstrahl im Monitor gerade vom Ende des Bildinhalts der just geschriebenen Bildschirmzeile zum Anfang des Bildinhalts in der neuen Zeile unterwegs und deshalb dunkel getastet. Für das Zählen der abgelaufenen Bildschirmzeilen kann man ja wieder den Timer B als Ereigniszähler verwenden (siehe auch im ST-Hardwareteil, Kapitel 4, "Der Multifunktionsbaustein MFP 68901").

Horizontales Finescrolling

ist nicht gar so einfach zu realisieren. Denn hierbei muß ein Bild (genauer gesagt jede Bildschirmzeile) an einem beliebigen Pixel beginnen können. Und das bedeutet nichts anderes, als daß man dem Videocontroller mitteilen muß, ab welcher Speicheradresse *und ab welchem Bit* in dieser Adresse denn die Bildschirmzeile beginnt.

Im STE wurde dafür eigens das HScroll-Register geschaffen.

Adresse Label	R/W	Bits	Bedeutung
$FF 8265 hscroll	*R/W*	XXXX SSSS	HScroll-Register, verzögert den Bildbeginn um $SSSS_{binär}$ Bits.

Mit diesem Register sind Werte zwischen 0 und 15 einstellbar. Das bedeutet, daß zunächst $SSSS_{binär}$-Pixel bei der Ausgabe als Bildinformation übersprungen werden. Bei HScroll = 0 hat man also keine horizontale Verschiebung auf dem Bildschirm und deshalb die gleichen Verhältnisse wie im ST.

Im Monochrom-Betrieb ist dieses Verhalten am einfachsten darzustellen, da ja dann jedem Bit ein Pixel auf dem Bildschirm entspricht. Eine Zeile mit 640 Pixel besteht dann aus 80 aufeinanderfolgenden Bytes im Bildschirmspeicher.

Weil der Videocontroller (genauer gesagt der SHIFTER) im ST(E) nicht einzelne Bytes aus dem Bildschirmspeicher holt und als Bildinformation umsetzt, sondern das immer wordweise geschieht, ist also eine Bildschirmzeile 40 Words lang.

Ist HScroll auf z. B. 5 eingestellt, so werden aus dem Word 0 des Bildschirmspeichers die höchstwertigen fünf Bits übersprungen und nicht als Bildinformation ausgegeben!

Die Bildschirmzeile beginnt fünf Bits "später" mit Bit 10 des Word 0 des Bildschirmspeichers und ist 640 Bit (= 640 Pixel bei Monochrom) danach zu Ende.

Das letzte Pixel der ersten Zeile würde also dem Bit 11 des Word 40 vom Bildschirmspeicher entsprechen. Die zweite Bildschirmzeile fängt dann frühestens wieder (wegen HScroll=5) mit Word 41, Bit 10 an und geht bis zu Word 81, Bit 11 usw. (siehe dazu auch Abbildung J.9).

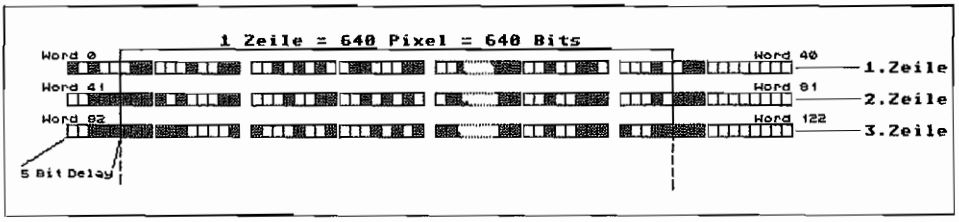


Abb. J.9: Organisation des Bildschirmspeichers im Monochrombetrieb mit HScroll=5

Im Gegensatz zu der Organisation des Bildschirmspeichers mit HScroll=0 ist eine Zeile bei HScroll>0 also mindestens ein Word länger. Denn die Zeile beginnt ja nicht mehr an einer Word-Grenze, sondern um HScroll-Bits nach rechts verschoben. Da die Anzahl Bits/Zeile aber weiterhin gleichbleibt, "ragt" das Ende der Zeile um HScroll-Bits in das 41.Word hinein!

Um diesem Umstand Rechnung zu tragen, hat man im STE ein weiteres Hardware-Register eingeführt. Das Line-Width-Register ("linewid" an Adr. $FF 820F) gibt an, um wie viele Words die Zeile denn "länger" als eine normale Bildschirmzeile (normalerweise 40 Words/ hohe Auflösung, 80 Words/mittlere und niedrige Auflösung) ist.

Da man hier ein 8-Bit-Register zur Verfügung hat, kann man hier Werte bis zu 255 verwenden. Das bedeutet aber auch nichts anderes, als daß man eine Bildschirmzeile im Speicher 255 Words länger als die "normale" Zeile machen kann!

Adresse **Label**	**R/W**	**Bits**	**Bedeutung**
$FF 820F linewid	R/W	OOOO OOOO	Line-width-Register Versatz in Words bis zum Beginn der nächsten Bildschirmzeile

Man kann also auch in horizontaler Richtung "große" Bildschirme aufbauen und dann durch entsprechende Einstellung des HScroll-Registers und des Bildschirmbasispointers (dbasehi, dbasel und dbaselow) jeweils den gewünschten Ausschnitt davon auf dem Bildschirm darstellen. Das nachfolgende Beispiellisting demonstriert das.

Dabei werden zwei "normale" Monochrom-Bilder (640 x 400=256 000 Pixel=16 000 Words) "nebeneinander" in einen Buffer kopiert. Also erst 40 Words (1. Zeile) von Bild 1, dann 40 Words (1. Zeile) von Bild 2, wieder 40 Words (2. Zeile) von Bild 1, dann 40 Words (2. Zeile) von Bild 2 usw.

```
1000 ' Horizontales Scrolling mit STE (Einfaches Demo im Monochrom-
     ' Modus!)
1010 '
```

```
1020 ' Es werden zwei Bilddateien im Doodle-Format eingeladen und
1030 ' "zu einem nebeneinanderliegenden" Großbild zusammenkopiert.
1040 ' Dieses Großbild wird dann per horizontalem Scrolling einmal
1050 ' "von links nach rechts abgefahren"!
1060 '
1070 Lineoffs%L=$FF820F:'              Benutzte Hardware-Reg.
                                       für Scrolling
1080 Hscroll%L=$FF8265
1090 '
1100 MEMORY_BLOCK 00,64000,A%L:'       Platz für zwei ".DOO"-
                                       Bilddateien
1110 MEMORY_BLOCK 01,64000,Buffer%L:'  Platz für das "Großbild"
1120 BLOAD "BILD_1.DOO",A%L:'          Zwei Bilder einladen
1130 BLOAD "BILD_2.DOO",A%L+32000
1140 '
1150 XBIOS (Physbase%L,2):'            Alte Bildschirm-
                                       speicheradresse merken.
1160 '
1170 ' Beide Einzelbilder "nebeneinander" im Buffer ablegen.
1180 FOR X%L=0 TO 399
1190    MEMORY_MOVE A%L+X%L*80,80 TO Buffer%L+X%L*160
1200    MEMORY_MOVE A%L+32000+X%L*80,80 TO Buffer%L+X%L*160+80
1210 NEXT X%L
1220 '
1230 XBIOS (,5,L -1,L Buffer%L,-1):'   Auf "Großbild"
                                       umschalten!
1240 POKE Lineoffs%L,39:'              Lineoffs = 40-1 Words beim
                                       Scrollen
1250 '
1260 FOR Xpos%L=0 TO 639:'               Horizontale Position
                                         schrittweise ändern.
1270    IF Xpos%L MOD 16=0 THEN :'       Word-Grenze erreicht?
```

```
1280       Scrnadr%L=Buffer%L+Xpos%L SHR 3:'    Ja! Neue Screen-
                                                adresse berech-
1290       XBIOS (,5,L -1,L Scrnadr%L,-1):'     nen und einstellen.
1300       XBIOS (,37):'                        Bis zum VBlank
                                                warten.
1310       POKE Hscroll%L,1:'                Hscroll = 0 überspringen.
1320    ELSE
1330       POKE Hscroll%L,Xpos%L AND $F:'       Nein! Pixel-Delay
                                                setzen.
1340    ENDIF
1350    WAIT 7E-3:'                          Kleine Pause!
1360 NEXT Xpos%L
1370 '
1380 XBIOS (,5,L -1,L Physbase%L,-1):'       Auf alten Screen
                                             zurückschalten
1390 POKE Hscroll%L,0:'                      Alles wieder in
                                             Ausgangszustand
1400 POKE Lineoffs%L,0
1410 FRE (A%L): FRE (Buffer%L):'             Benutzten Speicher wieder
                                             freigeben!
```

Um zum Schluß des Programms wieder auf die "normale" Bildschirmorganisation zurückgreifen zu können, wird der aktuelle Bildschirmbasispointer gerettet. Dann wird durch Neuladen des Bildschirmbasisspeichers mit der Anfangsadresse des "Großbildbuffers" auf den neuen Bildschirm geschaltet. Damit der Videocontroller des STE jetzt weiß, daß der Bildschirmspeicher "breiter" angelegt ist, wird das "linewid"-Register entsprechend eingestellt.

In einer Schleife wird jetzt die Startposition des dargestellten Bildschirms ständig um ein Pixel nach rechts gerückt. Dabei wird das "HScroll"-Register entsprechend mit hochgezählt und ein pixelweises Scrollen nach rechts erreicht. Wenn mehr als 16 Pixel (= 16 Bit = 1 Word in Monochrom) gescrollt wurde, muß die Anfangsadresse des Bildschirmspeichers um ein ganzes Word weitergeschaltet (hochgezählt) werden. Danach kommen wieder 16 Steps mit dem "HScroll"-Register, gefolgt durch das Inkrementieren des Bildschirmbasispointers usw.

Das Ändern der Videocontroller-Register sollte natürlich dann erfolgen, wenn es möglichst wenig auf dem Bildschirm stört, also am besten im VBlank. Die im Listing verwendete Xbios(,37)-Funktion (= Vsync) leistet dabei schon gute Hilfe. Zum Ende des Programms wird auf Standardeinstellung zurückgeschaltet und der alte Bildschirmbasispointer restauriert.

Mit dieser Hardware-Unterstützung lassen sich also relativ einfach große Bitplanes im Speicher (virtuelle Playfields) bearbeiten.

Im Monochrom- und Low-Resolution-Modus kommt es beim STE-Shifter bei H-Scrolling-Anwendungen zu kleinen Störungen im Bildaufbau, immer wenn das HSCROLL-Register von einem Nicht-Null-Wert auf Null geht! Atari empfiehlt deshalb, bei H-Scrolling-Anwendungen im Pixel-Raster die erforderlichen Display-Manipulationen bei solch einem bevorstehenden Wechsel von HSCROLL <> 0 auf HSCROLL = 0 nicht im VBlank, sondern kurz vorher durchzuführen!

Atari schlägt folgende Vorgehensweise vor, um die für das horizontale Scrolling verwendeten Hardware-Register (Video-Base-Address, Hscroll und linewid) zum richtigen Zeitpunkt zu setzen. In einer VBlank-Interrupt-Routine werden jeweils die neuen Registerwerte ermittelt, aber erst *später* durch eine Interrupt-Routine gesetzt. Dieser Interrupt wird durch den als Bildschirmzeilenzähler verwendeten Timer B des MFP ausgelöst! Wenn Hscroll auf Null wechselt, werden die Hardware-Register während der letzten Display-Zeile geändert. Bei Hscroll=0 wird nach der letzten Bildschirmzeile manipuliert!

Speicheraufrüstung leicht gemacht – Atari geht mit der Zeit

Im STE findet man endlich das RAM in zeitgemäßer Bauform, sprich als Bausteine, wie sie heute im Zeitalter fortgeschrittener Fertigungsmöglichkeiten immer verstärkter angeboten werden. Die Zeiten, in denen man für 4 MByte RAM auf der Platine noch den Platz eines halben DIN-A4-Blattes brauchte, sind nun langsam vorbei. Atari verwendet im STE zur Realisierung des RAMs sogenannte SIMMs. Dabei handelt es sich um kleine Platinenstreifen mit Direktsteckkontakten, auf denen je acht RAM-Chips in SMD-Technik aufgebracht sind. Diese "RAM-Streifen" gibt es sowohl zu 256K x 8 Bit (z. B. "58256-10") als auch 1M x 8 Bit (z. B. "581000-10") organisiert. Beide Typen können verwendet, sprich einfach in die vorhandenen vier Steckfassungen gesetzt werden. Damit lassen sich wieder die bereits vom ST bekannten Kombinationsmöglichkeiten für den RAM-Ausbau herstellen. Also RAM von 1MByte über 2,5MByte bis zu 4MByte (in meiner Testmaschine) ist damit möglich.

"Nimm SIMMS – dann stimmt's"

Die Einbindung der vier möglichen Fassungen mit "RAM-Streifen" ist im nachfolgenden Schaltungsauszug dargestellt. Je Bank sind auch hier wieder die Aufteilung in High-Byte und Low-Byte des Datenbusses zu sehen. Da auch hier Leitungen gespart werden sollen, wird der Adreßbus, wie in den STs, gemultiplext.

Mit den neun Leitungen des Multiplex-Adreßbusses lassen sich folglich nacheinander 2 x 9 = 18 Adreßleitungen übertragen und somit 2^{18} = 1.048.576 Speicherzellen ansprechen.

Da die RAM-Streifen byteweise organisiert sind, kann man max. 1MByte je Streifen ansprechen.

Verwendet werden auch hier dynamische RAMs, deren Inhalt nach einer bestimmten Zeit wieder "aufgefrischt" werden muß. Außerdem muß man entsprechende Steuerinformationen für die RAMs liefern, damit diese wissen, welche Hälfte der Adreßinformation gerade auf dem Multiplex-Adreßbus daherkommt. Dazu werden auch hier wieder die RAS- und CAS-Signale benötigt.

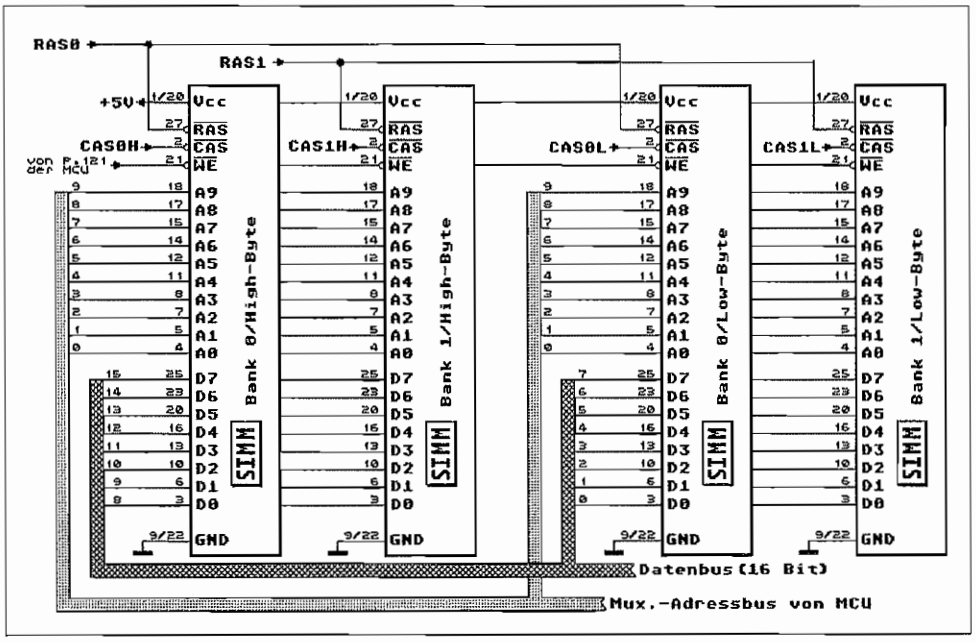


Abb. J.10: Das RAM im STE

Anhang K:
Die Echtzeituhr des MEGA-ST(E)

Die Computer der MEGA-ST(E)-Serie besitzen alle einen batteriegepufferten Uhrenchip des Typs "RP5C15". Auch die Tastaturcomputer des Typs STE könnten diesen Uhrenchip ansteuern, die Steuersignale sind an der GSTMCU vorhanden!

Ab TOS Version 1.02 (Blitter-TOS) wird dieser Uhrenchip auch benutzt, wenn er denn vorhanden ist! Das TOS übergibt die Zeit- und Datumsinformation an das GEMDOS, wenn das System initialisiert wird und jedesmal wenn ein Anwenderprogramm beendet wird (mit dem GEMDOS-Aufruf "Pterm0").

Die Einbindung des Uhrenchips in die Hardware des MEGA-ST zeigt der in Abbildung K.1 dargestellte Schaltungsauszug. Der Uhrenchip bekommt seinen Takt natürlich nicht vom Systemtakt, denn der "verschwindet" ja beim Ausschalten des Computers. Deshalb also noch ein zusätzlicher, eigener Quarz nur für den Uhrenchip. Wie der Schaltungsauszug zeigt, besitzt der Uhrenchip noch zwei weitere Anschlüsse, die von ATARI nicht benutzt werden.

Zum einen wäre da der Open-Collector-Ausgang "_ALARM". Der Uhrenchip kann nämlich nicht nur Zeit und Datum weiterzählen, sondern wie ein normaler Wecker zu einem bestimmten Datum und einer eingestellten Zeit ein Signal auslösen. Pin 15 des Uhrenchips schaltet immer dann nach Masse durch, wenn Alarmzeit und -tag mit den aktuellen Daten übereinstimmt. Das Durchschalten hält eine Minute an, weil danach ja Alarmzeit und aktuelle Zeit nicht mehr auf die Minute übereinstimmen. Damit dürften Bastler doch bestimmt etwas anfangen können (Interrupts auslösen o.ä.).

Der zweite zu betrachtende Ausgang, der von ATARI freundlicherweise direkt auf einen Testpunkt herausgeführt wurde, ist der Pin 3 (CLKOUT) des Uhrenchips. Hier kann, je nach Programmierung eines der Register des Uhrenchips, ein Takt abgegriffen werden, der direkt aus dem Systemtakt des Uhrenchips abgeleitet wird. Welche Taktfrequenzen an diesem Open-Collector-Ausgang entnommen werden können, wird noch bei der Besprechung der Chip-Register im einzelnen aufgeführt. Von einem Impuls pro Minute bis zu 16384 Impulsen pro Sekunde sind hier möglich!

Der Datenbus zum Uhrenchip ist auf ganze vier Leitungen zusammengeschrumpft. Mehr sind aber auch nicht erforderlich, da der Uhrenchip für jede Stelle der Uhrzeit- und Datumsinformation ein eigenes 4-Bit-Register besitzt. Die Selektion der einzelnen Register erfolgt über vier Adreßleitungen, obwohl der Uhrenchip über 32 Register (!) verfügt. Man bedient sich

dabei eines "Switch-Bits" in einem Register, mit dem man von einer Registerbank mit 16 Registern auf eine zweite Registerbank umschalten kann. Das Register mit dem "Switch-Bit" ist natürlich in beiden Bänken vorhanden; wie sollte man sonst wieder zurückschalten können?

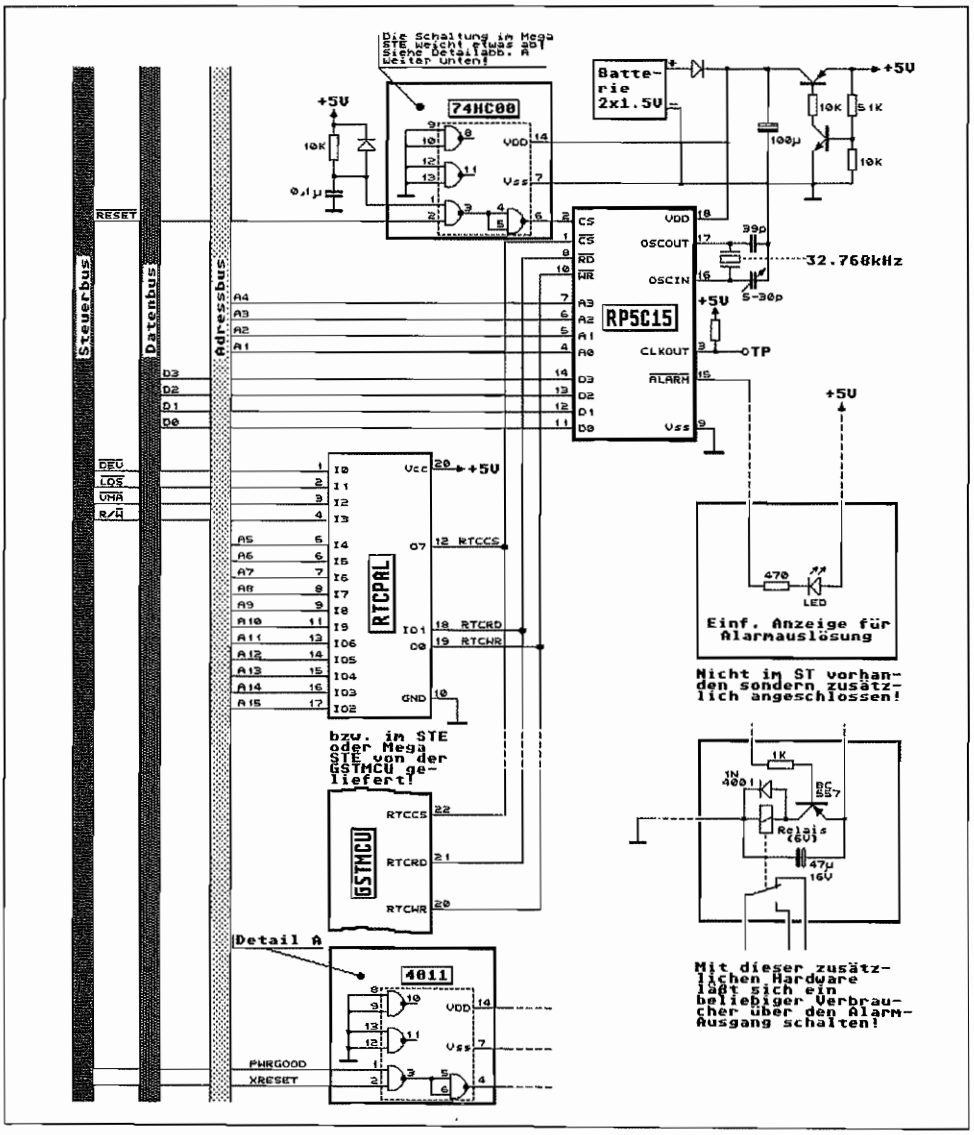


Abb. K.1: Die Echtzeituhr im MEGA-ST

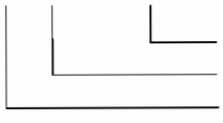
Die Register des Uhrenchips

Die Register des Uhrenchips liegen im Adreßraum des ST auf ungeraden Adressen, sind jedoch auch als niederwertigstes Nibble mit Word-Zugriffen auf die jeweils darunterliegende, gerade Adresse zu erreichen.

Nachfolgend die Tabelle mit den einzelnen Registern, deren Adresse und Bedeutung:

Zunächst die Register für die Bank 0

Adresse	Bitbelegung D3	D2	D1	D0	Bedeutung
$FF FC21	s	s	s	s	Sekunden-Einer (0..9)
$FF FC23	-	s	s	s	Sekunden-Zehner (0..5)
$FF FC25	m	m	m	m	Minuten-Einer (0..9)
$FF FC27	-	m	m	m	Minuten-Zehner (0..5)
$FF FC29	h	h	h	h	Stunden-Einer (0..9)
$FF FC2B	-	-	h	h	Stunden-Zehner (0..1 in 24h-Modus, sonst mit Bit D1 AM/PM-Anzeige im 12h-Modus (gesetztes Bit D1=1 entspricht PM-Zeit)).
$FF FC2D	-	w	w	w	Wochentag (0 = Sonntag). Vom TOS nicht benutzt.
$FF FC2F	D	D	D	D	Tage-Einer (0..9)
$FF FC31	-	-	D	D	Tage-Zehner (0..3)
$FF FC33	M	M	M	M	Monate-Einer (0..9)
$FF FC35	-	-	-	M	Monate-Zehner (0..1)
$FF FC37	Y	Y	Y	Y	Jahre-Einer (0..9)
$FF FC39	Y	Y	Y	Y	Jahre-Zehner (0..9) (da das TOS immer in Jahren ab 1980 rechnet, wird für das Jahr 1988 in "Jahre-Einer" also eine 8 und für "Jahre-Zehner" eine 0 eingetragen)
FF FC3B	ST	AL	-	BK	Modus-Register (Bitbedeutung siehe unten)



"Switch-Bit" (0= Bank 0, 1= Bank 1 angewählt)
Alarm-Bit (0= Alarm aus, 1= Alarm ein)
Stop-Bit (0= Uhr Stop, 1= Uhr läuft)

Adresse	Bitbelegung D3	D2	D1	D0	Bedeutung
$FF FC3D	0	0	0	0	Test-Register (alle Bits müssen 0 sein!)
$FF FC3F	T1	T2	RS	AR	Reset-Register (Bitbelegung siehe unten)
					Alarm-Reset (1= setzt alle Alarm-Reg. auf 0)
					Uhr-Reset (1= setzt alle int. Uhr-Reg. zurück)
					16Hz ein (0= 16Hz-Impulse an Pin 15 ausgeben)
					1Hz ein (0= 1Hz-Impulse an Pin 15 ausgeben)

Und nun die Register der Bank 1:

Adresse	Bitbelegung D3	D2	D1	D0	Bedeutung
$FF FC21	-	C2	C1	C0	Diese drei Bits steuern die Impulsfolge, welche an Pin 3 ausgegeben wird
	-	0	0	0	Open-Collector-Ausgang "CLKOUT" gesperrt
	-	0	0	1	Ausgangsfrequenz 16384 Hz
	-	0	1	0	Ausgangsfrequenz 1024 Hz
	-	0	1	1	Ausgangsfrequenz 128 Hz
	-	1	0	0	Ausgangsfrequenz 16 Hz
	-	1	0	1	Ausgangsfrequenz 1 Hz (Sekundentakt)
	-	1	1	0	Ausgangsfrequenz 1/60 Hz (Minutentakt)
	-	1	1	1	Open-Collector-Ausgang "CLKOUT" durchgeschaltet
$FF FC23	-	-	-	AD	Adjust-Bit. Bei Setzen von Bit "AD" gehen die Sekundenregister auf 0! Wenn die Sekunden zwischen 30..59 waren, werden gleichzeitig die Minuten um 1 hochgezählt.
$FF FC25	am	am	am	am	Alarmzeit Minuten-Einer (0..9)
$FF FC27	-	am	am	am	Alarmzeit Minuten-Zehner (0..5)
$FF FC29	ah	ah	ah	ah	Alarmzeit Stunden-Einer (0..9)
$FF FC2B	-	-	ah	ah	Alarmzeit Stunden-Zehner (0..2)
$FF FC2D	-	aw	aw	aw	Alarmzeit Wochentag (0=Sonntag)
$FF FC2F	aD	aD	aD	aD	Alarmzeit Tage-Einer (0..9)

Adresse	Bitbelegung D3	D2	D1	D0	Bedeutung
$FF FC31	-	-	aD	aD	Alarmzeit Tage-Zehner (0..3)
$FF FC33	-	-	-	-	Nicht benutzt!
$FF FC35	-	-	-	24	Ist das "24"-Bit gesetzt, so arbeitet der Uhrenchip im 24h-Modus.
$FF FC37	-	-	SJ	SJ	Schaltjahr-Register (es wird mit den Jahren von 0..3 hochgezählt, 0 = Schaltjahr!)
$FF FC39	-	-	-	-	Nicht benutzt!
$FF FC3B	ST	AL	-	BK	Modus-Register (Wie in Bank 0)
$FF FC3D	0	0	0	0	Test-Register (Wie in Bank 0)
$FF FC3F	T1	T2	RS	AR	Reset-Register (Wie in Bank 0)

Wie man sieht, ist dieser kleine Chip doch ganz schön leistungsfähig. Leider wird vom Betriebssystem außer der Abfrage des laufenden Datums und der Uhrzeit kaum von den Möglichkeiten Gebrauch gemacht.

