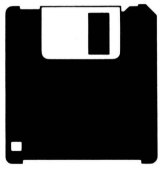
Markt&Technik

Mit großem Farbteil über alle wichtigen Bauteile

Frank Kremser
Jörg Koch

AMIGA SYSTEM-HANDBUCH

Für Amiga 500, 1000 und 2000
Eine detaillierte Hardware-Beschreibung: MC68000
★ Custom-Chips ★ Slots ★ Schnittstellen ★ Floppy
★ Hardware-Erweiterungen wie vergrößerter Speicher
★ Genlock-Interface ★ Scanner ★ Zahlreiche
Beispiele in C und Assembler



Auf 3½"-Diskette enthalten:
Alle Programme als Source-Code
und in lauffähiger Version.

Frank Kremser
Jörg Koch

AMIGA

SYSTEM-HANDBUCH

Für Amiga 500, 1000 und 2000
Eine detaillierte Hardware-Beschreibung:
MC68000 ★ Custom-Chips ★ Slots
★ Schnittstellen ★ Floppy ★ Hardware-
Erweiterungen wie vergrößerter Speicher
★ Genlock-Interface ★ Scanner
★ Zahlreiche Beispiele in C und Assembler

Markt&Technik Verlag AG

CIP-Titelaufnahme der Deutschen Bibliothek

Kremser, Frank:
Amiga-System-Handbuch : für Amiga 500, 1000 u. 2000 ;
e. detaillierte Hardware-Beschreibung: MC 68000, Custom-Chips, Slots, Schnittstellen, Floppy,
Hardware-Erweiterungen wie vergrößerter Speicher, Genlock-Interface, Scanner,
zahlr. Beispiele in C u. Assembler / Frank Kremser ; Jörg Koch. –
Haar bei München : Markt-u.-Technik-Verl., 1988
ISBN 3-89090-550-1
NE: Koch, Jörg

15 14 13 12 11 10 9 8 7 6 5 4
91 90

ISBN 3-89090-550-1

Einbandgestaltung: Grafikdesign Heinz Rauner
Druck: Schoder Druck GmbH & Co. KG, Gersthofen
Printed in Germany

Inhaltsverzeichnis

Vorwort

Technologischer Fortschritt und technische Innovation haben im Computerbereich keinen Halt gemacht. Der Amiga bietet derzeit das Neueste vom Neuen. [Er ersetzt die 8-Bit-Spielkameraden im Kinderzimmer, bringt dem Vater Leistungsfähigkeit beim Arbeiten mit CAD, Hausverwaltung oder Entspannung mit einem kleinen Spiel].

Um ein so komplexes System zu beherrschen, dem Anwender Spaß bei der Arbeit oder vielen Jugendlichen Freude beim Spielen zu bereiten, muß der Profi, Freak oder Hobbyprogrammierer näher in die Tiefen des Amiga-Systems einsteigen, denn nur auf dieser Ebene wird er die Lösungen finden, die seine Software leistungsfähig machen. Dies wird als maschinennahes Programmieren bezeichnet und soll ein Bestandteil unseres Hardware-Buches sein.

Maschinennah programmieren bedeutet nicht gleich den verstaubten Assembler aus der Diskettenbox zu holen, denn maschinennahe Programmierung läßt sich auch in C verwirklichen. Zwar wird damit nicht die maximale Geschwindigkeit des Systems erreicht, es lassen sich jedoch Programme einfacher entwickeln und in andere Programme einbinden. An die Assembler-Freaks haben wir auch gedacht und Demos in Maschinensprache dem Buch beigefügt. Alle Programme befinden sich auf Diskette, so daß ein aufwendiges Abtippen der Programme entfällt.

Auch an die Hobbybastler, die keinen Rechner kaufen können, ohne ihn aufzuschrauben und auseinandernehmen zu wollen, haben wir gedacht. Tips und Tricks zu Systemerweiterungen, Funktion des Systems und eine Bauanleitung für ein 20-DM-Genlock-Interface sind sicher große Leckerbissen für solche Freaks.

Wir haben in diesem Buch versucht, alle Anwender der Amiga-Serie zufriedenzustellen und gleich das Buch für die gesamte Amiga-Palette geschrieben. Dadurch kam oftmals Chaos in unserem »Entwicklungslabor« auf, denn bei drei »Freundinnen« mit Zubehör weiß man oftmals nicht, welche Schraube zu welchem Rechner gehört.

Alle Schaltungen und Programme sind weitgehend erprobt, so daß es keine Schwierigkeiten beim Nachbau bzw. der Anwendung der Software geben dürfte. Hobbybastler

mit nicht allzu großer Erfahrung können sich ruhig an den Nachbau einiger kritischer Schaltungen, wie interne RAM-Aufrüstungen, wagen. Die Amigas sind sehr robust und können einiges vertragen. Spätestens jedoch, wenn ihr Amiga bei dem Anzeigen einer Guru-Meditation abstürzen sollte, raten wir, das jeweilige Kapitel der Bauanleitung nochmals durchzulesen und den Aufbau der Schaltung sorgfältig zu überprüfen.

Zum Schluß möchten wir noch einigen Personen danksagen, denn an der Entwicklung eines solchen Buches sind meistens nicht nur die Autoren, die zwar den größten Teil der Arbeit haben, sondern auch viele andere Personen beteiligt, die Tips und Hilfestellungen zu diesem Projekt geben. Unser Dank gilt