Die Alarmfunktion des Chips kann man leider nicht nutzen, da die gegenwärtige TOS-Version nichts Besseres zu tun hat, als das Register "Alarmzeit Minuten-Einer" zum Test zu benutzen, ob denn der Uhrenchip überhaupt anwesend ist. Dazu wird bei der Systeminitialisierung ein unsinniger Wert in dieses Register geschrieben und anschließend verglichen, ob dieser Wert auch wirklich dort steht (= Chip vorhanden). Leider wird vor dem Test nicht der augenblickliche Inhalt des Registers gerettet und nach dem Test wieder zurückgeschrieben!

Zu allem Überfluß werden beim Aufruf der XBIOS-Funktion #22 ("Settime") und auch der GEMDOS-Funktionen #43 ("Tsetdate") und #45 ("Tsettime") zwar die Zeit- und Datumsregister im Uhrenchip korrekt gesetzt, aber im Modus-Register wieder das Alarm-Bit gelöscht, und ins RESET-Register wird ein Wert von $2 eingeschrieben. Das führt dann dazu, daß der Alarm-Ausgang ständig im 1 Hz- und 16 Hz-Takt gegen Masse schaltet.

Hat man nun (wie im Schaltbild angegeben) beispielsweise eine Leuchtdiode angeschlossen, so flackert diese im ständigen Takt,statt nur dann zu leuchten, wenn ein Alarm ausgelöst wird.

Außerdem wird bei jedem GEMDOS-Aufruf, der zum Verlassen eines Programms dient, der Uhrenchip ausgelesen. Dazu testet aber das TOS wieder jedesmal, ob der Uhrenchip denn vorhanden ist, was dann zu dem bereits beschriebenen Überschreiben der Register Alarmzeit Minuten-Einer und -Zehner und dem Rücksetzen des Alarmbits im Modusregister führt.

Will man also die Alarm-Funktion des Uhrenchips zumindest bei eingeschaltetem Rechner benutzen (z. B. als Weckfunktion durch Aufleuchten der Leuchtdiode. Statt der Leuchtdiode läßt sich aber auch, wie in Abbildung K.1 bereits angedeutet, eine Treiberschaltung ansteuern, über die ein Relais geschaltet wird, das dann z. B. einen Netzverbraucher einschaltet.), so sollte man zum Stellen von Uhrzeit/Datum nicht die XBIOS- bzw. GEMDOS-Funktion benutzen, sondern eigene Routinen verwenden, welche die Alarmfunktion des Uhrenchips unterstützen und nicht unterdrücken!

Eine Möglichkeit für eine eingeschränkte Nutzung der Alarmfunktion besteht darin, GEMDOS-Aufrufe, die zu einer Veränderung der Alarmregister führen, abzufangen. Dazu muß man den Vektor, der auf den GEMDOS-Dispatcher im ROM zeigt, auf eine eigene Routine umleiten. Diese testet bei jedem GEMDOS-Aufruf, ob Alarmregister zerstört werden könnten. Wäre das der Fall, liest man sich den gegenwärtigen Stand der Alarmregister aus und speichert diesen zwischen. Anschließend kann man den GEMDOS-Aufruf ausführen, der die Alarmdaten im Uhrenchip zerstört, und restauriert nach diesem GEMDOS-Aufruf wieder die Alarmregister mit den zwischengespeicherten Werten.

Anhang L: Der Coprozessor MC68881 im MEGA ST(E)

Im MEGA ST ist über den MEGABUS-Anschluß eine Karte mit einem 16 MHz-Coprozessor des Typs MC68881 nachrüstbar. Beim MEGA STE dagegen ist dieser Coprozessor im Platinenlayout bereits vorgesehen und teilweise direkt mit eingebaut. Da aber in beiden Rechnertypen weiterhin eine MC68000er-CPU zum Einsatz kommt, muß das Kommunikationsprotokoll zwischen CPU und Coprozessor per Software nachgebildet werden. Erst die Prozessoren 68020/68030 beherrschen dieses Protokoll selbst. Siehe dazu auch die Erläuterungen im Kapitel "Damit's noch schneller geht – der Coprozessor MC68882".

Beim MC68000 wird dagegen bei Erkennen eines Coprozessor-Befehls ein sogenannter Line-F-Trap ausgelöst, weil das höchstwertige Nibble des ersten Coprozessor-Befehlsworts $F lautet.

Da in früheren TOS-Versionen der Line-F-Trap von ATARI für bestimmte Betriebssystemcalls verwendet wurde, ist es beim Einsatz eines Coprozessors evtl. erforderlich, diesen Trap abzufangen und Coprozessorbefehle entsprechend dem nachfolgend näher erläuterten Kommunikationsprotokoll an die FPU weiterzuleiten (wie so ein Treiber in der Praxis aussehen könnte, war mal in der c't 4/90, "Schneller rechnen", beschrieben.) Für die MC68000-CPU erscheint der Coprozessor wie ein Peripheriebaustein, der bestimmte Register besitzt und darüber angesprochen wird.

Die Coprozessor Interface Register (CIR)

Für den Austausch von Signalen und Daten zwischen CPU und Coprozessor sind eine ganze Zahl von internen Registern in der FPU vorgesehen. Das Registermodell dieser *Coprozessor Interface Register* (*CIR*) stellt sich für den Programmierer wie folgt dar:

Adresse:

$FF FA40	Response	(READ)	WORD-Zugriffe
$FF FA42	Control	(WRITE)	
$FF FA44	Save	(READ)	

Adresse:

$FF FA46	Restore (R/W)	
$FF FA48	Operation Word (*)	WORD-Zugriffe
$FF FA4A	Command (WRITE)	
$FF FA4C	(Reserved) (*)	
$FF FA4E	Condition (WRITE)	
$FF FA50	Operand (R/W)	LONG
$FF FA54	Register Select (R)	WORD-Zugriffe
$FF FA56	(Reserved) (*)	
$FF FA58	Instruction Address (WRITE)	LONG
$FF FA5C	Operand Address (*)	LONG

Die mit (*) versehenen Register haben (noch) keine Funktion.

Über das *Response*-Register erfährt die CPU durch Auslesen, wie weit die FPU mit der Abarbeitung von einem Coprozessor-Befehl gekommen ist bzw. was die FPU von der CPU als nächstes erwartet. Es müssen ja evtl. noch Operanden in die FPU gebracht oder auch von dort abgeholt werden. Außerdem werden über dieses Register Informationen über an den Coprozessor gestellte Bedingungsabfragen ("wahr" oder "falsch") ausgetauscht.

Die CPU muß also bei der Emulation des Protokolls mit dem Coprozessor dieses Register auslesen und nach Auswertung entsprechend darauf reagieren.

Die im Response-Register übergebenen 16 Bit-Words werden auch als sogenannte *Response Primitives* bezeichnet. Die FPU beginnt mit der Ausführung eines neuen Befehls immer erst nach dem erstmaligen Auslesen des Response-Registers!

Ein Schreibzugriff auf das *Control*-Register (egal mit welchem Wert!) bewirkt beim MC 68881 einen Abbruch aller FPU-Operationen! Der MC68881 ist damit bereit für eine neue Operation. Das Ziel-FP-Datenregister einer so unterbrochenen Operation ist unbestimmt!

Die *Save*- und *Restore*-Register wird man im ST(E) wahrscheinlich selten gebrauchen, da sie für das Sichern und Wiederherstellen des Coprozessor-internen Zustands z. B. wegen eines Taskwechsels verwendet werden können. Deshalb soll hier nicht weiter darauf eingegangen werden.

Was die FPU eigentlich tun soll, erfährt sie in der Regel über das *Command*-Register! Durch einen Schreibzugriff wird der "normale" Dialog zwischen CPU und Coprozessor gestartet. Hier "landet" das zweite Befehlsword eines "normalen" Coprozessor-Befehls.

Das *Condition*-Register wird verwendet, wenn eine Bedingungsabfrage ausgeführt werden soll. Dabei landet dann das zweite Coprozessor-Befehlsword (mit den Bedingungsabfrage-Bits) in diesem Register! Ob die Bedingung erfüllt wurde oder nicht, teilt der Coprozessor dann über das Response-Register mit.

Das *Operand*-Register kann beschrieben und gelesen werden. Es dient dem Transfer von Daten zwischen CPU und FPU. Die Breite beträgt zwar 32 Bit, aber es können natürlich auch Operanden in Byte- oder Word-Größe übergeben werden. Wichtig dabei ist, daß alle Operanden in diesem Register immer "linksbündig" übergeben werden. D. h. beispielsweise, daß Byte- Operanden in den Bits 24..31 des Registers übergeben werden!

Wenn mehrere FP-Datenregister mit dem FMOVEM-Befehl transportiert werden sollen, so muß die FPU der CPU natürlich irgendwie mitteilen können, welche und wie viele FP-Register dies sind. Dazu werden die oberen acht Bits des *Register select*-Registers benutzt. Sie enthalten die Registermaske. Ein gesetztes Bit signalisiert jeweils ein zu transferierendes FP-Register. Dabei kommt es gar nicht mal auf die Reihenfolge der gesetzten Bits an, sondern vielmehr auf die Anzahl! So weiß die CPU dann, wie viele Registerinhalte bewegt werden müssen (jedes FP-Datenregister benötigt 12 Bytes Platz!).

Das *Instruction Address*-Register kann bei der Programmierung eines MC68881 außer acht gelassen werden. Hier wird dem Coprozessor bei Bedarf (nur beim MC68882 mit seiner parallelen Betriebsweise für verschiedene Befehle) der augenblickliche Wert des CPU-Programmzählers übergeben. Der MC68882 verlangt das immer dann, wenn eine Exception durch einen FPU-Befehl ausgelöst werden könnte. Damit hat ein Trap-Handler für FPU-Exceptions dann später die Möglichkeit, den Befehl, bei der die Exception auftrat, festzustellen und dort wieder aufzusetzen.

Das Protokoll

Die Kommunikation mit dem Coprozessor muß üblicherweise nach dem folgenden Schema ablaufen:

1. Die CPU trifft auf einen Coprozessor(= Line-F)-Befehl und löst eine Ausnahmeverarbeitung aus. In dieser Ausnahmeverarbeitungsroutine muß dann das entsprechende CIR mit befehlsspezifischen Informationen geladen werden.

   Bei "normalen" Befehlen wird das Command-Register mit dem zweiten Coprozessor-Befehlswort geladen und so der Dialog CPU <-> FPU gestartet.

   Bedingungsanfragen an die FPU werden durch Einschreiben des zweiten Befehlsworts in das Condition-Register gestartet.

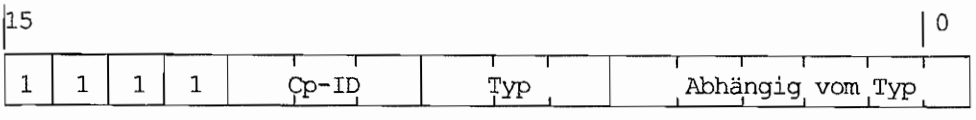
2. Durch Auslesen des Response-Registers erfährt die CPU über die Response primitives, ob die FPU noch mehr Daten benötigt. Folgende Möglichkeiten ergeben sich:

   - Die FPU ist "beschäftigt"! Also kann sich die CPU erstmal um evtl. aufgelaufene Interrupts usw. kümmern. Nach "getaner Arbeit" wird dann wieder im Response-Register der Stand der Dinge geprüft.
   - Es liegt eine Exception-Bedingung von seiten der FPU vor. Der entsprechende Exception-Vektor geht aus der Response primitive hervor. Die CPU kann die Ausnahmeverarbeitung einleiten.
   - Die FPU möchte etwas durch die CPU erledigt haben (Adreßberechnung für zu transportierende Daten und die Durchführung des Transports durch die CPU. Außerdem kann die FPU signalisieren, daß die CPU nach "getaner Arbeit" wieder im Response-Register nachschaut.).
   - Die FPU ist mit allem versorgt und signalisiert der CPU, daß sie sich mit "ihrem eigenen Kram" beschäftigen soll.

Die Durchführung des Protokolls hängt natürlich von den im ersten Coprozessor-Befehlswort enthaltenen Daten ab. Weitere wichtige Informationen sind dem zweiten Befehlswort zu entnehmen. Deshalb soll hier kurz auf deren Struktur eingegangen werden.

Das erste Coprozessor-Befehlswort

hat folgenden Aufbau:

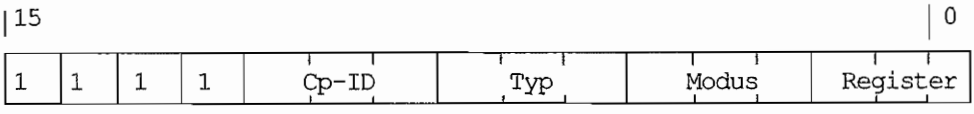


Durch das Cp-ID-Feld können max. acht Coprozessoren angesprochen werden.

Standardmäßig verwendet Motorola in seinem Assembler jedoch nur die Cp-ID = 001_{bin}. Im Typ-Feld ist kodiert, um welche Art Coprozessor-Befehl es sich handelt:

Typ	Bedeutung
0 0 0	Normaler Befehl (Rechenoperation oder Datentransport-Befehl)
0 0 1	FDBcc, FScc, FTRAPcc
0 1 0	FBcc.W
0 1 1	FBcc.L
1 0 0	FSAVE (internen FPU-Zustand sichern)
1 0 1	FRESTORE (internen FPU-Status wiederherstellen)
1 1 0	(Reserved)
1 1 1	(Reserved)

Die Bedeutung der Bits 0..5 ist vom Befehlstyp abhängig. Bei normalen Befehlen bestimmen die Bits 3..5 (Modus-Bits) den Adressierungsmodus, und in Bit 0..2 (Register-Bits) steht dann die Register-Nummer der CPU (wenn denn ein Register beteiligt ist).



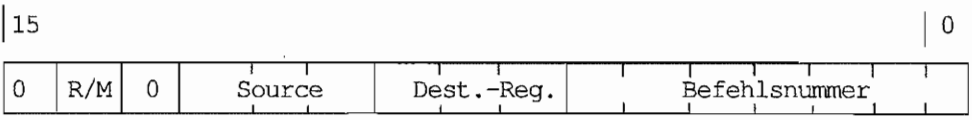
Adressierungs-modus	Mode Bits	Register-Bits
Dn	0 0 0	Reg.-Nummer des Datenreg. der CPU
An	0 0 1	Reg.-Nummer des Adreßreg. der CPU
(An)	0 1 0	Reg.-Nummer des Adreßreg. der CPU
(An)+	0 1 1	Reg.-Nummer des Adreßreg. der CPU
–(An)	1 0 0	Reg.-Nummer des Adreßreg. der CPU
(d_{16},An)	1 0 1	Reg.-Nummer des Adreßreg. der CPU
(d_8,An,Xn)	1 1 0	Reg.-Nummer des Adreßreg. der CPU
(bd,An,Xn)	1 1 0	Reg.-Nummer des Adreßreg. der CPU (*)
([bd,An,Xn],od)	1 1 0	Reg.-Nummer des Adreßreg. der CPU (*)
([bd,An],Xn,od)	1 1 0	Reg.-Nummer des Adreßreg. der CPU (*)
(xxx).W	1 1 1	0 0 0
(xxx).L	1 1 1	0 0 1
#<data>	1 1 1	1 0 0
(d_{16},PC)	1 1 1	0 1 0
(d_8,PC,Xn)	1 1 1	0 1 1

Adressierungs-modus	Mode Bits	Register-Bits
(bd,PC,Xn)	1 1 1	0 1 1 (*)
([bd,PC,Xn],od)	1 1 1	0 1 1 (*)
([bd,PC],Xn,od)	1 1 1	0 1 1 (*)

(*)= *Adressierungsart existiert beim MC68000/68010 nicht!*

Der Aufbau des zweiten Coprozessor-Befehlsworts

Das *Command-Word* hat bei allen arithmetischen Befehlen und den Datentransportbefehlen (außer FSINCOS, FMOVECR, FMOVE from FPcr und FMOVEM) den folgenden Aufbau:



Folgende Zuordnung gilt, wenn das R/M-Bit gesetzt ist:

Source-bits (*)	Bedeutung
0 0 0	Datenformat ist LONG
0 0 1	Datenformat ist Single Real
0 1 0	Datenformat ist Extended Real
0 1 1	Datenformat ist Packed Decimal
1 0 0	Datenformat ist WORD
1 0 1	Datenformat ist Double Real
1 1 0	Datenformat ist BYTE

(*)= *Nur beim Befehl "FMOVE from FPn" verwendet!*

Wenn das *R/M-Bit* allerdings gelöscht ist, wird in den Sourcebits die Nummer des Floatingpoint-Registers (FPn) angegeben, welches als Quelle benutzt wird.

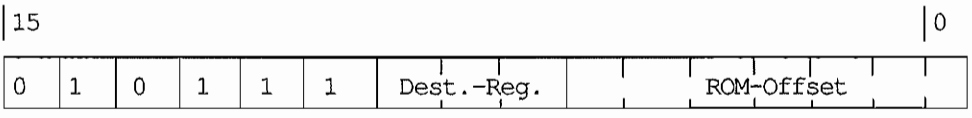
Die drei *Dest.-Reg-Bits* geben die Nummer des Floating-Point-Registers an, welches als Ziel verwendet werden soll. In der *Befehlsnummer* ist die Nummer des jeweiligen Befehls enthalten.

(Die Befehle sind im Kapitel "Damit's noch schneller geht – Der Coprozessor MC68882" von ihrer Funktion her kurz beschrieben!):

Befehls-nummer	Befehlsmnemonic	Befehls-nummer	Befehlsmnemonic
0	FMOVE to FPn	$18	FABS
1	FINT	$19	FCOSH
2	FSINH	$1A	FNEG
3	FINTRZ	$1C	FACOS
4	FSQRT	$1D	FCOS
6	FLOGNP1	$1E	FGETEXP
8	FETOXM1	$1F	FGETMAN
9	FTANH	$20	FDIV
$A	FATAN	$21	FMOD
$C	FASIN	$22	FADD
$D	FATANH	$23	FMUL
$E	FSIN	$24	FSGLDIV
$F	FTAN	$25	FREM
$10	FETOX	$26	FSCALE
$11	FTWOTOX	$27	FSGLMUL
$12	FTENTOX	$28	FSUB
$14	FLOGN	$38	FCMP
$15	FLOG10	$3A	FTST
$16	FLOG2		

Der Befehl *FSINCOS* berechnet ja zugleich Sinus *und* Kosinus einer Zahl. Er hat als Befehlsnummer den Wert von $3X. Dabei muß für X die Nummer des zweiten FP-Zielregisters angegeben werden, wo dann der Wert des berechneten Kosinus landet!

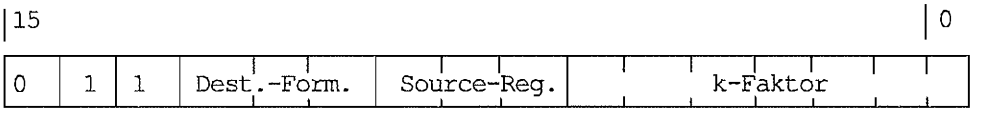
Mit dem *FMOVECR*-Befehl kann man häufig benötigte Konstanten (Pi, e usw.) aus einem Konstanten-ROM der FPU holen. Der Wert für den Offset in dieses ROM wird dazu in den unteren sieben Bits des Command-Words eingetragen.



Eine Liste dieser Konstanten mit Offset-Werten ist ebenfalls in dem Kapitel "Damit's noch schneller geht – Der Coprozessor MC68882" enthalten.

Die Dest.-Reg.-Bits müssen mit der entsprechenden Nummer des Ziel-FP-Registers geladen werden.

Mit *FMOVE from FPn* wird ein FP-Register ausgelesen und an einer Zieladresse (darf auch ein CPU-Register sein, wenn die Operandenlänge Byte, Word oder Long ist!) abgelegt.



In den *Source-Reg.-Bits* ist die Nummer des Quell-FP-Registers anzugeben.

Bei *k-Faktor* ist eine Formatangabe zu machen, wenn das Ergebnis im Packed Decimal-Format erstellt werden soll (Näheres zum k-Faktor im Kapitel "Damit's noch schneller geht – Der Coprozessor MC68882").

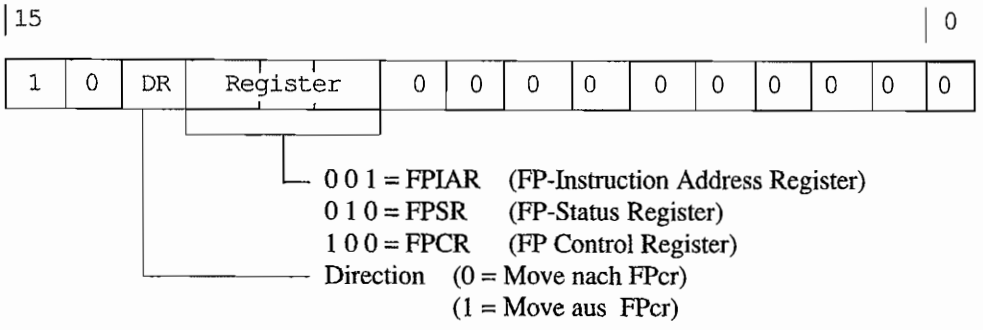
Die Angabe muß im Zweierkomplement erfolgen.

Wenn mit dynamischem k-Faktor gearbeitet wird (der k-Faktor steht dann in einem Datenreg. der CPU), so ist in den Bits 4..6 die Nummer des CPU-Datenregisters anzugeben (Bits 0..3 bleiben dann gelöscht!).

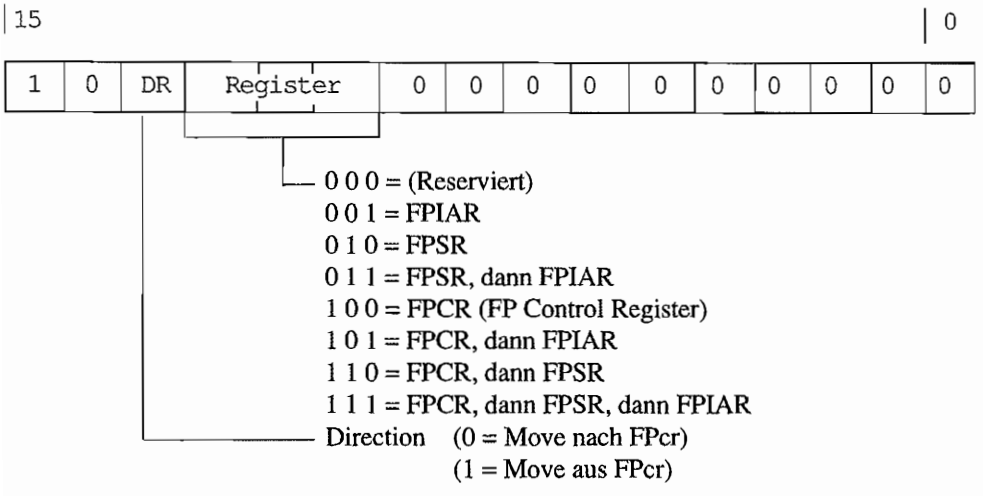
Die Dest.-Form.-Bits sind nach der folgenden Tabelle kodiert:

Dest.-Format	**Bedeutung**
0 0 0	Datenformat ist LONG
0 0 1	Datenformat ist Single Real
0 1 0	Datenformat ist Extended Real
0 1 1	Datenformat ist Packed Decimal (statischer k-Faktor)
1 0 0	Datenformat ist WORD
1 0 1	Datenformat ist Double Real
1 1 0	Datenformat ist BYTE
1 1 1	Datenformat ist Packed Decimal (dynamischer k-Faktor)

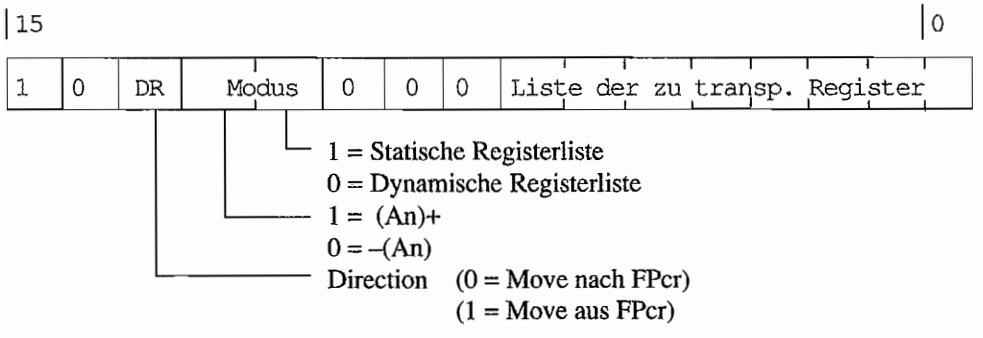
Mit *FMOVE FPcr* wird auf die Control-Register der FPU zugegriffen. Das Command-Word dazu sieht folgendermaßen aus:



Mehrere Control-Register der FPU werden mit dem *FMOVEM FPcr* bewegt.



Mehrere FP-*Daten*registerinhalte können ebenfalls mit einem *FMOVEM FPn*-Befehl transportiert werden.



Die Modus-Bits legen den Adressierungsmodus fest und wie die Liste der Register vorliegt.

Statische Liste

Die untersten acht Bits stellen eine Auswahlmaske für die zu transportierenden Register dar. Wenn ein Register bewegt werden soll, ist das entsprechende Bit gesetzt, sonst gelöscht!

Aufbau der Liste:

	Bitnummer							
Adr.-Art	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
-(An)	FP7	FP6	FP5	FP4	FP3	FP2	FP1	FP0
(An)+	FP0	FP1	FP2	FP3	FP4	FP5	FP6	FP7

Dynamische Liste

Die FP-Register-Auswahlmaske steht in einem CPU-Datenregister. Die Nummer des CPU-Datenregisters ist in den Bits 4..6 des Command-Words eingetragen. Der Listenaufbau ist der gleiche wie bei der statischen Liste, nur daß eben die acht untersten Bits des CPU-Datenregisters diese Maske enthalten müssen!

Nun zu den "nicht normalen" Coprozessor-Befehlen. FPU-Anweisungen, die FPU-Bedingungscodes auswerten, wie *FScc*, *FDBcc*, *FTRAPcc* und *FBcc* enthalten ja schon im ersten Coprozessor-Befehlswort (das mit $F.. beginnt!) eine andere Typkennung. Damit werden sie von den "normalen Befehlen" abgegrenzt.

Der Aufbau dieser Befehle ist wegen der unterschiedlichen Benutzung der unteren sechs Bits des ersten Coprozessor-Befehlsworts deshalb nochmal komplett mit allen erforderlichen Befehlsworten angegeben.

***FScc*-Befehl ($FF -> Reg./Speicherstelle, wenn Bedingung erfüllt)**

15															0
1	1	1	1	CP-ID			0	0	1	Modus			Register		
0	0	0	0	0	0	0	0	0	0	Bedingungsabfrage					

Modus- und Register-Bits haben die gleiche Bedeutung, wie bereits bei der Erläuterung des ersten Coprozessor-Befehlsworts gezeigt. Die Bedingungsabfrage-Bits enthalten dabei das Bitmuster, mit der die Abfrage kodiert ist.

Die so "formulierte" Bedingung ist dann von der FPU auszuwerten und am Ende der Bearbeitung mit "Ja" oder "Nein" an die CPU zu beantworten.

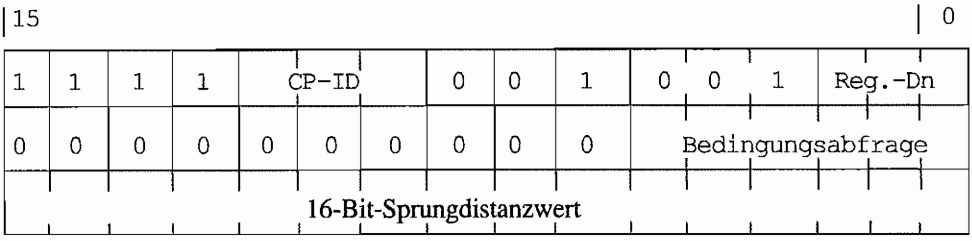
Die Beantwortung erfolgt dann über ein Bit im Response-Register.

Bedingungs-Bits	Mnemonic	Bedeutung
0 0 0 0 0 0	F	Falsch
0 0 0 0 0 1	EQ	Gleich
0 0 0 0 1 0	OGT	Größer
0 0 0 0 1 1	OGE	Gleich oder größer
0 0 0 1 0 0	OLT	Kleiner
0 0 0 1 0 1	OLE	Kleiner oder gleich
0 0 0 1 1 0	OGL	Größer oder kleiner
0 0 0 1 1 1	OR	Größer, kleiner oder gleich (es ist kein NAN!)
0 0 1 0 0 0	UN	(Die gleichen Bedingungsbits wie vorher,
0 0 1 0 0 1	UEQ	nur daß hiermit geprüft wird, ob
0 0 1 0 1 1	UGT	eine nicht zulässige Zahl (NAN)
0 0 1 1 0 0	UGE	entstanden ist bzw. bei der letzten
0 0 1 1 0 1	ULE	Verknüpfung mit "im Spiel war"!)
0 0 1 1 1 0	NE	" " " " "
0 0 1 1 1 1	T	" " " " "
0 1 0 0 0 0	SF	In diesem Teil der Tabelle werden die
0 1 0 0 0 1	SEQ	gleichen Bedingungsabfragen wie vorher
0 1 0 0 1 0	GT	nochmals mit ihren (aber anderen) Mnemonics
0 1 0 0 1 1	GE	aufgeführt.
0 1 0 1 0 0	LT	
0 1 0 1 0 1	LE	Durch das gesetzte Bit 4 wird der
0 1 0 1 1 0	GL	FPU bei solchen Bedingungsabfragen
0 1 0 1 1 1	GLE	signalisiert, daß eine Exception
0 1 1 0 0 0	NGLE	ausgelöst werden soll, wenn bei
0 1 1 0 0 1	NGL	der Bedingungsabfrage ein NAN
0 1 1 0 1 0	NLE	mit "im Spiel war"!
0 1 1 0 1 1	NLT	(Das BSUN-Bit im EXC-Byte des FPSR
0 1 1 1 0 0	NGE	wird dann gesetzt!)
0 1 1 1 0 1	NGT	
0 1 1 1 1 0	SNE	
0 1 1 1 1 1	ST	

FDBcc-Befehl

Syntax: `FDBcc Dn,<label>`

Funktion:
```
IF  Bedingung cc wahr, THEN No operation
    ELSE   Dn := Dn - 1;
       IF  Dn <> -1, THEN go to <label>
```



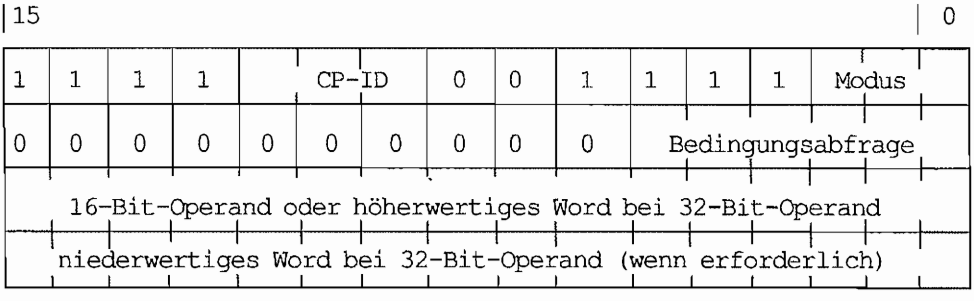
Bei *Reg.-Dn* ist die Nummer des CPU-Datenregisters einzutragen, welches dekrementiert wird, wenn die Bedingung nicht erfüllt ist. Die *Bedingungsabfrage-Bits* sind wie in der Tabelle zuvor zu setzen.

Das dritte Coprozessor-Befehlsword enthält dann den Sprungdistanzwert im Zweierkomplement.

FTRAPcc-Befehl

Syntax:
```
FTRAPcc.W   #<data>
FTRAPcc.L   #<data>
```

Funktion: Wenn die Bedingung cc erfüllt ist, wird ein Trap ausgelöst und der evtl. folgende Operand dem Traphandler übergeben.

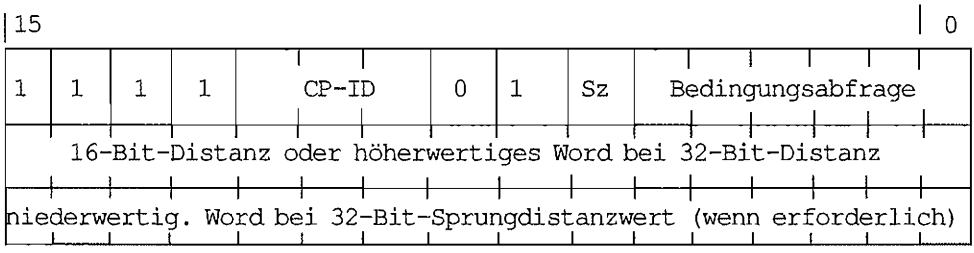


Modus: 0 1 0 Operand ist 16 Bits lang
0 1 1 Operand ist 32 Bits lang
1 0 0 Es folgt kein Operand!

***FBcc*-Befehl**

Syntax: `FBcc.W <label>`
`FBcc.L <label>`

Funktion: Wenn die Bedingung cc erfüllt ist, wird zum <label> verzweigt.



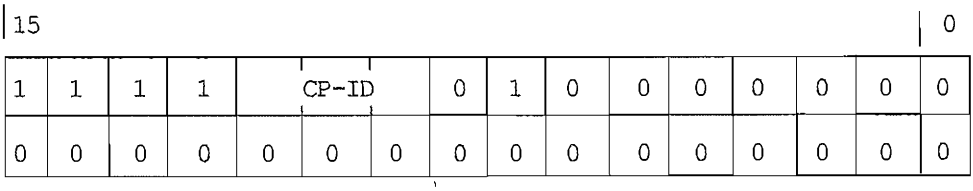
Sz 0 = 16 Bit-Sprungdistanzwert
1 = 32 Bit-Sprungdistanzwert

Der Sprungdistanzwert ist im Zweierkomplement anzugeben!

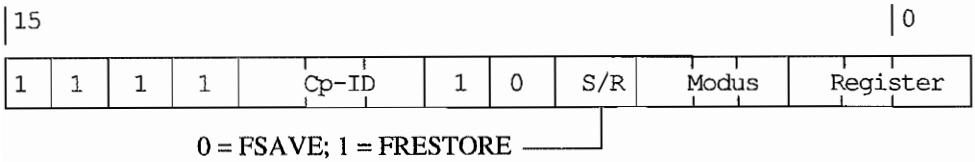
***FNOP*-Befehl**

Syntax: `FNOP`

Funktion: No Operation!



Es folgen die Spezialbefehle für das Sichern und Wiederherstellen des FPU-Zustandes – z. B. wegen eines Taskwechsels in Multitasking-Anwendungen. Sie unterscheiden sich im ersten Coprozessor-Befehlsword nur im S/R-Bit. Ein zweites Befehlsword ist nicht erforderlich!



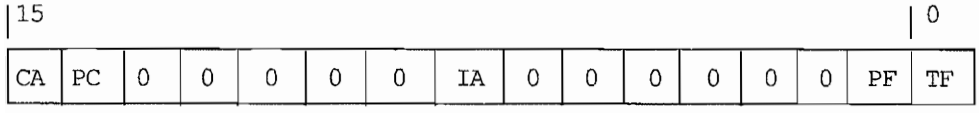
Modus- und Register-Bits haben die gleiche Bedeutung, wie schon bei den "normalen" Befehlen erläutert.

Die Coprozessor Response primitives

Wie ja schon weiter oben erwähnt, kann die CPU durch Auslesen des Response-Registers feststellen, wie weit die FPU ist bzw. was diese noch z. B. an Daten benötigt. Die FPU geht davon aus, daß nach dem Auslesen des Response-Registers der entsprechende Inhalt von der CPU übernommen wurde und entsprechend ausgewertet wird! Das bedeutet, daß die FPU schon intern weiterarbeiten kann, bis sie wieder über das Response-Register weitere Service-Anfragen an die CPU signalisiert! Diese als Response primitives bezeichneten 16-Bit-Worte sollen nun kurz erläutert werden.

Null primitive

Diese Response primitive wird hauptsächlich zur Synchronisation zwischen CPU und FPU benutzt. Daran kann die CPU nämlich erkennen, ob die FPU noch beschäftigt ist oder ob später nochmal "nachgeschaut" werden soll!



CA Come again-Bit. Wenn das Bit gesetzt ist, bedeutet das, daß die CPU nach Erledigung der angeforderten Aufgaben das Response-Register nochmals auslesen soll.

PC Pass-Program-Counter-Bit. Ein gesetztes Bit bedeutet: Programmzählerinhalt soll sofort der FPU im Instruction Address-Register übergeben werden. Kann beim MC 68881 ignoriert werden.

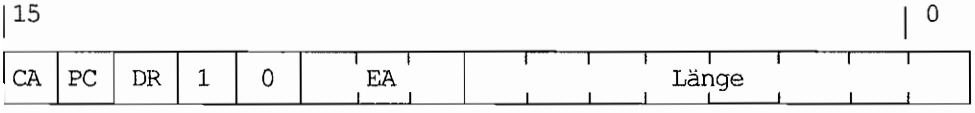
IA Interrupts-Allowed-Bit. Durch ein gesetztes Bit signalisiert die FPU, daß sich die CPU (bei Bedarf) um anstehende Interrupts kümmern darf.

PF Process-Finished-Bit. Ein gesetztes Bit meldet der CPU, daß die FPU zu neuen Taten bereit ist. Ist das Bit gelöscht, so ist die FPU noch mit einer Operation beschäftigt.

TF True/False-Bit. Das Resultat einer Bedingungsanfrage an die FPU wird hiermit signalisiert. TF = 0: Bedingung nicht erfüllt! TF = 1: Na, raten Sie mal!

Evaluate effective address and transfer data primitive

Diese Response primitive tritt auf, wenn Daten zwischen FP-Daten- oder -Control-Registern und CPU-Registern oder Speicherbereichen ausgetauscht werden sollen. Die CPU hat anhand der Daten in dieser Primitive und den im ersten Coprozessor-Befehlswort enthaltenen Mode- und Register-Bits die gewünschte "Effektive Adresse" zu ermitteln und eine entsprechende Anzahl von Bytes (wie viele, steht in "Länge"!) über das Operanden-Register zu transportieren.



Das CA- und PC-Bit haben die gleiche Bedeutung, wie bei der Null primitive beschrieben.

DR Direction-Bit. Bestimmt die Datentransportrichtung. Ein gesetztes Bit sagt der CPU, daß aus der FPU (über das Operanden-Register) gelesen werden soll.

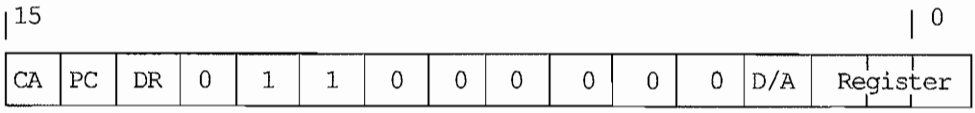
EA Kodierbits für die Klasse der Effektiven Adresse. Hiermit gibt die FPU die Klasse der effektiven Adresse an, die ermittelt und dann zum Datentransport verwendet werden soll. Die genaue Bestimmung, ob die effektive Adresse ein CPU-Register, Speicher oder Konstante ist, wird ja durch die Mode- und Register-Bits im ersten Coprozessor-Befehlswort vorgenommen. Mit den EA-Bits kann nur noch mal geprüft werden, ob die vereinbarte Adressierungsklasse auch mit der Kodierung in den Mode- und Register-Bits übereinstimmt. Kann ignoriert werden. Wer sich für die genaue Zuordnung von Adressierungsklassen zu Adressierungsarten interessiert, sei auf das "MC68881/68882 Floating-Point Coprozessor User's Manual" von Motorola verwiesen.

Länge Das Längen-Feld gibt die Anzahl der Bytes an, die beim Datentransport über das Operanden-Register bewegt werden müssen. Mögliche Werte sind 1 (Byte), 2 (Word), 4 (Long oder Single Precision), 8 (2 FPCR oder Double Precision) oder 12 (alle 3 FPCR, Extended Precision oder Packed dezimal).

Transfer single main processor register primitive

Diese Primitive wird verwendet, um den Inhalt eines CPU-Registers zur FPU zu übertragen (übergeben wird der CPU-Registerinhalt wieder "linksbündig" über das Operand-Register der FPU!).

Verwendung findet diese Primitive bei dem Dialog anläßlich eines FMOVEM FPn-Befehls. Wenn nämlich die Liste der zu übertragenden FP-Datenregister als dynamische Liste vorliegt (also als Bitmaske in einem CPU-Datenregister!), bekommt die CPU hiermit angezeigt, welches CPU-Register diese Liste enthält. Die CPU hat dann den Registerinhalt im Operandenregister zu übergeben.



Das CA- und PC-Bit haben die gleiche Bedeutung, wie bei der Null primitive beschrieben. Das DR-Bit legt wieder die Transportrichtung fest (1 = FPU => extern).

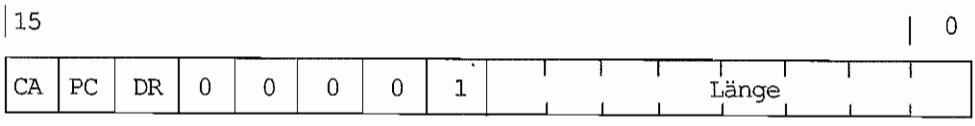
D/A Data-/Address-Register-Bit. Ein gelöschtes Bit signalisiert, daß ein CPU-Datenregister gemeint ist. Sonst geht's um ein Adreßregister (kommt nicht vor!).

Regist. Hiermit wird die Registernummer (0..7) übergeben.

Transfer multiple coprocessor registers primitive

Wird benutzt beim Transfer von mehreren FPU-Registern. Die Quelle oder das Ziel des Datentransports (die "effektive Adresse") geht ja bereits aus den Mode- und Register-Bits im ersten Coprozessor-Befehlswort hervor. Die Liste der zu transferierenden Register muß die CPU nach Erkennen dieser Primitive aus dem Register select-Register auslesen!

Jedes gesetzte Bit entspricht dabei einem Transfer von in "Länge" angegebener Menge von Bytes!



Das CA-, PC- und DR-Bit haben die gleiche Bedeutung, wie bereits zuvor beschrieben.

Länge Operandenlänge in Bytes.

Auf die Response primitives für die Exception-Bearbeitung soll hier nicht näher eingegangen werden. Wenn der Coprozessor als Peripheriebaustein betrieben wird (im ST(E) der Fall), kommen diese speziellen Anwendungen ohnehin kaum vor!

Wer aber auch das bis ins Detail wissen will (muß), sollte auf das bereits mehrmals erwähnte MC68881/68882-Usermanual zurückgreifen.

Anhang M: Kurzübersicht über die Hardware-Register

Beim TT finden sich die Hardware-Register zum einen im ST-Adreßraum, zum anderen am oberen "Ende" des Speicherbereichs (ab Adresse $FFFF 8000). Angegeben ist jeweils nur die Adresse im ST-Adreßraum!

Speicherkonfiguration

Adresse	Zugriff	Bezeichnung	Label
$FF 8001	R/W	**Memory-Config.-Register** (Byte) ----CCCC Bank 1 00 — 64 KBit Chips (128 KByte/Bank) 01 — 256 KBit Chips (512 KByte/Bank) 10 — 1 MBit Chips (2 MByte/Bank) 11 — Reserviert Bank 0 00 — 64 KBit Chips (128 KByte/Bank) 01 — 256 KBit Chips (512 KByte/Bank) 10 — 1 MBit Chips (2 MByte/Bank) 11 — Reserviert *Achtung!* Diese Bitbelegung gilt nicht beim TT!	memconf

Videocontroller

Adresse	Zugriff	Bezeichnung	Label
$FF 8201	R/W	**Video-Base-Reg. High** (Byte)	dbaseh
$FF 8203	R/W	**Video-Base-Reg. Mid** (Byte)	dbasel
$FF 8205	R(*)	**Video-Adr.-Counter High** (Byte)	vcounthi
$FF 8207	R(*)	**Video-Adr.-Counter Mid** (Byte)	vcountmid
$FF 8209	R(*)	**Video-Adr.-Counter Low** (Byte)	vcountlow
		(*) = Beim STE ist auch Schreiben möglich!	

Adresse	Zugriff	Bezeichnung	Label
$FF 820A	R/W	**Sync-Mode-Register** (Byte) ------V S S: Synchronisation (0=Intern, 1=Extern) (Beim TT genau andersherum!!!) V: Vertikalfrequenz (0=60 Hz, 1=50 Hz) (Nicht beim TT!)	syncmode
$FF 820C	R/W	**Video-Base-Reg. Low** (Byte) (Nur bei TT/STE-Maschinen!)	dbaselow
$FF 8240	R/W	---- rRRR gGGG bBBB **Farbreg. 0** 0 = Normal Monochrom 1 = Invertiert Monochrom	color0
$FF 8242	R/W	---- rRRR gGGG bBBB **Farbreg. 1**	color1
$FF 8244	R/W	---- rRRR gGGG bBBB **Farbreg. 2**	color2
$FF 8246	R/W	---- rRRR gGGG bBBB **Farbreg. 3**	color3
$FF 8248	R/W	---- rRRR gGGG bBBB **Farbreg. 4**	color4
$FF 824A	R/W	---- rRRR gGGG bBBB **Farbreg. 5**	color5
$FF 824C	R/W	---- rRRR gGGG bBBB **Farbreg. 6**	color6
$FF 824E	R/W	---- rRRR gGGG bBBB **Farbreg. 7**	color7
$FF 8250	R/W	---- rRRR gGGG bBBB **Farbreg. 8**	color8
$FF 8252	R/W	---- rRRR gGGG bBBB **Farbreg. 9**	color9
$FF 8254	R/W	---- rRRR gGGG bBBB **Farbreg. 10**	color10
$FF 8256	R/W	---- rRRR gGGG bBBB **Farbreg. 11**	color11
$FF 8258	R/W	---- rRRR gGGG bBBB **Farbreg. 12**	color12
$FF 825A	R/W	---- rRRR gGGG bBBB **Farbreg. 13**	color13
$FF 825C	R/W	---- rRRR gGGG bBBB **Farbreg. 14**	color14
$FF 825E	R/W	---- rRRR gGGG bBBB **Farbreg. 15**	color15

R = Rot-Intensität (r = LSB nur bei TT/STE)
G = Grün-Intensität (g = LSB nur bei TT/STE)
B = Blau-Intensität (b = LSB nur bei TT/STE)

Adresse	Zugriff	Bezeichnung	Label
$FF 8260	R/W	**ST-Shift-Mode-Register** (Byte) ------SS Hor.- x Vert.-Auflösung 00 —— 320x200, 4 Planes (16 Farben) 01 —— 640x200, 2 Planes (4 Farben) 10 —— 640x400, 1 Plane (Monochrom) 11 —— Reserviert	shiftmd
$FF 8262	R/W	**TT-Shift-Mode Register** (Word) S--H-MMM----PPPP Reg.-Bank-Nr. für ST-Palette 000 ST-Low (320 x 200, 4 Planes) 001 ST-Mid (640 x 200, 2 Planes) 010 ST-High (640 x 400, 1 Plane) 011 <Reserviert> 100 TT-Mid (640 x 480, 4 Planes) 101 <Reserviert> 110 TT-High (1280 x 960, 1 Plane) 111 TT-Low (320 x 480, 8 Planes) Hyper-Mono-Modus Sample & Hold-Mode Register existiert nur im TT!	shift_tt
$FF 827E	R/W	**STACY Display-Steuerung** -----LD- (Byte) =1: Display Off =1: Beleuchtung Off Register existiert nur im STACY (STBOOK?)	stacydsp

TT-Farbpalettenregister

Register existieren nur beim TT!

Adresse	Zugriff	Bezeichnung		Label
$FF 8400	R/W	----RRRrGGGgBBBb	**TT-Pal. 0**	TT_col0
$FF 8402	R/W	----RRRrGGGgBBBb	**TT-Pal. 1**	TT_col1
...				
$FF 85FC	R/W	----RRRrGGGgBBBb	**TT-Pal. 254**	TT_col254
$FF 85FE	R/W	----RRRrGGGgBBBb	**TT-Pal. 255**	TT_col255
		r, g, b = Least Significant Bits!		

DMA-Controller

Adresse	Zugriff	Bezeichnung	Label
$FF 8604	R/W	**Controller-Access-Register** (nur wenn DMA-Mode-Reg., Bit 4=0!)	diskctl
oder		**Sector-Counter-Register** (nur wenn DMA-Mode-Reg., Bit 4=1!) Word-Zugriff, nur Low-Byte benutzt!	
$FF 8606	R	**DMA-Status-Register** (Word) -----SSS (Nur Low-Byte relevant!) └─ 0= Fehler bei DMA-Op. └─ 0= Sekt. Count. Reg. = 0 └─ 0= DATA REQ. v. FDC/ACSI inaktiv	fifo

Adresse	Zugriff	Bezeichnung	Label
	W	**DMA-Mode-Register** (Word) --------CCCCCCCC- CA1 (Achtung! Cmd-Phase für ACSI-Bus oder FDC-Reg.-Adressierg.) CA2 1= ACSI-, 0= FDC-Zugriff 1= Sect.-Cnt.-Register 0= Contr.-Acc.-Register 0 — Reserviert 0 — Reserviert (int. Verwendung) 1= DMA mit FDC 0= DMA mit ACSI-Bus 0= DMA-Read 1= DMA-Write an Periph.	fifo
$FF 8609	R/W	**DMA-Base+Count.-Reg.-High** (Byte)	dmahigh
$FF 860B	R/W	**DMA-Base+Count.-Reg.-Mid** (Byte)	dmamid
$FF 860D	R/W	**DMA-Base+Count.-Reg.-Low** (Byte)	dmalow

Floppy Disk Controller WD1770/1772

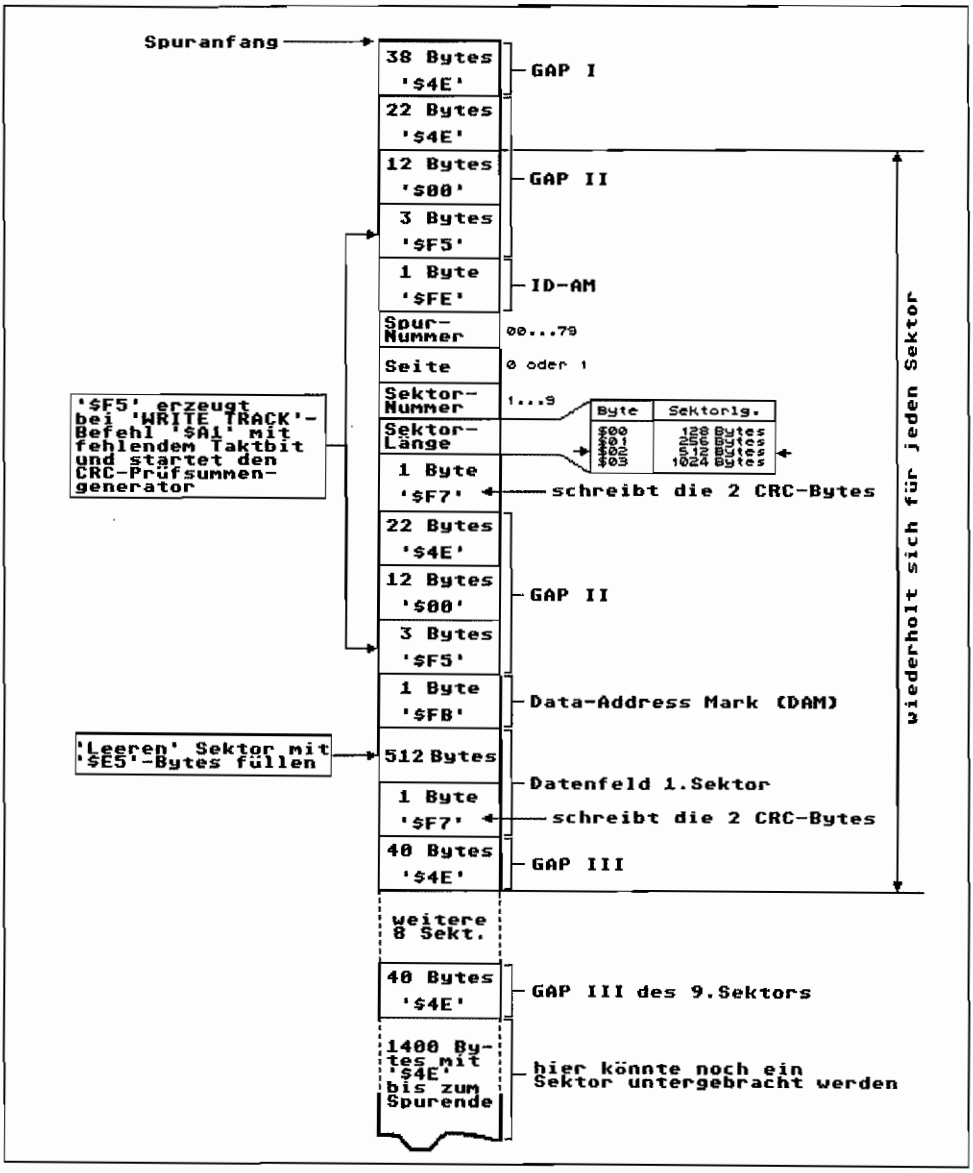


Abb. M.1: So wird ein Track im Speicher zusammengebaut und dann mit dem Befehl WRITE TRACK auf Disk geschrieben (Formatierung)

Typ	Befehl	Kommandobyte Bit 7	6	5	4	3	2	1	0
I	RESTORE	0	0	0	0	h	V	r1	r0
I	SEEK	0	0	0	1	h	V	r1	r0
I	STEP	0	0	1	u	h	V	r1	r0
I	STEP-IN	0	1	0	u	h	V	r1	r0
I	STEP-OUT	0	1	1	u	h	V	r1	r0
II	READ SECTOR	1	0	0	m	h	E	0	0
II	WRITE SECTOR	1	0	1	m	h	E	P	a0
III	READ ADDRESS	1	1	0	0	h	E	0	0
III	READ TRACK	1	1	1	0	h	E	0	0
III	WRITE TRACK	1	1	1	1	h	E	P	0
IV	FORCE INTERRUPT	1	1	0	1	I_3	I_2	I_1	I_0

Flags

für Kommandos des Typs I

h = 'Motor on'-Flag

h = 0 : 'Spin-up' eingeschaltet
h = 1 : 'Spin-up' ausgeschaltet

u = 'Update'-Flag

u = 0 : Kein 'Update'
u = 1 : Aktualis. des Track Reg.

V = 'Verify'-Flag

V = 0 : Kein Verify
V = 1 : Test auf korrekte Spur

r1,r0 = Stepimpuls-Rate

0 0 = 6 ms
0 1 = 12 ms
1 0 = 2 ms
1 1 = 3 ms

für Kommandos des Typs II + III

m = 'Mult. Sector'-Flag

m = 0 : 'Single sector'-Zugriff
m = 1 : 'Multiple sector'-Zugriff

E = 'Head settling'-Flag

E = 0 : Keine Verzögerung
E = 1 : 30ms Verzögerung

a0= 'Data addr. mark'-Bit

a0= 0 : Schreibe 'Normale' DAM
a0= 1 : Schreibe 'Deleted' DAM

P = 'Write Prekompensat.'

P = 0 : 'Prekompensation' Ein
P = 1 : 'Prekompensation' Aus

für Typ IV-Kommandos

'FORCE INTERRUPT'-Bedingungsbits

I_0 = 1 : Nicht zulässig
I_1 = 1 : Nicht zulässig
I_2 = 1 : Interrupt beim nächsten Indexpuls
I_3 = 1 : Sofortige Unterbrechung der Operation
$I_0 \ldots I_3$ = 0 : Operation ohne Interrupt abbrechen

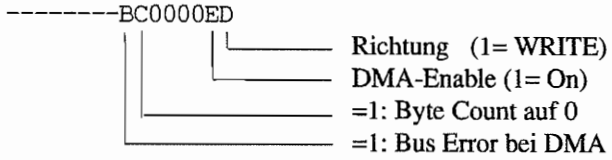
Abb. M.2: Die Kommandos des Floppy-Controllers WD1772

Bit-Nr.	Bedeutung
7	Zustand des 'Motor on'-Anschlusses ('1' = Motor on)
6	Bei READ SECTOR und READ TRACK ohne Funktion. Bei jedem WRITE-Befehl wird hiermit ein Schreibschutz angezeigt.
5	Typ I-Kommandos: Gesetzt sobald Motor-'Spin up' erfolgt ist. (6 Indexpulse nach 'Motor on') Typ II..III-Kommandos: '1'= Sektor mit 'Deleted'-Data mark. '0'= Sektor mit 'Valid'-Data mark.
4	Gesetztes Bit zeigt an, daß gewünschter Sektor, Spur oder Seite nicht gefunden wurde. (Record not found)
3	Gesetzt wenn CRC-Error festgestellt wurde.
2	Typ I-Kommandos: Zustand des 'Track 00'-Anschluß. ('1' = Kopf über Spur 00) Typ II..III-Kommandos: Wird '1' wenn die CPU (DMA) beim Datentransfer zu langsam reagiert hat. ('Lost Data'-Bit')
1	Typ I-Kommandos: Zustand des Index-Anschluß Typ II..III-Kommandos: Zustand des DRQ-Anschluß. ('1' = Data Register muß 'bedient' werden.)
0	'BUSY'-Bit. Ein Kommando ist in Bearbeitung.

Abb. M.3: Die Bedeutung der Bits im FDC-Status-Register

SCSI-DMA (Nur TT!)

Adresse	Zugriff	Bezeichnung	Label
$FF 8701	R/W	**DMA-Address-Pointer** (Highest Byte)	tt_dmabas
$FF 8703	R/W	**DMA-Address-Pointer** (High Byte)	tt_dmabas+2
$FF 8705	R/W	**DMA-Address-Pointer** (Low Byte)	tt_dmabas+4
$FF 8707	R/W	**DMA-Address-Pointer** (Lowest Byte)	tt_dmabas+6
$FF 8709	R/W	**DMA-Bytezähler** (Highest Byte)	tt_dmacnt
$FF 870B	R/W	**DMA-Bytezähler** (High Byte)	tt_dmacnt+2
$FF 870D	R/W	**DMA-Bytezähler** (Low Byte)	tt_dmacnt+4
$FF 870F	R/W	**DMA-Bytezähler** (Lowest Byte)	tt_dmacnt+6
$FF 8710	R	**Restdatenregister** (High-Word)	tt_dmarsd
$FF 8712	R	**Restdatenregister** (Low-Word)	tt_dmarsd+2
$FF 8714	R/W	**Kontrollregister** (Word)	tt_dmactl



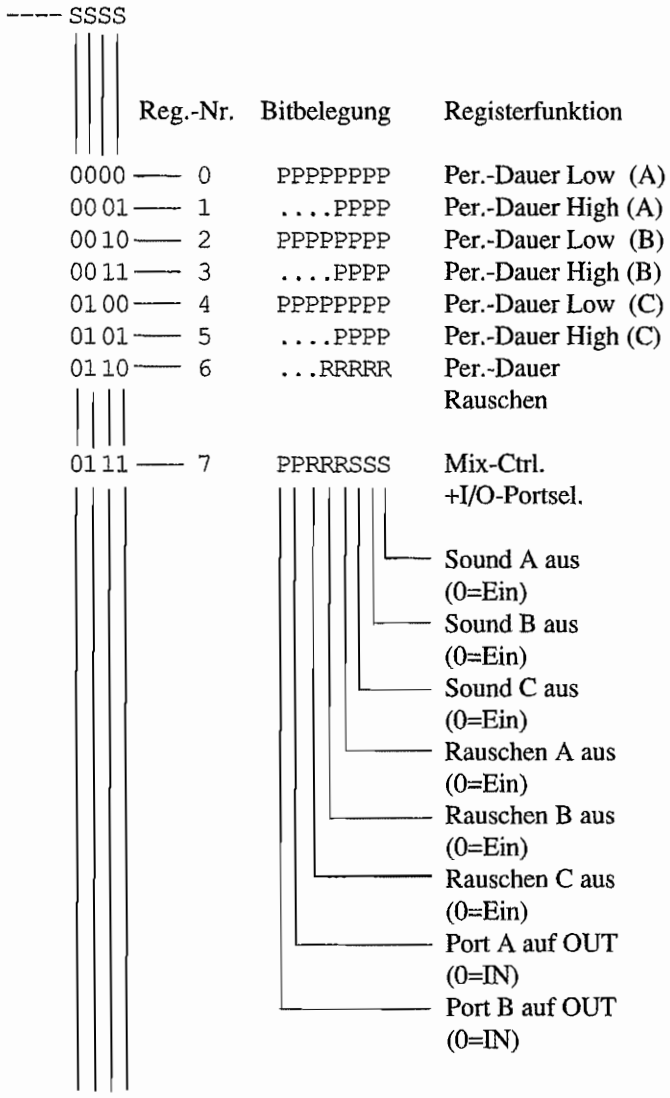
SCSI-Controller 5380 (Nur TT!)

Adresse	Zugriff	Bezeichnung	Label
$FF 8781	R	**Akt. Inhalt des SCSI-Datenbusses**	s_data
oder	W	**Ausgabedaten für SCSI-Datenbus**	s_data
$FF 8783	W	**Initiator-Befehlsregister** CCCCCCCC (Byte) Assert Data to Bus Assert _ATN Assert _SEL Assert _BSY Assert _ACK 0 Test Modus Assert _RST	s_icr
$FF 8783	R	**Initiator-Befehlsregister** CCCCCCCC (Byte) Assert Data to Bus Assert _ATN Assert _SEL Assert _BSY Assert _ACK Lost Arbitration Arbitration in Progress Assert _RST	s_icr
$FF 8785	R/W	**Betriebsartenregister** MMMMMMM (Byte) Arbitrate (=1: On) DMA-Mode (=1: On) Monitor Busy Enable End of Process-Int. Enable Parity Interrupt Enable Parity Check =1: Targetmode =1: Blockmode-DMA	s_mode

Adresse	Zugriff	Bezeichnung	Label
$FF 8787	R/W	**Target Befehlsregister** 0000TTTT (Byte) Assert _I/O Assert _C/D Assert _MSG Assert _REQ	s_tcr
$FF 8789	R	**Busstatusregister** SSSSSSSS (Byte) Zustand Parity-Line Zustand _SEL (1=aktiviert) Zustand _I/O " Zustand _C/D " Zustand _MSG " Zustand _REQ " Zustand _BSY " Zustand _RST "	s_idstat
	W	**ID-Mask f. SELECTION-Phase**	s_idstat
$FF 878B	R	**Statusregister** SSSSSSSS (Byte) Zustand _ACK (1=aktiviert) Zustand _ATN " 1 = Busy Error 1 = Phase match 1 = IRQ aktiv 1 = Parity Error 1 = DMA-Request steht an 1 = End of DMA	s_dmastat
	W	**Start DMA-Ausgabe**	s_dmastat
$FF 878D	R	**Eingabedaten vom SCSI-Bus**	s_targrcv
	W	**Start Target-DMA-Eingabe**	s_targrcv
$FF 878F	R	**Reset Interrupts + Parityfehler**	s_inircv
	W	**Start Initiator-DMA-Eingabe**	s_inircv

Soundchip AY-3-8910

Adresse	Zugriff	Bezeichnung	Label
$FF 8800	R	**Selektiertes Reg. auslesen**	giread
	W	**Chip-Register anwählen**	giselect



Adresse	Zugriff	Bezeichnung			Label
	W	**Chip-Register anwählen**			
	giselect	----SSSS			
			Reg.-Nr.	Bitbelegung	Registerfunktion
		1000	8	---MLLLL	Vol. A / Envelope Ein
		1001	9	---MLLLL	Vol. B / Envelope Ein
		1010	10	---MLLLL	Vol. C / Envelope Ein Level-Ctrl.-Bits(LCB) 1=Env. Generator Ein 0=Volume durch LCB
		1011	11	PPPPPPPP	Per.-Dauer Env. Low
		1100	12	PPPPPPPP	" " High
		1101	13	----CEEE	Hüllkurvenform
		1110	14	DDDDDDDD	I/O Port A (Ausg. Disk Side (1=Side 0) Drv-Sel. A (0=Select) Drv-Sel. B (0=Select) RS 232 RTS-Leitg. RS 232 DTR-Leitg. Strobe Parallelport P.3 Monitor-Buchse (ST) SPEAKER ON /TT CLKDIR/ MEGA STE 0=LAN 1=SERIAL 2 (Nur TT + MEGA STE)
		1111	15	DDDDDDDD	I/O Port B (Parallel-Port für Drucker)

Adresse	Zugriff	Bezeichnung	Label
$FF 8802	W	**Schreiben in angew. Reg.**	giwrite

DMA-Sound-Subsystem

Diese Register existieren im ST nicht!

Adresse	Zugriff	Bezeichnung	Label
$FF 8900	R/W	**Sound-DMA-Control** ---- ---- ---- --RE (WORD) └ DMA-Sound Enable (1=Ein) └ Frame Repeat (1=Repeat)	sndmactl
$FF 8902	R/W	**Frame-Start-Adr. High-Byte** ---- ---- 00HH HHHH	sndbashi
$FF 8904	R/W	**Frame-Start-Adr. Middle-Byte** ---- ---- MMMM MMMM	sndbasmi
$FF 8906	R/W	**Frame-Start-Adr. Low-Byte** ---- ---- LLLL LLLL	sndbaslo
$FF 8908	R/W	**Frame-Adr.-Counter High-Byte** ---- ---- 00HH HHHH	sndadrhi
$FF 890A	R/W	**Frame-Adr.-Counter Middle-Byte** ---- ---- MMMM MMMM	sndadrmi
$FF 890C	R/W	**Frame-Adr.-Counter Low-Byte** ---- ---- LLLL LLLL	sndadrlo
$FF 890E	R/W	**Frame-End-Adresse High-Byte** ---- ---- 00HH HHHH	sndendhi
$FF 8910	R/W	**Frame-End-Adresse Middle-Byte** ---- ---- MMMM MMMM	sndendmi
$FF 8912	R/W	**Frame-End-Adresse Low-Byte** ---- ---- LLLL LLLL	sndendlo
$FF 8920	R/W	**Sound-Mode-Control** ---- ---- M000 00RR (Word) 00 – 6258 Hz Samplerate 01 – 12517 Hz ” 10 – 25033 Hz ” 11 – 50066 Hz ” 0 = Stereo / 1 = Mono	sndmode

Adresse	Zugriff	Bezeichnung	Label
$FF 8922	R/W	**MICROWIRE(TM) Adr.+Data-Bits** DDDD DDDD DDDD DDDD	MWDATA
$FF 8924	R/W	**MICROWIRE(TM) Mask-Register** MMMM MMMM MMMM MMMM	MWMASK

Uhrenchip des TT

Nachfolgend aufgeführte Register existieren nur im TT!

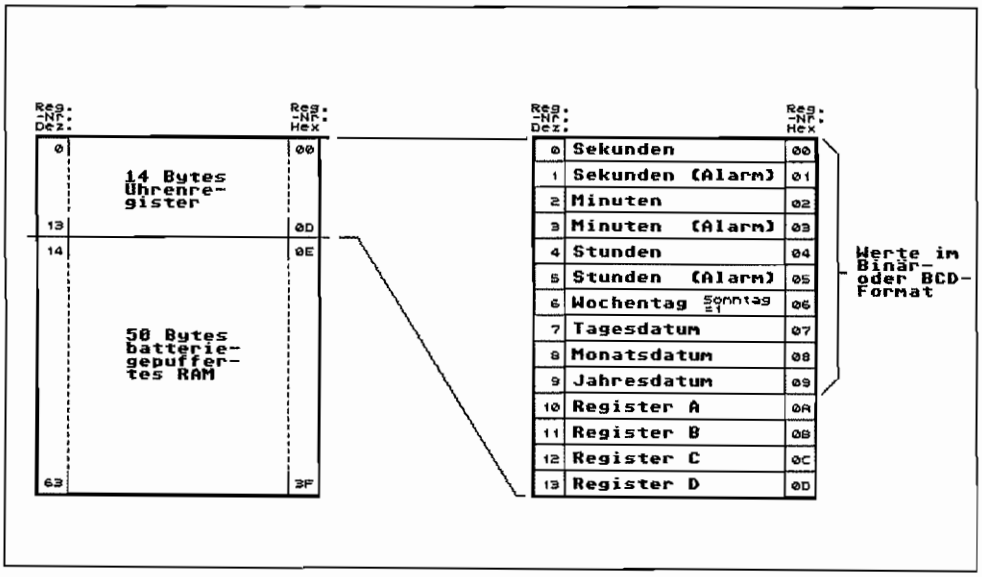


Abb. M.4: Die Register des TT-Uhrenchips

Adresse	Zugriff	Bezeichnung	Label
$FF 8961	R/W	**Registerauswahl im Uhrenchip**	rtc_rnr

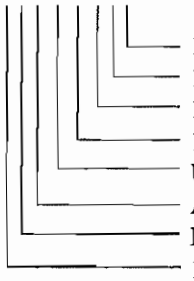
Adresse	Zugriff	Bezeichnung	Label
$FF 8963	R/W	**Daten des selekt. Uhrenregisters**	rtc_data

Reg.
Nr. Bitbeleg. Bedeutung

$A UDDDRRRR

Rate-Select f. Int.-Periode
Divider-Ctrl. (Taktfrequenz!)
1= Update in Progress

$B SPAUQD2Z



1=Sommer-/Winterzeit umschalten
1=24-Stunden-Modus
1=Zeitwerte in Binär, sonst BCD
1=SQW-Ausgang einschalten
Update ended-Int. Enable
Alarm-Int. Enable
Periodic-Int. Enable
1=Uhr stellen (Update gestoppt)

$C IPAU0000

Update ended Int.-Flag
Alarm-Int. Flag
Periodic-Int.-Flag
Int. Request-Flag (1=Int. steht an)

$D V0000000

1= RAM und Zeitwerte sind gültig

Blitter

Nachfolgende Register existieren nicht im TT!

Adresse	Zugriff	Bezeichnung	Label
$FF 8A00	R/W	**Halftone-RAM**, Word 0	Halftone
...			
$FF 8A1E	R/W	**Halftone-RAM**, Word 15	

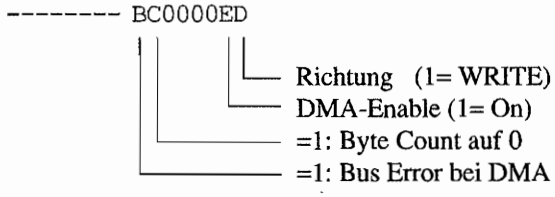
Adresse	Zugriff	Bezeichnung	Label
$FF 8A20	R/W	**Source-X-Inc.-Reg.** (Word)	Src_Xinc
$FF 8A22	R/W	**Source-Y-Inc.-Reg.** (Word)	Src_Yinc
$FF 8A24	R/W	**Source-Address-Reg.** (Long!)	Src_addr
$FF 8A28	R/W	**ENDMASK 1** (Word)	Endmask1
$FF 8A2A	R/W	**ENDMASK 2** (Word)	Endmask2
$FF 8A2C	R/W	**ENDMASK 3** (Word)	Endmask3
$FF 8A2E	R/W	**Dest.-X-Inc.-Reg.** (Word)	Dst_Xinc
$FF 8A30	R/W	**Dest.-Y-Inc.-Reg.** (Word)	Dst_Yinc
$FF 8A32	R/W	**Dest.-Address-Reg.** (Long!)	Dst_Addr
$FF 8A36	R/W	**Worte/Zeile im Bit-Block** (Word)	X_Count
$FF 8A38	R/W	**Zeilen/Bit-Block** (Word)	Y_Count
$FF 8A3A	R/W	**Halftone-OP-Register** (Byte) ------HH || Verknüpfungsergebnis zwischen Halftone- und Source-Daten 00 — Alle Bits ="1" 01 — Nur Halftone 10 — Nur Source 11 — Halftone AND Source	HOP
$FF 8A3B	R/W	**Log. OP-Register** (Byte) ----OOOO |||| (Verknüpfg. zwischen Quell- und Zieldaten) 0000 — Ziel = "0" 0001 — Ziel = Quelle AND Ziel 0010 — Ziel = Quelle AND NOT Ziel 0011 — Ziel = Quelle 0100 — Ziel = NOT Quelle AND Ziel 0101 — Ziel = Ziel 0110 — Ziel = Quelle XOR Ziel	OP

Adresse	Zugriff	Bezeichnung	Label
$FF 8A3B	R/W	**Log. OP-Register** (Byte) ----OOOO |||| 0111 — Ziel = Quelle OR Ziel 1000 — Ziel = NOT Quelle AND NOT Ziel 1001 — Ziel = NOT Quelle XOR Ziel 1010 — Ziel = NOT Ziel 1011 — Ziel = Quelle OR NOT Ziel 1100 — Ziel = NOT Quelle 1101 — Ziel = NOT Quelle OR Ziel 1110 — Ziel = NOT Quelle OR NOT Ziel 1111 — Ziel = "1"	OP
$FF 8A3C	R/W	**Line-Number-Register** (Byte) BHS-LLLL Line-# für Halftone-RAM Smudge (Line-# aus Source Buffer) HOG-Bit (Buszugriff Blitter <-> CPU) 0= Blitter + CPU abwechselnd 1= Blitter allein, CPU gestoppt Busy-Bit (1= Transfer starten)	Line_Num
$FF 8A3D	R/W	**SKEW-Register** (Byte) FN--SSSS Bitversatz zwischen Quell- und Ziel-Block NFSR (No Final Source Read) FXSR (Force Extra Source Read)	Skew

SCC-DMA (Nur TT!)

Adresse	Zugriff	Bezeichnung	Label
$FF 8C01	R/W	**DMA-Address-Pointer** (Highest Byte)	scdmabas

Adresse	Zugriff	Bezeichnung	Label
$FF 8C03	R/W	**DMA-Address-Pointer** (High Byte)	scdmabas+2
$FF 8C05	R/W	**DMA-Address-Pointer** (Low Byte)	scdmabas+4
$FF 8C07	R/W	**DMA-Address-Pointer** (Lowest Byte)	scdmabas+6
$FF 8C09	R/W	**DMA-Bytezähler** (Highest Byte)	scdmacnt
$FF 8C0B	R/W	**DMA-Bytezähler** (High Byte)	scdmacnt+2
$FF 8C0D	R/W	**DMA-Bytezähler** (Low Byte)	scdmacnt+4
$FF 8C0F	R/W	**DMA-Bytezähler** (Lowest Byte)	scdmacnt+6
$FF 8C10	R	**Restdatenregister** (High-Word)	scdmarsd
$FF 8C12	R	**Restdatenregister** (Low-Word)	scdmarsd+2
$FF 8C14	R/W	**Kontrollregister** (Word)	scdmactl



Serial Communications Controller (SCC)

Dieser Baustein existiert nur im TT/MEGA STE!

Adresse	Zugriff	Bezeichnung	Label
$FF 8C81	R/W	**Ctrl.Reg. Kanal A**	sccctl_a
$FF 8C83	R/W	**Data-Reg. Kanal A**	sccdat_a
$FF 8C85	R/W	**Ctrl.Reg. Kanal B**	sccctl_b
$FF 8C87	R/W	**Data-Reg. Kanal B**	sccdat_b

Die *Control-Register* werden auf folgende Weise in zwei Stufen "bedient":

- Der erste Schreibzugriff auf die Control-Register "landet" in Write-Register 0 des entsprechenden SCC-Kanals und wird zur Einstellung der gewünschten Registernummer verwendet.

- Der nächste Zugriff (Schreiben oder Lesen) auf das Control-Register erreicht dann das im ersten Zugriff ausgewählte Register!

Die *Data-Register* erlauben mit einem Schreibzugriff direktes Einschreiben eines Bytes in den Sendebuffer (Write-Register 8). Bei einem Lesezugriff erhält man das Byte aus dem Empfangsbuffer (Read-Register 8).

Also ist ein Zugriff auf die "zeitkritischen Daten" ohne große Umwege möglich.

Näheres zum komplexen Registersatz des SCC findet sich im TT-Hardware-Teil, Kapitel 7, "Seriell, aber schnell – Der SCC macht's möglich".

System Control Unit (SCU)

Dieser Baustein übernimmt im TT/MEGA STE u .a. das Interrupt-Handling. Im ST/STE ist der Baustein nicht vorhanden!

Adresse	Zugriff	Bezeichnung	Label
$FF 8E01	R/W	**System Int. Mask-Register** SSSSSSS- (Byte) System-Soft.-Int. (1=aktiviert) HSYNC-Interrupt (1=aktiviert) Nicht benutzt VSYNC-Interrupt (1=aktiviert) IRQ von SCC (1=aktiviert) IRQ von MFPs (1=aktiviert) _SYSFAIL-Int. VME (1=aktiviert)	sys_mask
$FF 8E03	R	**System Int. Status-Register** Bitbelegung wie vor. Gesetzte Bits stellen Interruptanforderungen dar. (Vor "sys_mask" auslesen, da Lesezugriff auf "sys_mask" IRQs zurücksetzt!)	sys_stat
$FF 8E05	R/W	**System Software Int. erzeugen**	sys_int
$FF 8E07	R/W	**VME-Bus-IRQ Level 3 erzeugen**	vme_int
$FF 8E09	R/W	**SCU General Purpose Reg. 1**	scu_gp1
$FF 8E0B	R/W	**SCU General Purpose Reg. 2**	scu_gp2

Adresse	Zugriff	Bezeichnung	Label
$FF 8E0D	R/W	**VME-Bus Int. Mask-Register** (Byte) VVVVVVV- _IRQ1 v. VME-Bus (1=aktiviert) _IRQ2 v. VME-Bus (1=aktiviert) _IRQ3 / Soft-Int. (1=aktiviert) _IRQ4 v. VME-Bus (1=aktiviert) _IRQ5 / SCC-Int. (1=aktiviert) _IRQ6 / MFP-Int. (1=aktiviert) _IRQ7 v. VME-Bus (1=aktiviert)	vme_mask
$FF 8E0F	R	**VME-Bus Int. Status-Register** Bitbelegung wie vor. Gesetzte Bits stellen Interrupt-anforderungen dar. (Vor "vme_mask" auslesen, da Lese-zugriff auf "vme_mask" IRQs zurücksetzt!)	vme_stat
$FF 8E21	R/W	**MEGA STE Cache + Taktctrl.** (Byte) Die unteren beiden Bits kontrollieren die Taktfrequenz und den Cache-Speicher.	ste_ctl

Joystickports beim STE

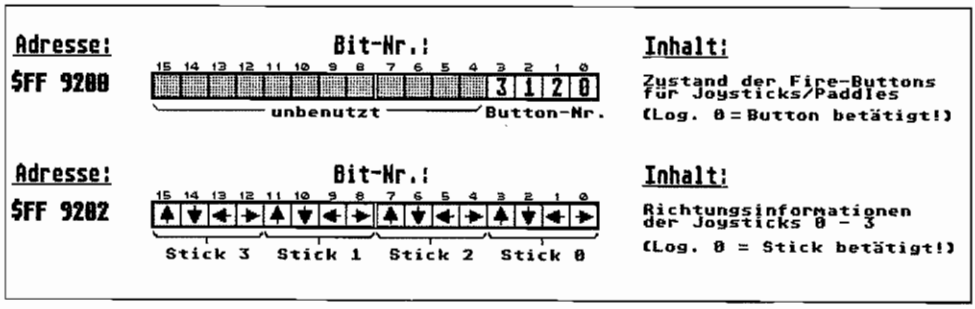


Abb. M.5: Die Adressen der neuen Joystickports

Adresse	Label	Inhalt
$FF 9220	XPEN	X-Position des Lightpens
$FF 9222	YPEN	Y-Position des Lightpens

Adresse	Label	Inhalt
$FF 9210	PADDL0	Position des Paddle 0
$FF 9212	PADDL1	Position des Paddle 1
$FF 9214	PADDL2	Position des Paddle 2
$FF 9216	PADDL3	Position des Paddle 3

Multifunktionsbaustein MFP MC68901

Dieser Chip ist in allen Maschinen vorhanden, jedoch zum Teil geringe Abweichungen bei der Belegung der GPIP-Anschlüsse!

Alle Register sind mit Bytezugriffen zu erreichen.

Adresse	Zugriff	Bezeichnung	Label
$FF FA01	R/W	**GPIP-Data-Register** IIIIIIII (Meldeleitungen, interruptfähig) BUSY von par. Port (Drucker) RS 232 Data Carrier Detect (DCD) RS 232 Clear To Send (CTS) GPU Done (nicht TT!) CLKDIR (nur TT!) IRQ von Tastatur-/ MIDI-ACIA INT von FDC/ACSI-Bus RS 232 Ring Indicator (RI) Monochrome Monitor Detect XORED m. Frame-Ende-Signal (STE/TT)	gpip
$FF FA03	R/W	**Active-Edge-Register** Bitbelegung wie bei "gpip"! Interruptauslösung bei 1=steig., 0=fall. Flanke	aer
$FF FA05	R/W	**Data-Direction-Register** Bitbelegung wie bei "gpip"! Signalrichtung für GPIP-Port 1 = Out, 0 = In	ddr

Adresse	Zugriff	Bezeichnung	Label
$FF FA07	R/W	**Interrupt-Enable-Register A** EEEEEEEE Timer B (Display enable Signal) XMIT Error (RS 232) XMIT Buffer empty (RS 232) RCV Error (RS 232) RCV Buffer full (RS 232) Timer A (BUSY-Signal bei ST) Timer A (Frame-Ende bei TT/STE) Port I6 (RS 232 Ring Indicator) Port I7 (Monochr.-Monitor Detect)	iera
$FF FA09	R/W	**Interrupt-Enable-Register B** EEEEEEEE Port I0 (BUSY v. par. Schnittst.) Port I1 (RS 232 Data Carrier Detect) Port I2 (RS 232 Clear To Send) Port I3 (GPU done, nicht TT!) Timer D (Baudraten-Gen. RS 232) Timer C (200 Hz Systemtakt) Port I4 (IRQ Tastatur-/ MIDI-ACIA) Port I5 (IRQ von FDC/ACSI-Bus)	ierb
$FF FA0B	R/W	**Interrupt-Pending-Register A** (Bitbelegung siehe "iera")	ipra
$FF FA0D	R/W	**Interrupt-Pending-Register B** (Bitbelegung siehe "ierb")	iprb
$FF FA0F	R/W	**Interrupt-In-Service-Register A** (Bitbelegung siehe "iera")	isra
$FF FA11	R/W	**Interrupt-In-Service-Register B** (Bitbelegung siehe "ierb")	isrb
$FF FA13	R/W	**Interrupt-Mask-Register A** (Bitbelegung siehe "iera")	imra

Adresse	Zugriff	Bezeichnung	Label
$FF FA15	R/W	**Interrupt-Mask-Register B** (Bitbelegung siehe "ierb")	imrb
$FF FA17	R/W	**Interrupt-Vektor-Register** VVVVS--- End-of-Interrupt-Modus High-Nibble der Vektornummer	vr
$FF FA19	R/W	**Timer-A-Ctl.-Reg.** Bei ST wird Timer A-Eingang vom BUSY-Anschluß angesteuert. Bei TT/STE wird Timer A-Eingang mit Frame-Ende-Signal des DMA-Sound- Moduls angesteuert. Bitbelegung wie bei Timer-B-Ctl.-Reg.!	tacr
$FF FA1B	R/W	**Timer-B-Ctl.-Reg.** ---RCCCC (Display Enable-Signal) Timer Ausgang auf Low	tbcr

Modus	*Vorteiler*
0000 Stop	–
0001 Delay	4
0010 Delay	10
0011 Delay	16
0100 Delay	50
0101 Delay	64
0110 Delay	100
0111 Delay	200
1000 Ereignis	–
1001 Pulsbreite	4
1010 Pulsbreite	10
1011 Pulsbreite	16
1100 Pulsbreite	50
1101 Pulsbreite	64
1110 Pulsbreite	100
1111 Pulsbreite	200

Adresse	Zugriff	Bezeichnung	Label
$FF FA1D	R/W	**Timer C+D Control Reg.**	tcdcr

```
-CCC-DDD
```

(C: 200Hz-System Takt)
(D: Baudrate für RS232 des MFP)

	Modus	*Vorteiler*	
000-000	STOP	–	(Takt=2,4576 MHz)
001-001	Delay	4	
010-010	Delay	10	
011-011	Delay	16	
100-100	Delay	50	
101-101	Delay	64	
110-110	Delay	100	
111-111	Delay	200	

Timer: (C) (D)

$FF FA1F	R/W	**Timer A Data Register**	tadr
$FF FA21	R/W	**Timer B Data Register**	tbdr
$FF FA23	R/W	**Timer C Data Register**	tcdr
$FF FA25	R/W	**Timer D Data Register**	tddr
$FF FA27	R/W	**Sync.-Character-Register**	scr
$FF FA29	R/W	**USART-Control-Register**	ucr

```
CCCCCCC-
```

1=Even-, 0=Odd-Parity
Parity Enable

	Start	*Stop*	*Modus*
00	0	0	Synchron
01	1	1	Asynchron
10	1	1,5	Asynchron
11	1	2	Asynchron

Wortlänge in Bits	
00	8
01	7
10	6
11	5

Clock Mode (1=Clk/16, 0=Clk)

Adresse	Zugriff	Bezeichnung	Label
$FF FA2B	R/W	**Receiver-Status-Register** SSSSSSSS Receiver Enable Synchronous Strip Enable Match bzw. Character in progress Found/Search bzw. Break Frame Error Parity Error Overrun Error Buffer Full	rsr
$FF FA2D	R/W	**Transmitter-Status-Register** SSSSSSSS Transmitter Enable Sender-Ausgangsleitung 00 — Hochohmig	tsr
$FF FA2D	R/W	**Transmitter-Status-Register** SSSSSSSS Sender-Ausgangsleitung 01 — Low-Pegel 10 — High-Pegel 11 — Schleife Break senden Senden beendet Auto Turnaround Underrun Error Sendepuffer leer	tsr
$FF FA2F	R/W	**USART-Data-Register** (Lesen holt Byte aus Empfangspuffer) (Schreiben bringt Byte in Sendepuffer)	udr

Floating Point Processor MC68881 (SFP004)

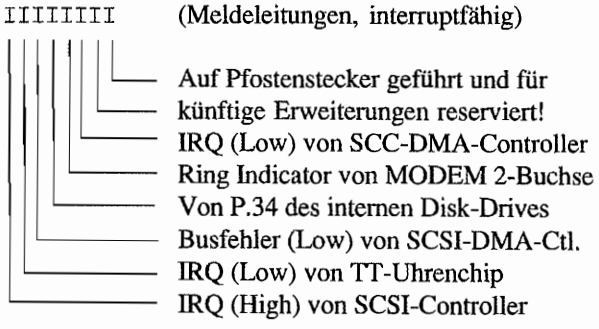
Die Register des Floating-Point-Processor-Adapters (SFP004) sind im MEGA-ST(E) ab Adresse $FF FA40 zu finden! Nähere Informationen zu diesen Hardware-Registern finden Sie im Anhang im Kapitel "Der Coprozessor MC68881 im MEGA ST(E)".

Adresse	Zugriff	Bezeichnung		Label
$FF FA40	R	**Response-Register**	(Word)	FPstat
$FF FA42	W	**Control-Register**	(Word)	FPctl
$FF FA44	R	**Save-Register**	(Word)	FPsave
$FF FA46	R/W	**Restore-Register**	(Word)	FPrestor
$FF FA48	?	(Reserviert!)		
$FF FA4A	W	**Command-Register**	(Word)	FPcmd
$FF FA4C	?	(Reserviert!)		
$FF FA4E	R	**Condition-Code-Reg.**	(Word)	FPccr
$FF FA50	R/W	**Operanden-Register**	(Long!)	FPop
$FF FA54	R	**Register Select**	(Word)	FPselct
$FF FA56	?	(Reserviert!)		
$FF FA58	W	**Instruction Address**	(Long!)	FPiadr
$FF FA5C	?	(Reserviert!)	(Long!)	

TT-MFP MC68901

Dieser Chip ist nur im TT zu finden! Alle Register sind mit Bytezugriffen zu erreichen.

Adresse	Zugriff	Bezeichnung	Label
$FF FA81	R/W	**GPIP-Data-Register**	GPIP_TT



Adresse	Zugriff	Bezeichnung	Label
$FF FA83	R/W	**Active-Edge-Register** Bitbelegung wie bei "GPIP_TT"! Interruptauslösung bei 1 = steig., 0 = fall. Flanke	AER_TT
$FF FA85	R/W	**Data-Direction-Register** Bitbelegung wie bei "GPIP_TT"! Signalrichtung für GPIP-Port 1 = Out, 0 = In	DDR_TT
$FF FA87	R/W	**Interrupt-Enable-Register A** EEEEEEEE Timer B (Display enable Signal) XMIT Error (RS 232) XMIT Buffer empty (RS 232) RCV Error (RS 232) RCV Buffer full (RS 232) Timer A Port I6 (IRQ von Uhrenchip) Port I7 (IRQ von SCSI-Controller)	IERA_TT
$FF FA89	R/W	**Interrupt-Enable-Register B** EEEEEEEE Port I0 Port I1 Port I2 (IRQ v. SCC-DMA-Controller) Port I3 (RI von MODEM 2-Buchse) Timer D (Baudr.-Gen. für SERIAL 1) Timer C (Takt für SCC-Kanal B) Port I4 (P.34 von int. Floppy-Disk) Port I5 (Busfehler SCSI-DMA-Chip)	IERB_TT
$FF FA8B	R/W	**Interrupt-Pending-Register A** (Bitbelegung siehe "IERA_TT")	IPRA_TT
$FF FA8D	R/W	**Interrupt-Pending-Register B** (Bitbelegung siehe "IERB_TT")	IPRB_TT

Adresse	Zugriff	Bezeichnung	Label
$FF FA8F	R/W	**Interrupt-In-Service-Register A** (Bitbelegung siehe "IERA_TT")	ISRA_TT
$FF FA91	R/W	**Interrupt-In-Service-Register B** (Bitbelegung siehe "IERB_TT")	ISRB_TT
$FF FA93	R/W	**Interrupt-Mask-Register A** (Bitbelegung siehe "IERA_TT")	IMRA_TT
$FF FA95	R/W	**Interrupt-Mask-Register B** (Bitbelegung siehe "IERB_TT")	IMRB_TT
$FF FA97	R/W	**Interrupt-Vektor-Register** VVVVS--- S — End-of-Interrupt-Modus V — High-Nibble der Vektornummer	VR_TT
$FF FA99	R/W	**Timer-A-Ctl.-Reg.** Bitbelegung wie bei Timer-B-Ctl.-Reg.!	TACR_TT
$FF FA9B	R/W	**Timer-B-Ctl.-Reg.** ---RCCCC (Display Enable-Signal) C — *Modus* / *Vorteiler*: 0000 — Stop – 0001 — Delay 4 0010 — Delay 10 0011 — Delay 16 0100 — Delay 50 0101 — Delay 64 0110 — Delay 100 0111 — Delay 200 1000 — Ereignis – 1001 — Pulsbreite 4 1010 — Pulsbreite 10 1011 — Pulsbreite 16 1100 — Pulsbreite 50 1101 — Pulsbreite 64 1110 — Pulsbreite 100 1111 — Pulsbreite 200 R — Timer-Ausgang auf Low	TBCR_TT

Adresse	Zugriff	Bezeichnung	Label
$FF FA9D	R/W	**Timer C+D Control Reg.**	TCDCR_TT

```
-CCC-DDD
 |||  |||
 |||  |||   (C: Takt für SCC-Kanal B)
 |||  |||   (D: Baudrate für SERIAL 1)
```

	Modus	*Vorteiler*	
000-000	STOP	–	(Takt=2,4576 MHz)
001-001	Delay	4	
010-010	Delay	10	
011-011	Delay	16	
100-100	Delay	50	
101-101	Delay	64	
110-110	Delay	100	
111-111	Delay	200	

Timer: (C) (D)

$FF FA9F	R/W	**Timer A Data Register**	TADR_TT
$FF FAA1	R/W	**Timer B Data Register**	TBDR_TT
$FF FAA3	R/W	**Timer C Data Register**	TCDR_TT
$FF FAA5	R/W	**Timer D Data Register**	TDDR_TT
$FF FAA7	R/W	**Sync.-Character-Register**	SCR_TT
$FF FAA9	R/W	**USART-Control-Register**	UCR_TT

```
CCCCCCC-
|||||  |
|||||  +---- 1=Even-, 0=Odd-Parity
|||||  +---- Parity Enable
|||||
|||||        Start Stop  Modus
|||00 ------  0     0     Synchron
|||01 ------  1     1     Asynchron
|||10 ------  1     1,5   Asynchron
|||11 ------  1     2     Asynchron
|||
|||          Wortlänge in Bits
||00 ------      8
||01 ------      7
||10 ------      6
||11 ------      5
|+------------ Clock Mode (1=Clk/16, 0=Clk)
```

Adresse	Zugriff	Bezeichnung	Label
$FF FAAB	R/W	**Receiver-Status-Register** SSSSSSSS Receiver Enable Synchronous Strip Enable Match bzw. Character in progress Found/Search bzw. Break Frame Error Parity Error Overrun Error Buffer Full	RSR_TT
$FF FAAD	R/W	**Transmitter-Status-Register** SSSSSSSS Transmitter Enable Sender-Ausgangsleitung 00 Hochohmig 01 Low-Pegel 10 High-Pegel 11 Schleife Break senden Senden beendet Auto Turnaround Underrun Error Sendepuffer leer	TSR_TT
$FF FAAF	R/W	**USART-Data-Register** (Lesen holt Byte aus Empfangspuffer) (Schreiben bringt Byte in Sendepuffer)	UDR_TT

Tastatur-ACIA

Adresse	Zugriff	Bezeichnung	Label
$FF FC00	R	**ACIA-Statusregister** (Byte) SSSSSSSS Receive Data Register Full Transmit Data Register Empty Data Carrier Detect Clear To Send Framing Error Receiver Overrun Parity Error Interrupt Request	keyctl
$FF FC01	W	**ACIA-Steuerregister** (Byte) CCCCCCCC (ACIA-Betriebsart/Vorteilerfaktor) 00 — Baudrate=Taktfrequenz 01 — Baudrate=Taktfrequenz/16 10 — Baudrate=Taktfrequenz/64 11 — ACIA Reset *Bits/Wort* *Parity* *Stopbits* 000 — 7 gerade 2 001 — 7 ungerade 2 010 — 7 gerade 1 011 — 7 ungerade 1 100 — 8 – 2 101 — 8 – 1 110 — 8 gerade 1 111 — 8 ungerade 1 00 — RTS auf Low, TxINT gesperrt 01 — RTS auf Low, TxINT freigeben 10 — RTS auf High, TxINT gesperrt 11 — RTS auf Low, TxINT gesperrt “Break” senden RxINT freigeben	keyctl

Adresse	Zugriff	Bezeichnung	Label
$FF FC02	R/W	**ACIA-DATA-Register** (Byte) (READ = Empfangsdaten) (WRITE= Sendedaten)	keybd

MIDI-ACIA

Adresse	Zugriff	Bezeichnung	Label
$FF FC04	R	**ACIA-Statusregister** (Byte) (siehe Tastatur-ACIA)	midictl
	W	**ACIA Steuerregister** (Byte) (siehe Tastatur-ACIA)	midictl
$FF FC06	R/W	**ACIA DATA Register** (Byte) (READ = Empfangsdaten) (WRITE= Sendedaten)	midi

Anhang N: Pinbelegungen der Chips

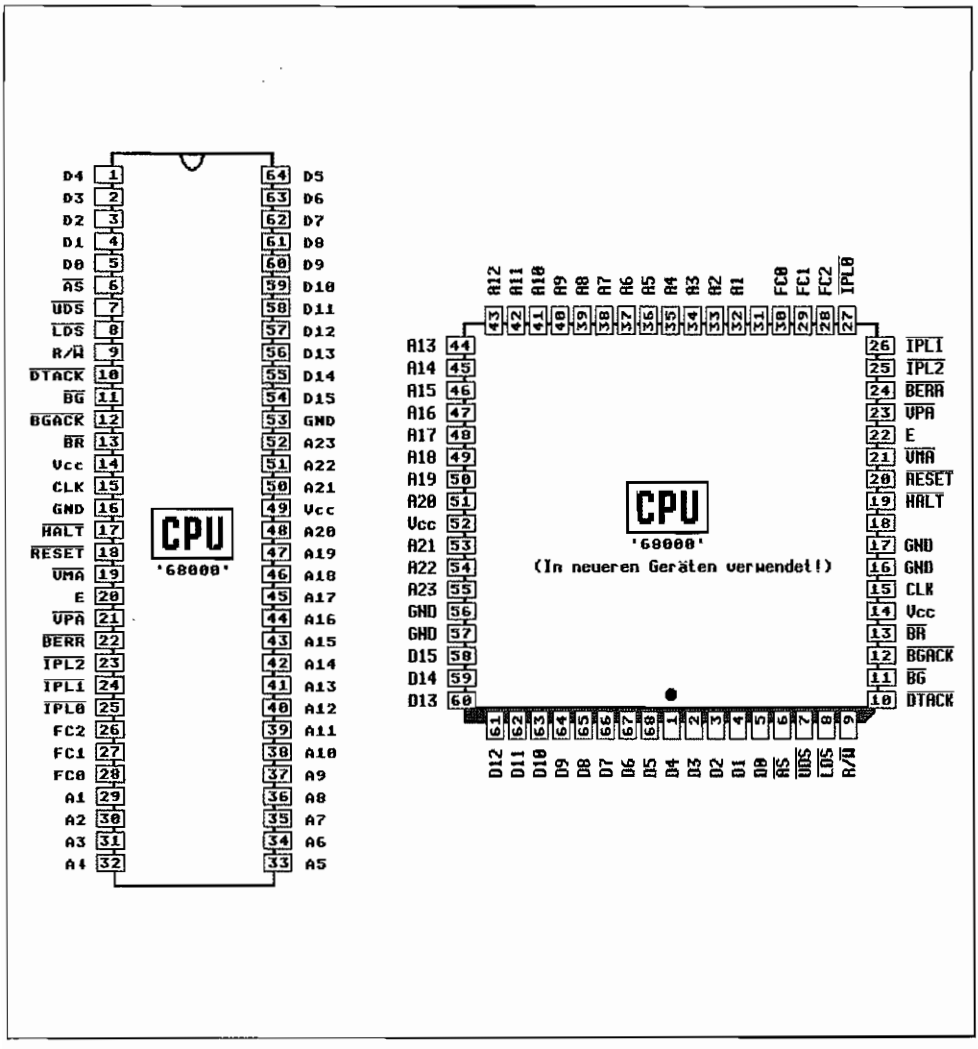


Abb. N.1: CPU-Pinbelegungen im ST/STE

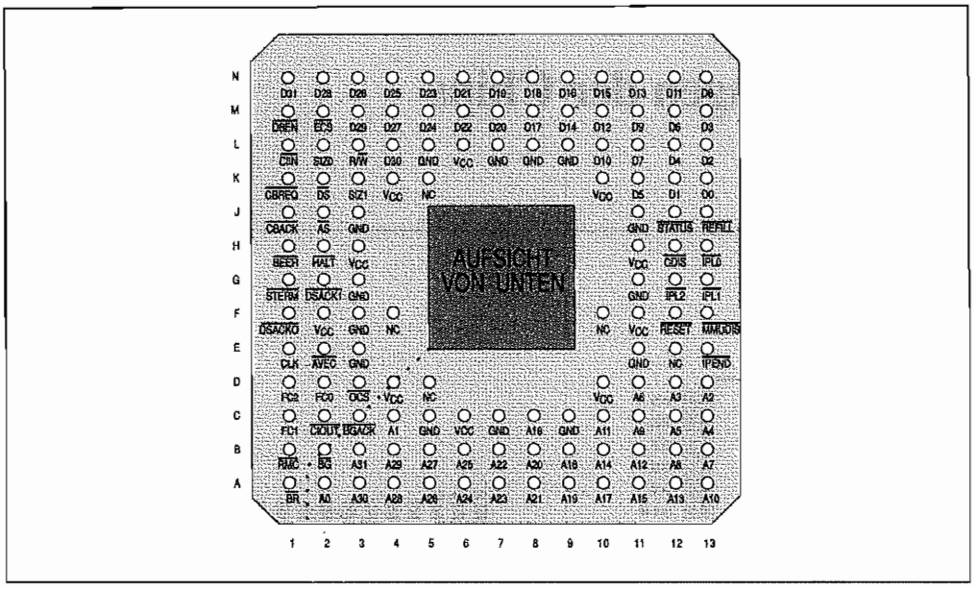


Abb. N.2: CPU MC 68030 im TT (PIN-GRID-ARRAY-Format)

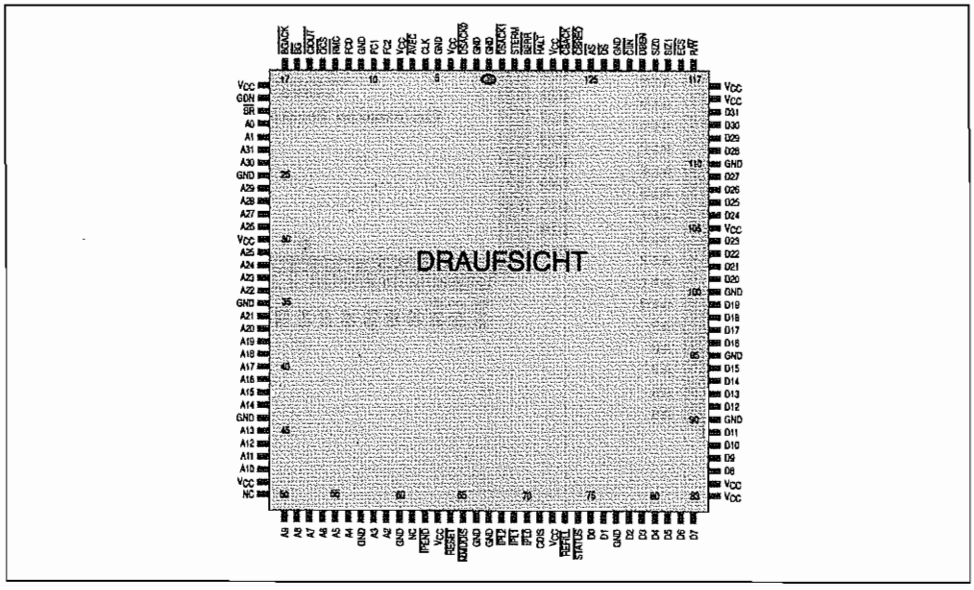


Abb. N.3: CPU MC 68030 im TT (SMD-Format)

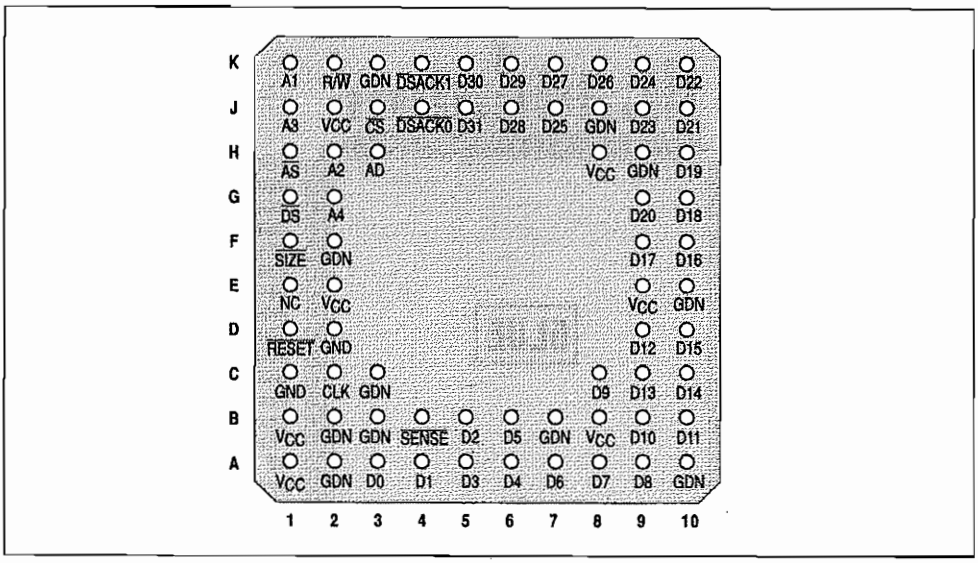


Abb. N.4: Die Pinbelegung des 68881/68882 im PIN-GRID-ARRAY-Format

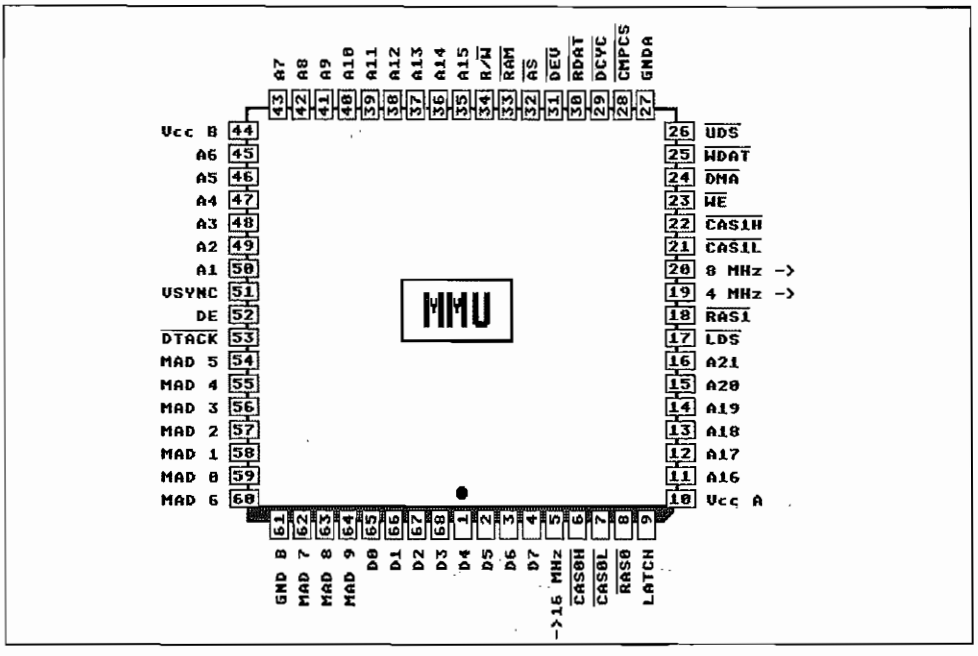


Abb. N.5: Pinbelegung der ST-MMU

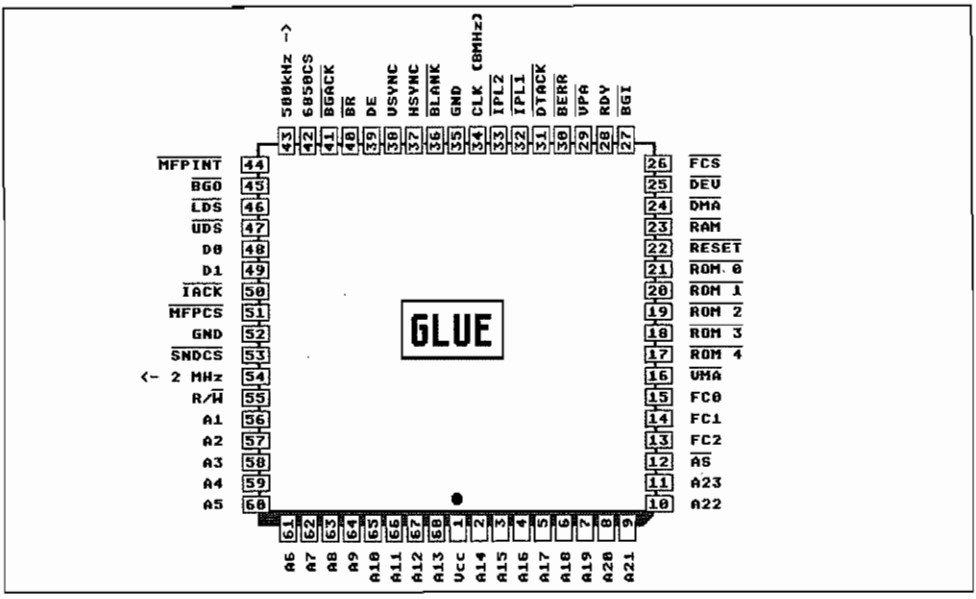


Abb. N.6: Pinbelegung des ST-GLUE

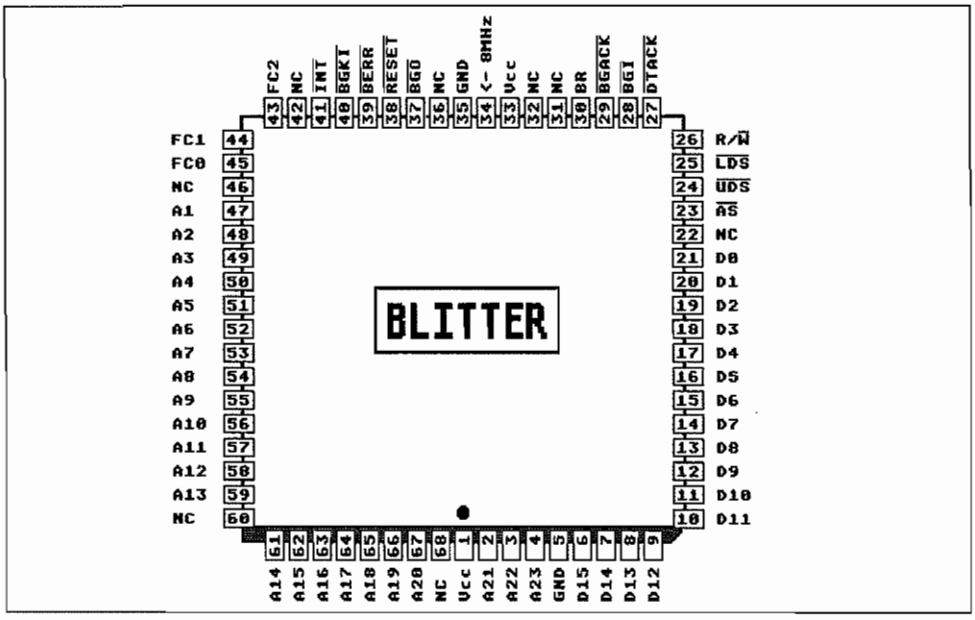


Abb. N.7: Pinbelegung des ST/STE-BLITTER

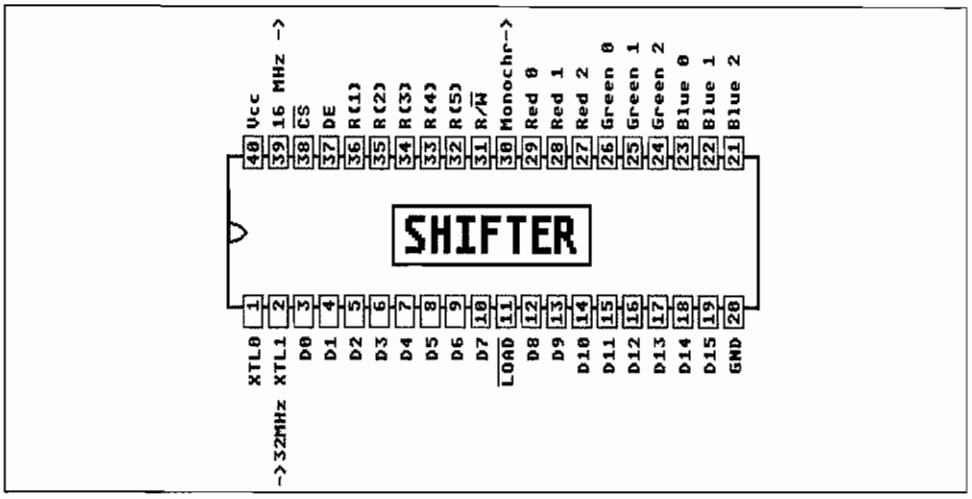


Abb. N.8: Pinbelegung des ST-SHIFTER

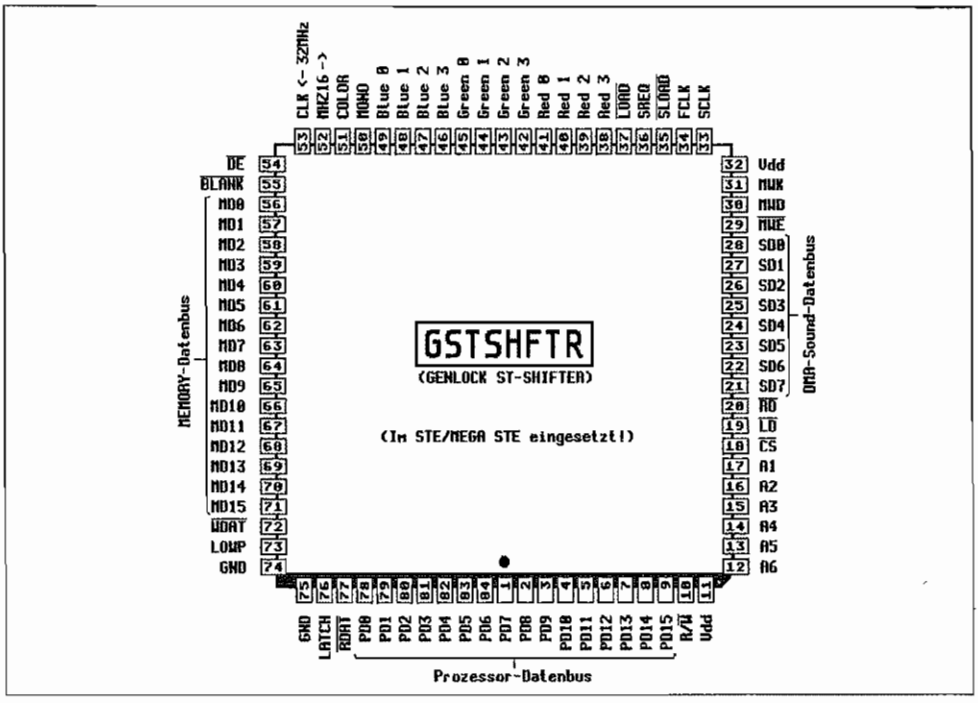


Abb. N.9: Pinbelegung des STE-SHIFTER

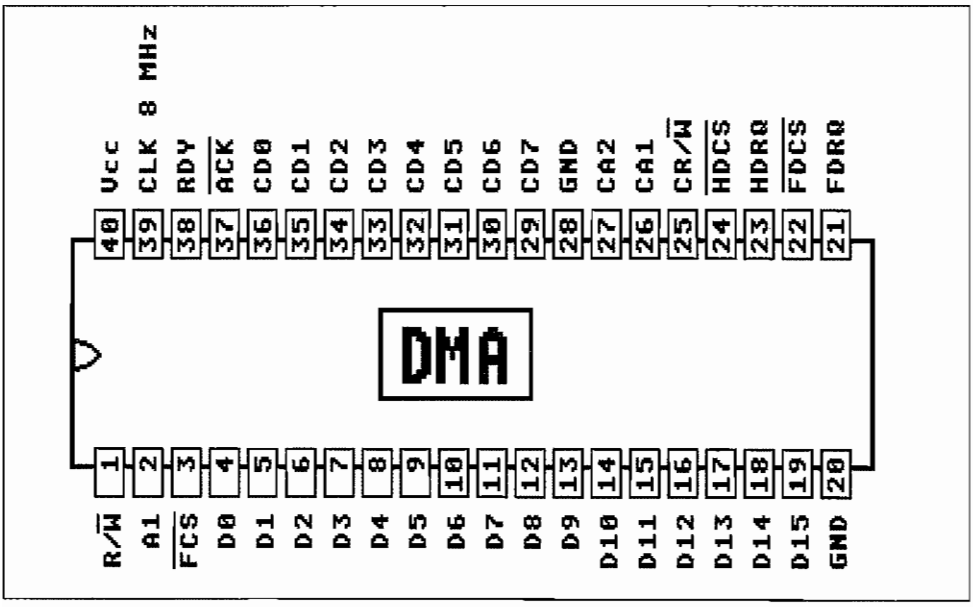


Abb. N.10: Pinbelegung des ST/STE/TT-DMA-Chips

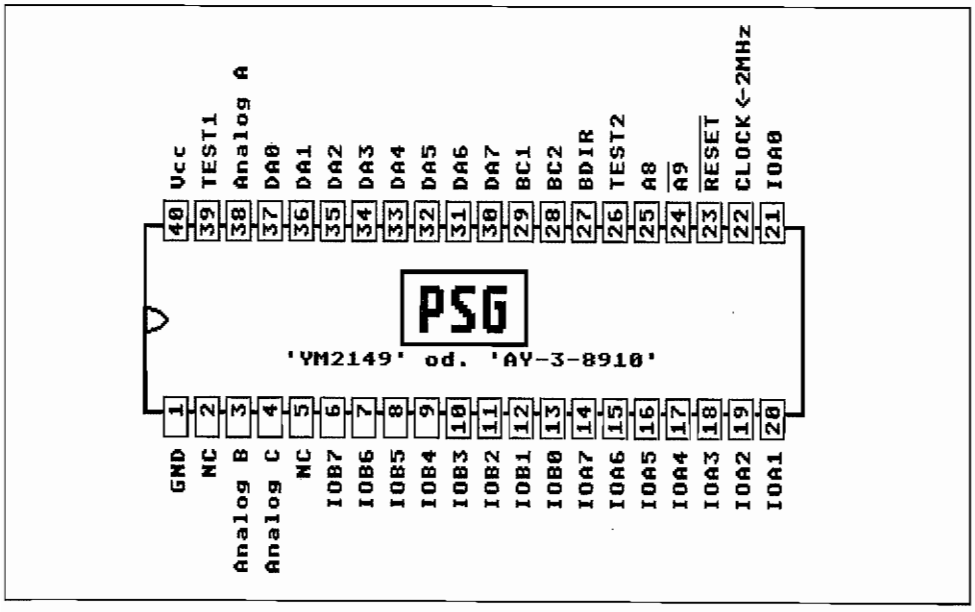


Abb. N.11: Pinbelegung des ST/STE/TT-PSG

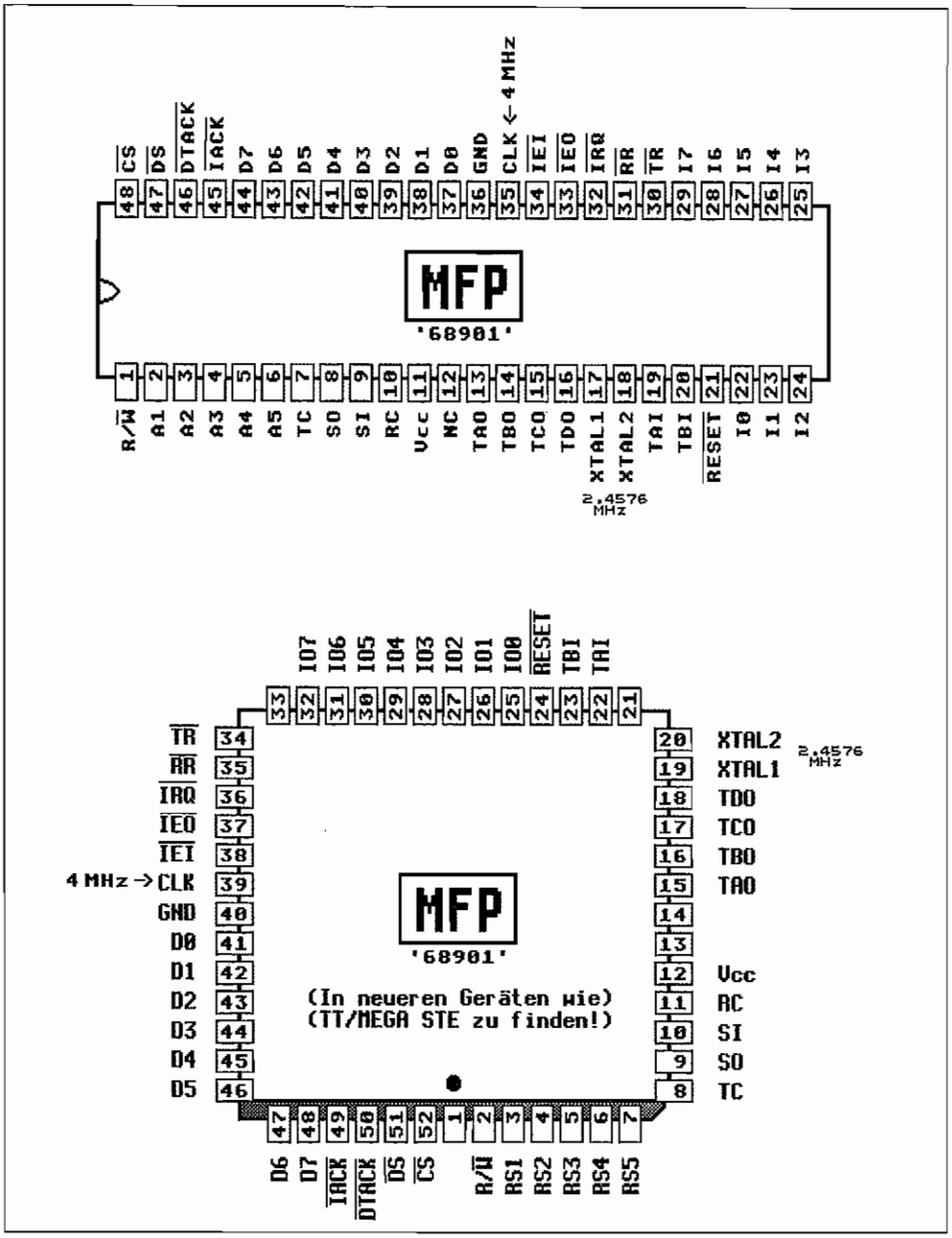


Abb. N.12: Pinbelegung des ST-MFP und des STE/TT-MFP

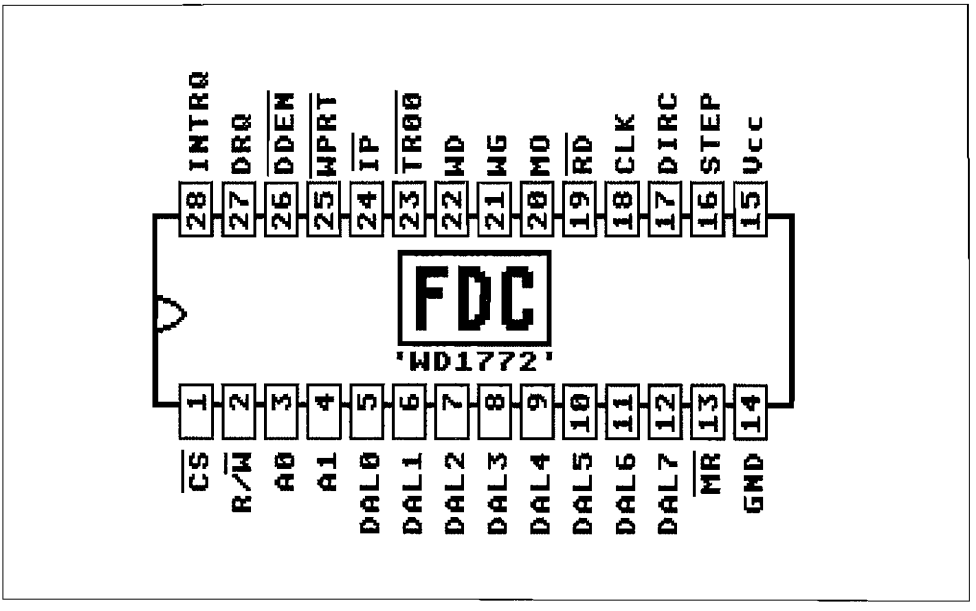


Abb. N.13: Pinbelegung des ST/STE/TT-FDC

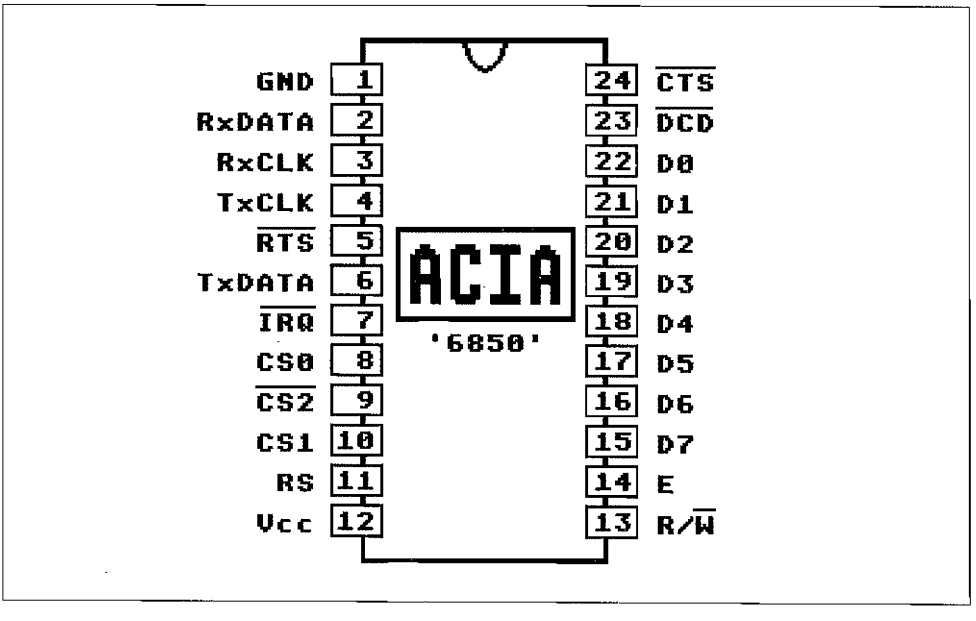


Abb. N.14: Pinbelegung des ST/STE/TT-ACIA

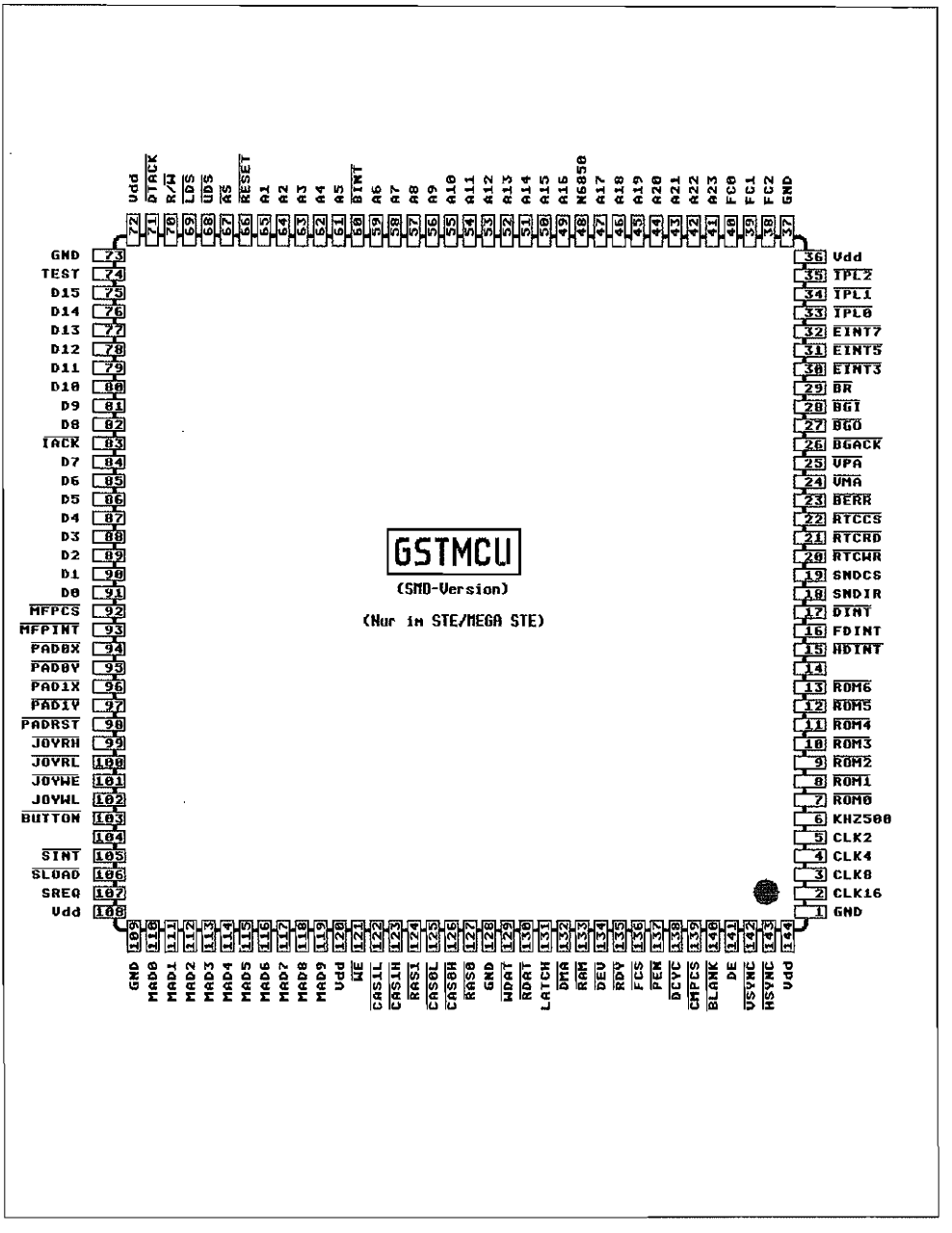


Abb. N.15: Pinbelegung der STE/GSTMCU

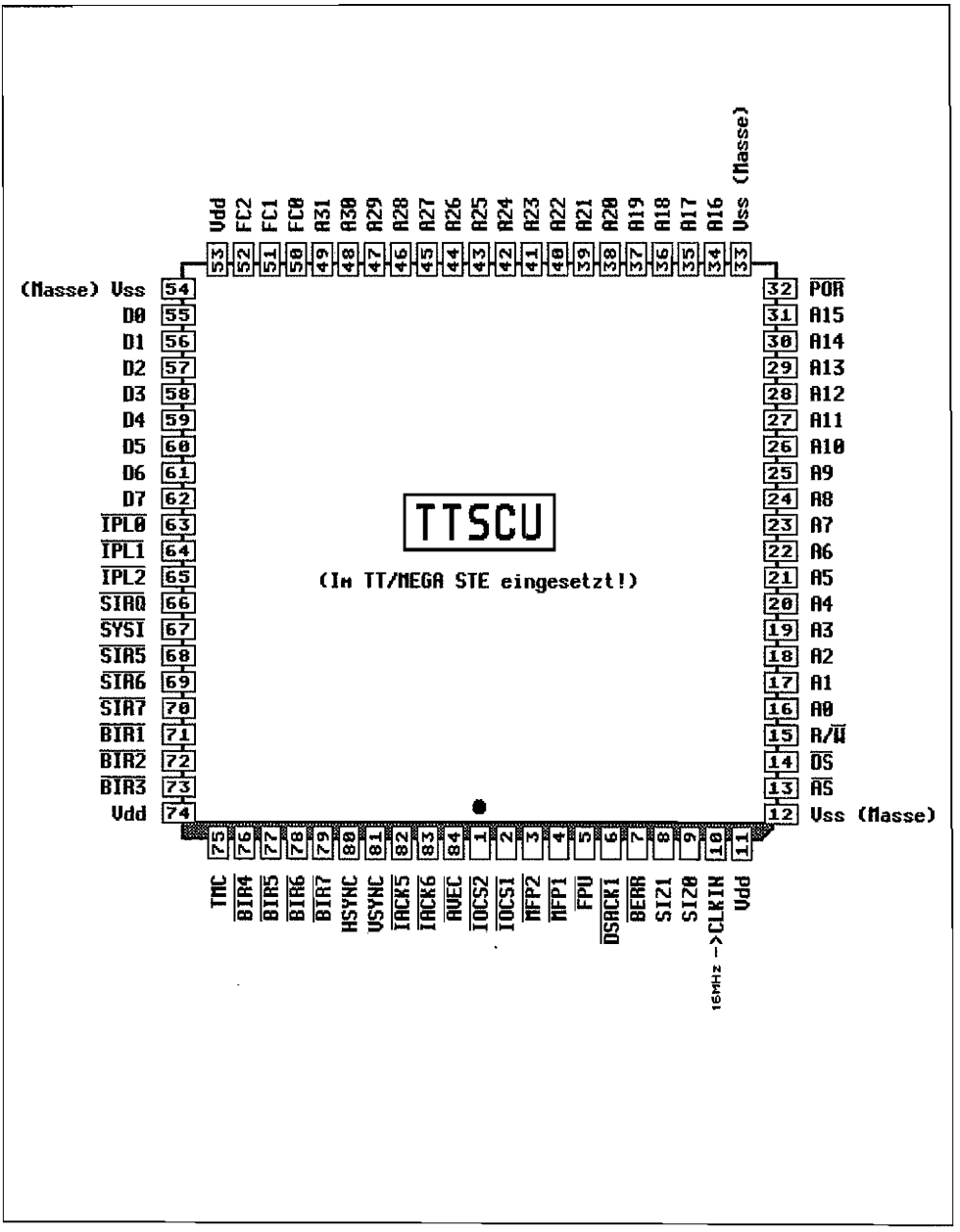


Abb. N.16: Pinbelegung der STE/TT-TTSCU

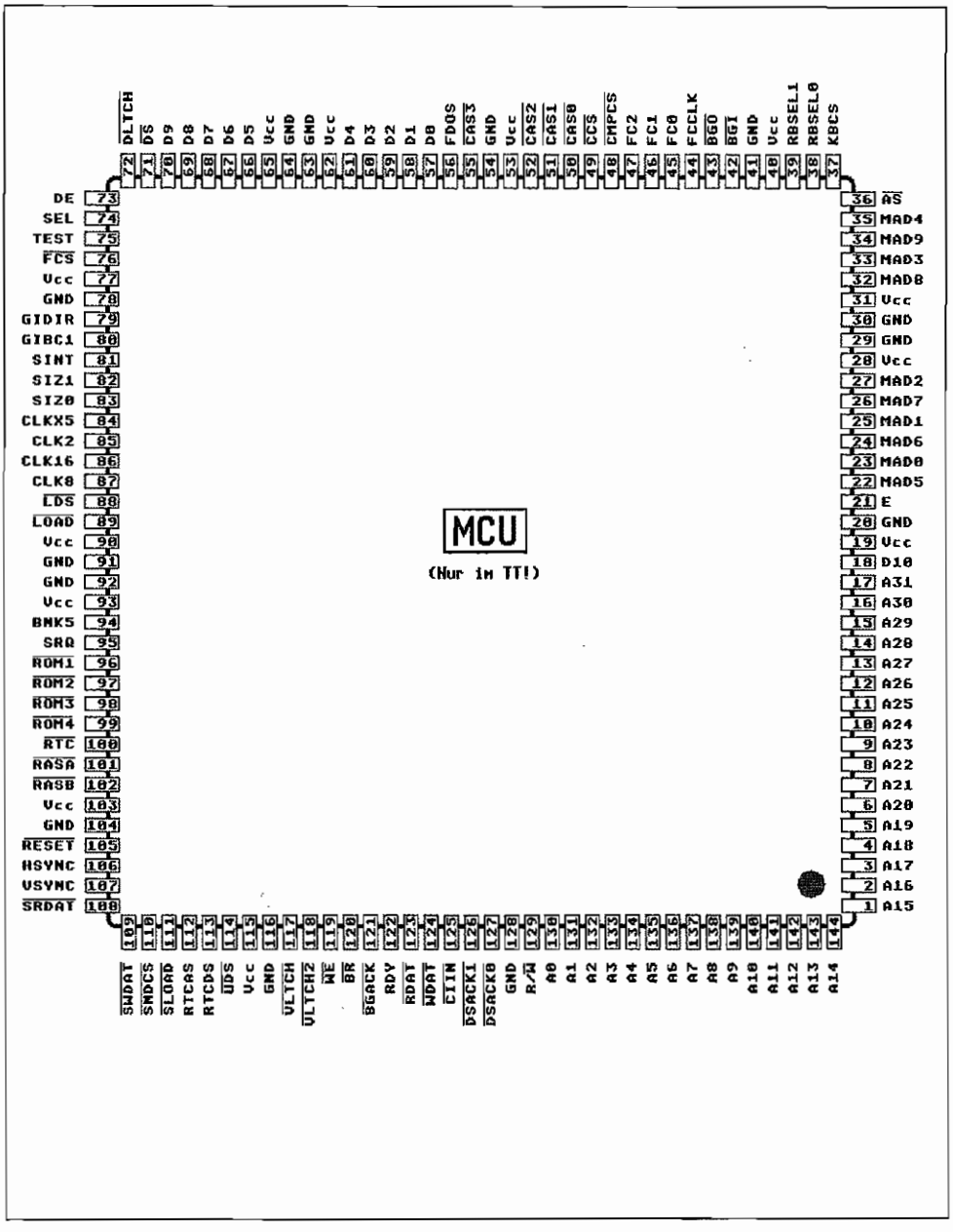


Abb. N.17: Pinbelegung der TT-MCU

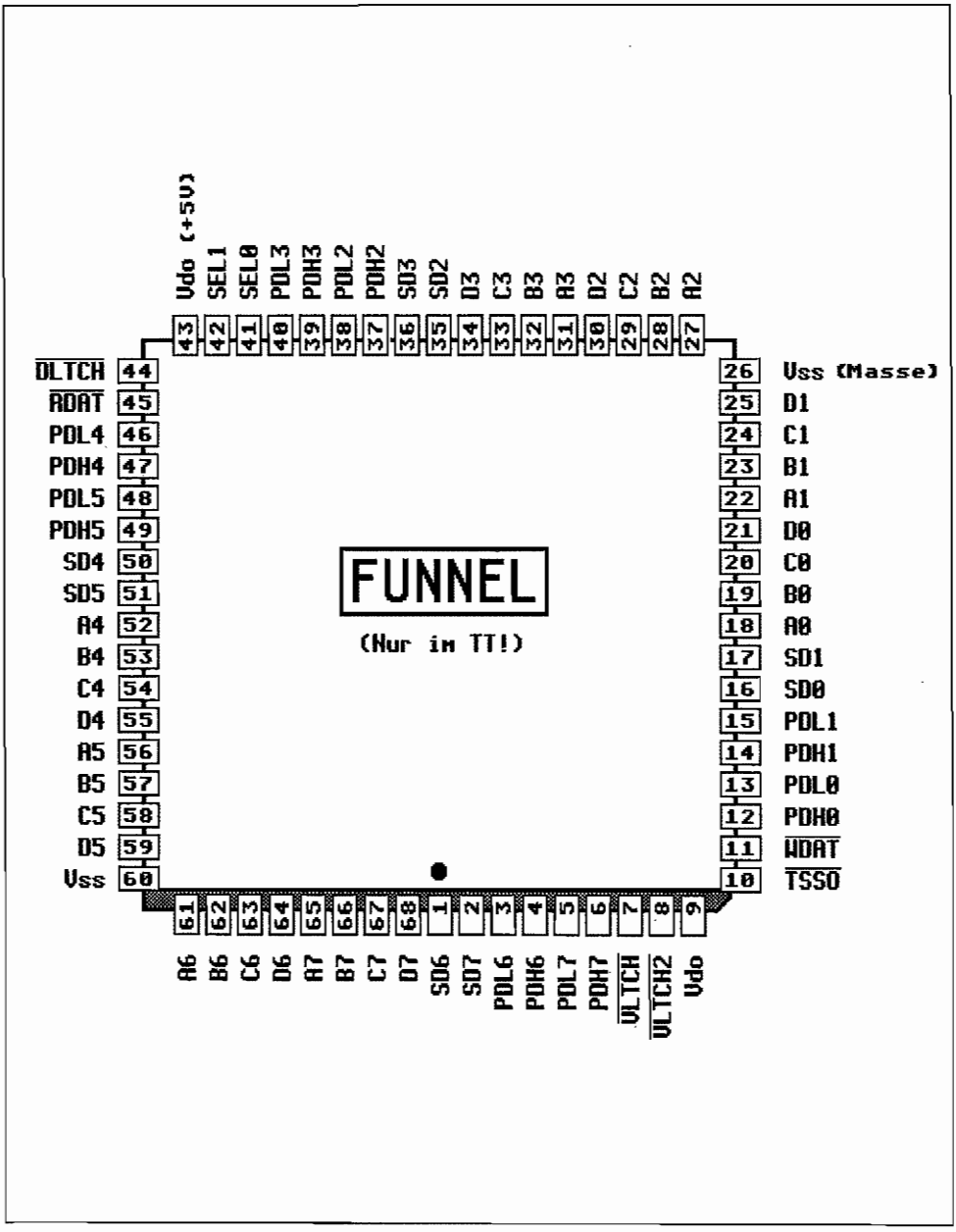


Abb. N.18: Pinbelegung des TT/FUNNEL-Chips

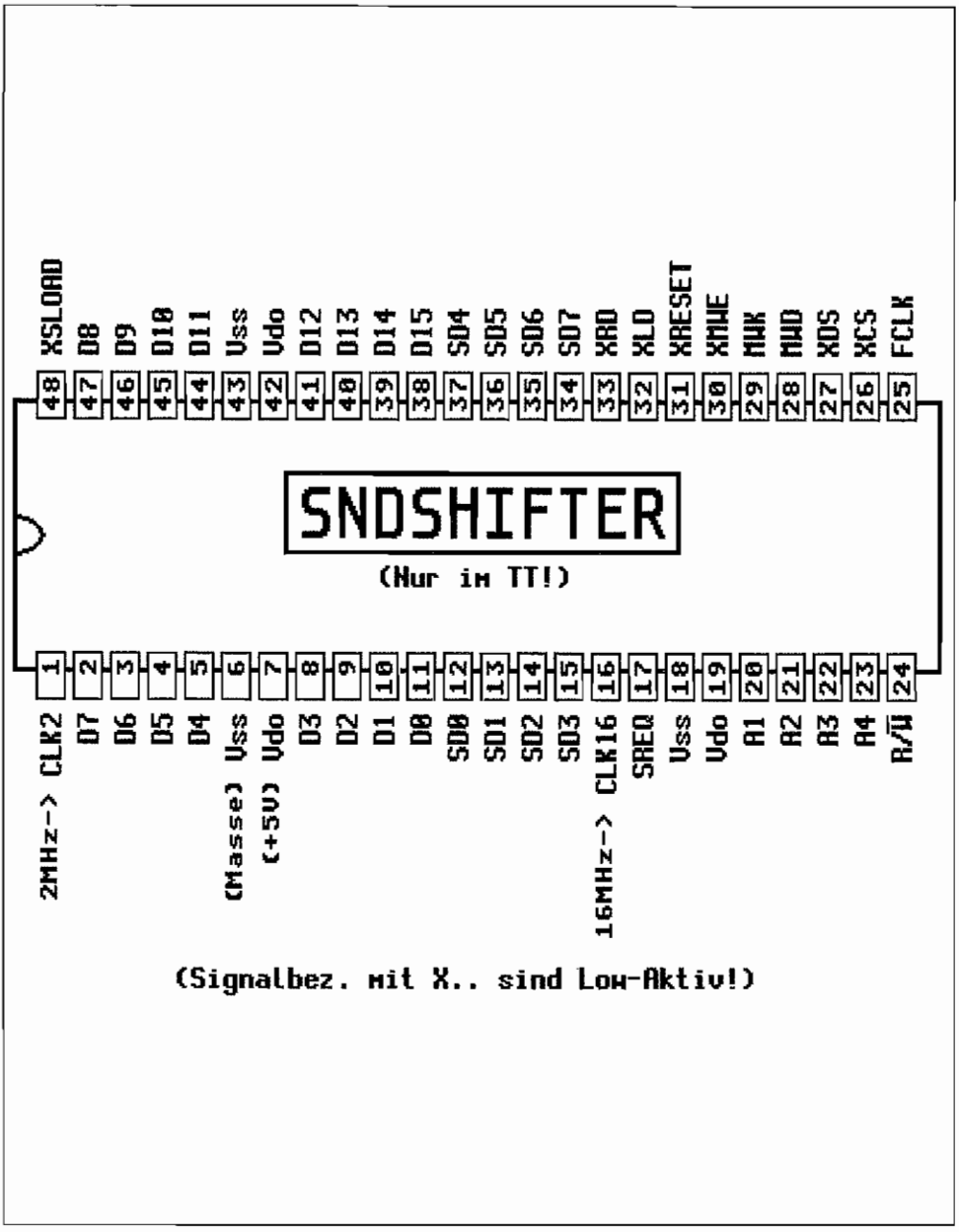


Abb. N.19: Pinbelegung des TT-SNDSHIFTER

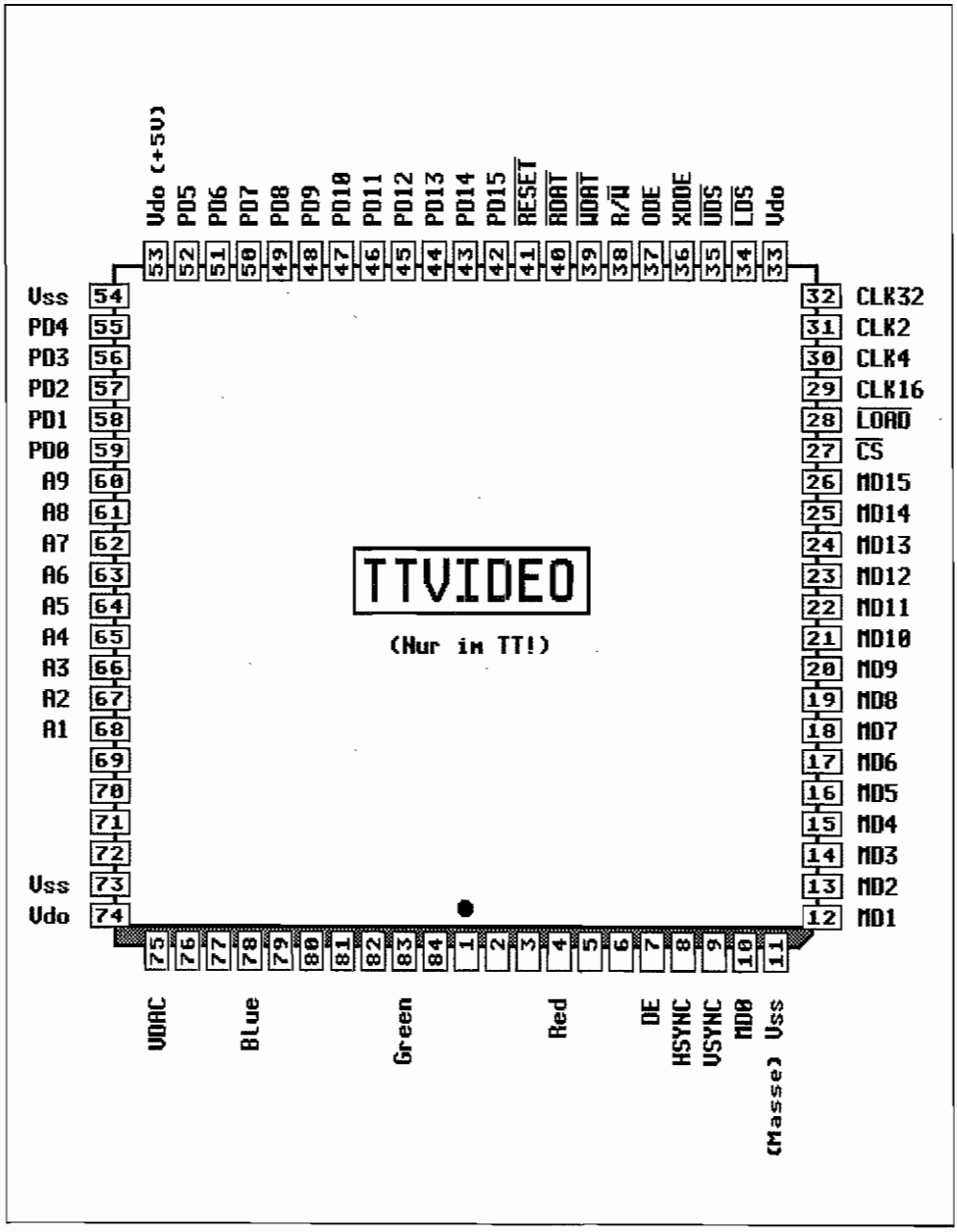


Abb. N.20: Pinbelegung des TTVIDEO-Chips

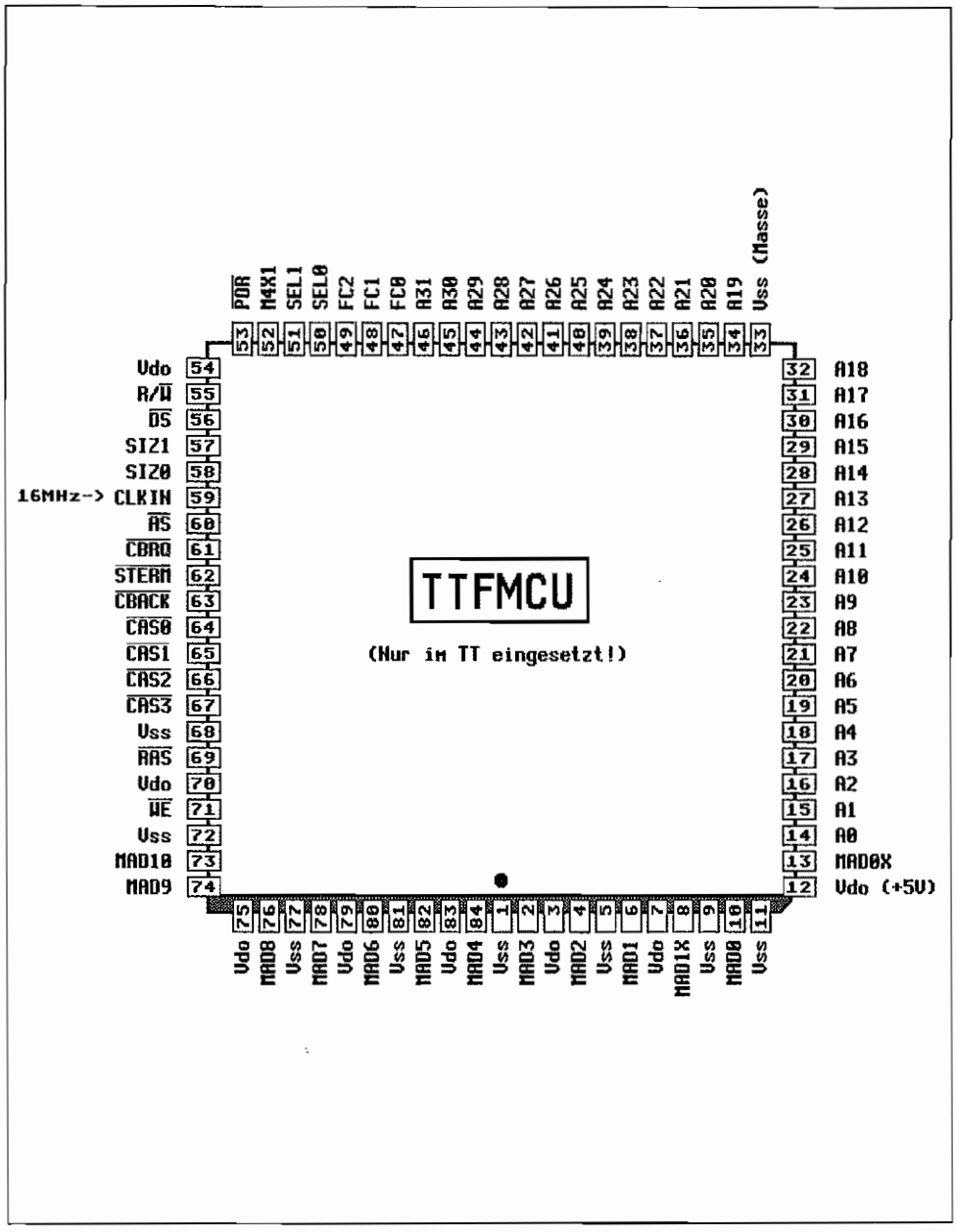


Abb. N.21: Pinbelegung der TT-Fast-MCU

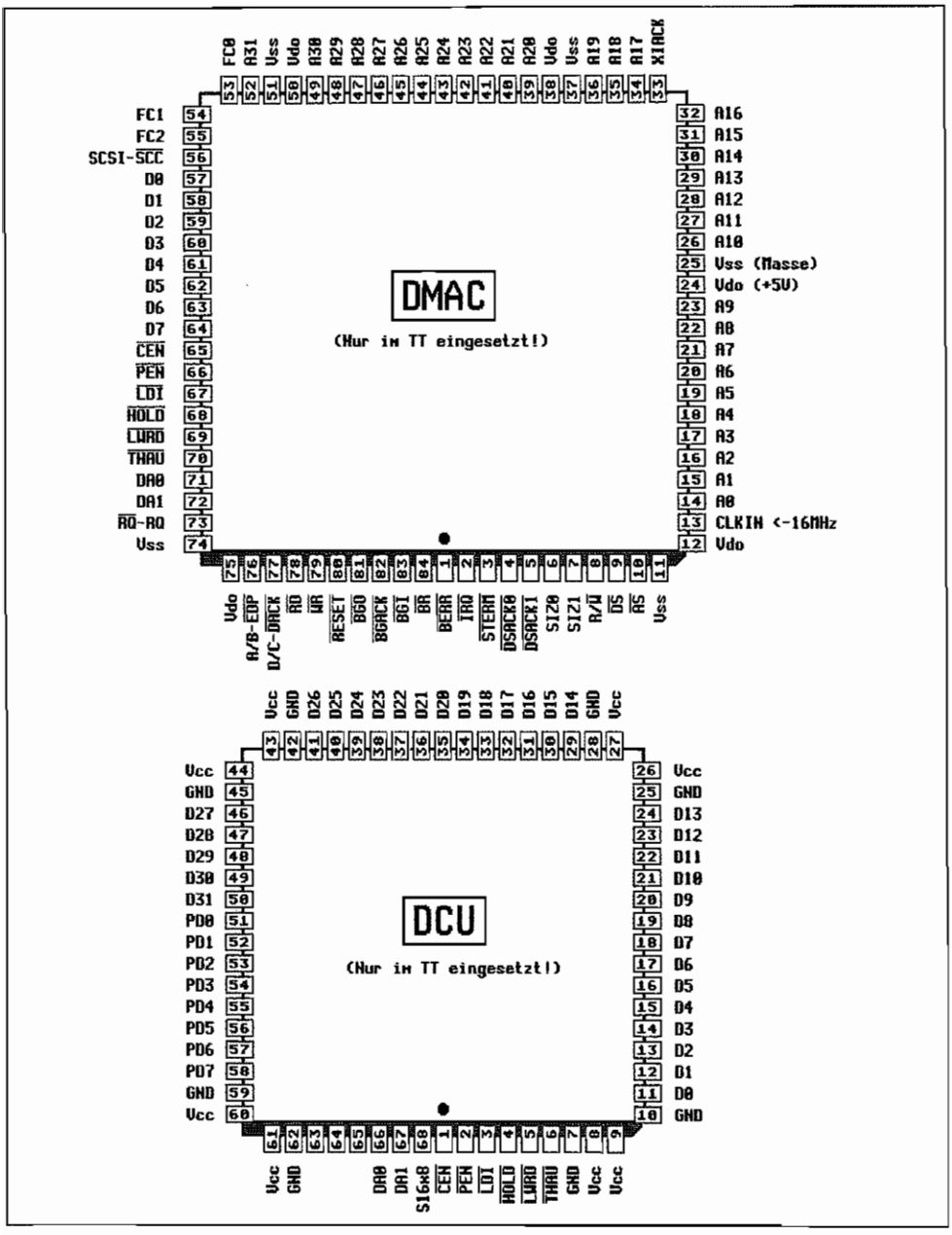


Abb. N.22: Pinbelegung des TT-DMAC und des TT-DCU-Chips

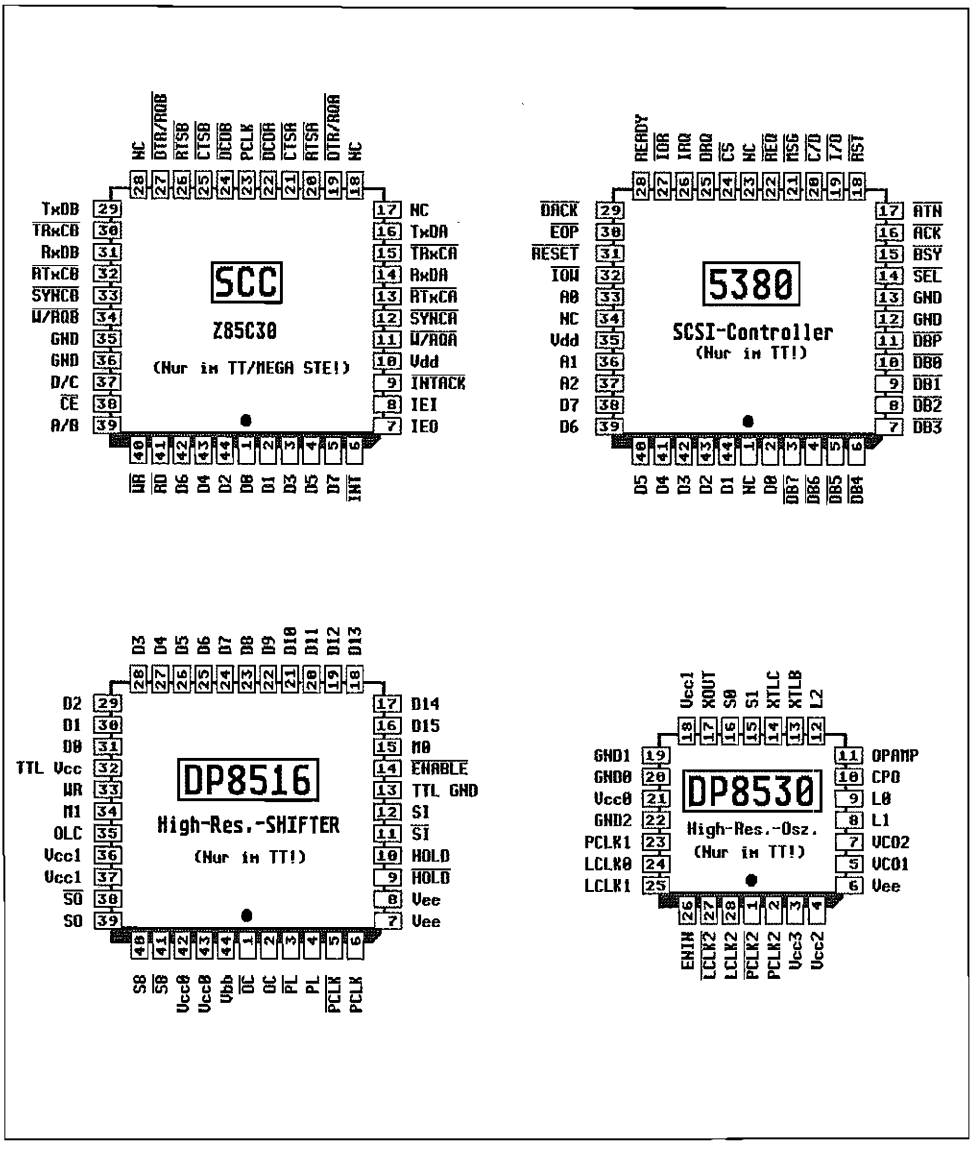


Abb. N.23: Pinbelegung des STE/TT-SCC
Pinbelegung des TT/SCSI-Controllers
Pinbelegung des TT-High-Resolution-SHIFTER
Pinbelegung des TT-High-Resolution-Oszillators

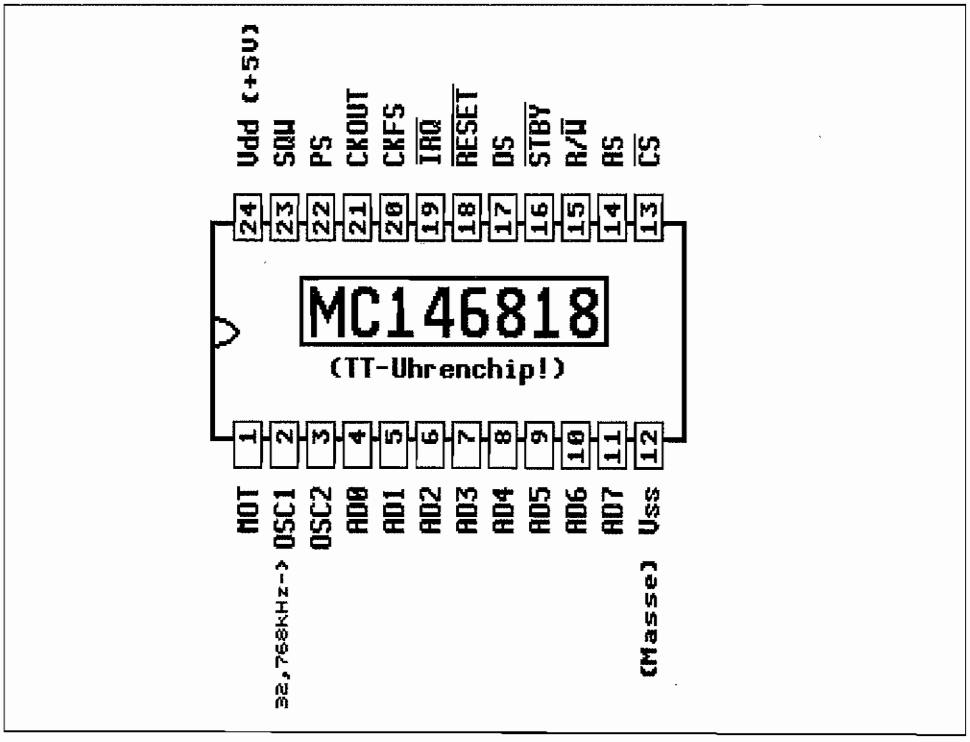


Abb. N.24: Pinbelegung des TT-Uhrenchips

Anhang O: Quellen und weiterführende Literatur

Dokumentation von Atari

"AHDI 3.00 Release Notes", Atari Corporation, 18. April 1990

"A Hitchhiker's Guide to the BIOS", Revision 1.1, Atari Corporation, 3. Mai 1990

"ALN Manual", Allan Pratt, Atari Corporation, 6. Juli 1987

"Atari CDAR 504 Developer Reference Manual", Mike Schmal, Atari Corporation, 14. September 1988

"Atari GEMDOS Reference Manual", Atari Corporation, 30. Juli 1990

"Atari MetaDOS Developers Manual", Mike Schmal, Atari Corporation, 13. Juni 1990

"Atari TT030 Hardware Reference Manual", Atari Corporation, Juni 1990

"Atari TT030 VME Expansion", Atari Corporation, 1991

"Engineering Hardware Specification of the Atari ST Computer System", Atari Corporation, 7. Januar 1986

"GEMDOS Extended Argument (ARGV) Specification", Ken Badertscher, Atari Corporation, 1. November 1989 (in: "Info-Atari16, Volume 89, Issue 595")

"GEM Programmer's Guide Volume 1: VDI", Third Edition, Atari Corporation, Januar 1989

"GEM Programmer's Guide Volume 2: AES", Third Edition, Atari Corporation, Januar 1989

"Intelligent Keyboard (ikbd) Protocol", Atari Corporation, 26. Februar 1985

"MadMac Manual", Landon Dyer, Atari Corporation, 12. August 1987

"Pexec Cookbook", Atari Corporation, 19. März 1991

"Rainbow TOS Release Notes", Atari Corporation, 7. August 1989

"S.A.L.A.D. – Still another Line A Document", Mark Jansen, Atari Corporation, 17. Dezember 1987

"Specification for Atari GEMDOS File Sharing & Record Locking", Mike Fulton, Atari Corporation, in: "ATARI.RSC – The Atari Developers Resource", Vol. IV, Issue 2, April/May 1991

"STE Developer Addendum", Atari Corporation

"STE TOS Release Notes", Atari Corporation, 12. Januar 1990

"The Atari Page Printer Interface", Atari Corporation, 3. Februar 1988

"The long awaited Line "A" Document", Atari Corporation, 1985

"TT030 TOS Release Notes", Atari Corporation, 8. Oktober 1990

"User Manual for the Atari ST Bit-Block Transfer Processor (Blitter)", Revision 2.2, Atari Corporation, 17. Juni 1987

"XCONTROL – Extensible Control Panel for ST/MEGA/STe/TT Computers", Atari Corporation, Version 1.0, 1991

Weitere Dokumentationen

Advanced Micro Devices: "Am53C80 SCSI-Interface Controller", 1988

Advanced Micro Devices: "Z85C30 Serial Communication Controller", 1989

American National Standards Institute, Inc. (ANSI): "American National Standard for Information Systems - Programming Language – C", ANSI X3.159-989 (die definitive Informationsquelle für alles, was mit der Programmiersprache C zu tun hat)

American National Standards Institute, Inc. (ANSI): "SMALL COMPUTER SYSTEMS INTERFACE (SCSI)", ANSI X3T9.2/82-2, Revision 17B (Definition des SCSI-Interfaces)

Digital Research: "Introduction to GEM Programming"

Digital Research: "GEM VDI Reference Guide GEM Version 2.0", 1986

Digital Research: "GEM AES Reference Guide GEM Version 2.0", 1986

General Instruments: "Programmable sound Generator AY-3-8910", Februar 1979

Hitachi Semiconductors: "HD6301 CMOS Microcontroller Unit"

Maxon Computer: "MGE MAXON Graphic Expansion, Handbuch", MAXON Computer, Eschborn, 1989

Motorola Semiconductors: "M68000 16/32-Bit Microprocessor", Motorola Inc. 1984

Motorola Semiconductors: "MC68901 – Multi-Function Peripheral"

Motorola Semiconductors: "Asynchronous Communications Interface Adapter (ACIA), MC6850"

Motorola Semiconductors: "MC68030 Enhanced 32-Bit Microprocessors User's Manual", Prentice Hall

Motorola Semiconductors: "MC68881/68882 Floating-Point Coprocessors User's manual, Second Edition", Prentice Hall

Morrison, Jerry (Electronic Arts): "Electronic Arts IFF 85 Standard for Interchange Format Files"

National Semiconductor: "DP8516 Video shift Register", 1988

National Semiconductor: "DP8530 Clock Generator", 1987

Tim Oren (Digital Research): "Professional GEM", Antic Publishing

Valvo Unternehmensbereich Bauelemente der Philips GmbH: "Der 16bit-Microprocessor SC 68000, Eigenschaften"

Western Digital Corporation: "WD1770/1772 Floppy Disk Controller/Formatter"

Bücher

Abraham, Englisch, Günther, Szczepanowski: "Atari ST GEM", Data Becker, Düsseld. 1986, ISBN 3-89011-251-X (enthält VDI-Listing und Auszüge aus AES-Listing vom GEM 1.0)

Addison Wesley Publishing Company, Inc.: "Inside Macintosh" (insbesondere das Kapitel über "USER INTERFACE GUIDELINES")

Asimov/Silverberg: "Nightfall", Pan Books Ltd, London 1991, ISBN 0-330-32096-3

Aumann, Maier, Stöpper: "Atari ST – Das Floppy Arbeitsbuch", SYBEX, Düsseldorf 1986, ISBN 3-88745-642-4

Balma/Fitler: "GEM-Programmier-Handbuch", SYBEX, 1987, ISBN 3-88745-692-0 (in erster Linie für PC-GEM; die Autoren haben bei DR an der Entwicklung von GEM mitgewirkt)

Banahan, Mike: "The C Book, Featuring the Draft ANSI C Standard", Addison-Wesley, Bonn 1988, ISBN 0-201-17370-0 (das definitive Buch für Leute, die C lernen wollen oder sich für den neuen ANSI-Standard interessieren)

Brod/Stepper: "Scheibenkleister II – Massenspeicher am ST", Maxon Computer GmbH, Eschborn 1989, ISBN 3-927065-00-5 (wohl ausführlichstes Buch zu Floppies und Festplatten)

Brückmann, Englisch, Gerrits: "Atari ST Intern", Data Becker, Düsseldorf 1986, ISBN 3-89011-119-X (enthält in der 2. Auflage ein Source-Listing von BIOS und XBIOS im TOS 1.0)

Dannloff, Craig & McClelland, Duke: "Das Macintosh Buch", SYBEX, Düsseldorf 1990, ISBN 3-88745-770-6

Detering, Reinhard: "UNIX Handbuch System V", SYBEX, Düsseldorf 1989, ISBN 3-88745-577-0

Durant/Carlson/Yao: "Programmers's Guide to Windows", SYBEX, Alameda 1987, ISBN 0-89588-496-8

Electronic-Sonderheft: "Der VME-Bus", Franzis-Verlag 1986

Gabler, Hermann: "Grundlagen der Text- und Datenuebermittlung", R. v. Decker's Verlag Heidelberg, ISBN 3-7685-1188-X (wer in die Technik der professionellen Text- und Datenübermittlung "reinriechen" will, ist mit diesem Buch bestens bedient)

Geiß, Dieter und Jürgen: "Vom Anfänger zum GEM-Profi", Hüthig Verlag, Heidelberg 1990

Hofstadter, Douglas: "Metamagical Themas: Questing for the Essence of Mind and Pattern", Basic Books Inc. 1985, ISBN 0-14-008534-3

Illik, J. A.: "Erfolgreich programmieren in C", SYBEX, Düsseldorf 1985, ISBN 3-8745-055-8

Jankowski/Rabich/Reschke: "Atari ST Profibuch", SYBEX, Düsseldorf 1987, 1988, ISBN 3-88745-501-0 (ein Buch mit ausführlichem Literaturnachweis)

Kernighan/Ritchie: "The C Programming Language, 2nd Edition", Prentice-Hall 1988, ISBN 0-13-110362-8

Knuth, Donald: "The TeXbook", Addison Wesley Publishing Company, 1984, ISBN 0-201-13448-9

Kofler, Michael: "Das Atari ST Grafikbuch", SYBEX, Düsseldorf 1987, ISBN 3-88745-673-4

Kramer/Riebl/Hübner: "Das TOS-Listing, BIOS-GEMDOS-VDI", Heinz Heise-GmbH, Hannover 1988, ISBN 3-88229-002-1 (enthält Sourcelisting von BIOS, XBIOS, GEMDOS und AHDI aus TOS 1.02)

Kraus, Helmut: "Mikrocomputerlexikon", SYBEX, Düsseldorf 1989, ISBN 3-88745-518-5

IBM: "IBM Systems Application Architecture, Common User Access, Advanced Interface Design Guide", International Business Machines Corp., 1989. Doc.-Number SY0328-300-R00-1089

Lammers, Susan: "Faszination Programmieren", Markt&Technik-Verlag, Haar 1987, ISBN 3-89090-418-1 (enthält interessante Interviews mit führenden Programmierern)

Mathy, Frank: "Programmierung von Grafik und Sound auf dem Atari ST", Markt&Technik Verlag, Haar 1987, ISBN 3-89090-405-X

Newmann/Sproull: "Grundzüge der interaktiven Computergrafik", McGraw-Hill, Hamburg 1986, ISBN 3-89028-015-3

Nieber, Christian: "Atari ST – Programmieren in Maschinensprache", SYBEX, Düsseldorf 1987, ISBN 3-88745-678-5

Peel, Katherine: "The concise Atari ST 68000 programmer's reference guide", Glentop Publishers Ltd. 1986, ISBN 1-85181-017-X

Reschke/Wiethoff: "Atari Profibuch", SYBEX, Düsseldorf 1985, ISBN 3-88745-605-X (ausführliche Dokumentation zu einem Rechner, der mit dem ST gar nichts zu tun hat)

Rosenbeck, Peter: "C-Programmierung unter TOS", Markt&Technik Verlag, Haar 1985, ISBN 3-89090-226-X

The Institute of Electrical und Electronics Engineers, Inc. (IEEE): "Information technology - Portable Operating System Interface (POSIX) - Part 1: System Application Program Interface (API) [C Language]", IEEE Std 1003.1-1990, ISO/IEC 9945-1, ISBN 1-55937-061-0

Schulz, A.: "Software-Entwurf, Methoden und Werkzeuge", R. Oldenbourg Verlag, München, Wien 1988, ISBN 3-486-20690-7

Vieillefond, C.: "Programmierung des 68000", SYBEX, Düsseldorf 1985, ISBN 3-88745-060-4

Williams, Steve: "Programming the 68000", SYBEX, San Francisco 1985, ISBN 0-89588-133-0 (ein sehr ausführliches Buch mit interessantem Anschauungsmaterial; der Autor hat bei Digital Research gearbeitet und geht daher insbesondere auf CP/M 68K ein – dabei trifft man auf alte Bekannte wie AS68, LO68 und SID)

Wollschläger, Peter: "Atari ST Assemblerbuch", Markt&Technik Verlag, Haar 1987, ISBN 3-89090-467-X

X/OPEN: "X/Open Portability Guide, Volume 1: XVS Commands and Utilities", Elsevier Science Publishing, Amsterdam 1987, ISBN 0-444-70174-5

Young, Michael J.: "Programmer's Guide to the OS/2 Presentation Manager", SYBEX, Alameda 1989, ISBN 0-89588-569-7

Zeitschriften

"Atari Journal – Die Fachzeitschrift für ST und TT", Heim Fachverlag, Darmstadt

"Byte", McGraw-Hill, Inc. U.S., ISSN 0360-5280/91

"c't – Magazin für Computertechnik", Heinz Heise GmbH Hannover, ISSN 0724-8679

"ST-Computer – Die Fachzeitschrift für Atari ST- und TT-Anwender", Heim Fachverlag Darmstadt (insbes. Artikelserie von Alex Esser im 1. Sonderheft und ab Ausgabe 12/1987)

"ST-Magazin – Alles über Atari ST und TT", Markt&Technik Verlag AG, Haar b. München

"TOS – Magazin plus Software für den Atari ST & TT", ICP-Innovativ Computer-Presse Verlag GmbH & Co. KG, Vaterstetten

Stichwortverzeichnis

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

PROFIBUCH

ATARI Profibuch ST-STE-TT

Empfohlen für:
- ✓ Einsteiger
- ✓ Fortgeschrittene
- ✓ Profis

- ✓ Anleitung
- ✓ Referenz
- Tools

Buchregal-Systematik
Atari ST/STE/TT: Software/Hardware/Peripherie

Die Autoren Hans-Dieter Jankowski, Dietmar Rabich und Julian F. Reschke haben wieder ihr geballtes Wissen zu den Atari-Rechnern der ST-, STE- und TT-Modellreihen zusammengetragen und übersichtlich dokumentiert.

Jetzt neu in dieser komplett überarbeiteten Ausgabe:

- Dokumentation zu allen neuen TOS-Versionen aus STE, MEGA STE und TT
- Ausführliche „User Interface Guidelines" mit Programmbeispielen
- Der Cookie-Jar
- Die Programmierung von XCONTROL-Modulen
- Das ARGV-Verfahren
- Völlig überarbeitete einführende Kapitel zu BIOS, XBIOS, GEMDOS, VDI und AES
- Neue Hardware in STE und MEGA STE (Grafik, DMA-Sound, LAN-Schnittstelle [SCC], Microwire-Interface, VME-Bus)
- Weitere neue Hardware im TT030 (Grafik, TT-MFP, serielle Schnittstellen, SCSI-Controller, ST-RAM und TT-RAM, der MC68881/2, die SCU, Realtime-Clock)
- Standardbefehlssatz von SCSI-Festplatten

Außerdem finden Sie auch alle „altbewährten" Themen – also Betriebssystem und Hardware aller ST-Modelle – wieder.

Best.-Nr. 3888
ISBN 3-88745-888-5
DM 79,– / öS 616,–

SYBEX

9 783887 458881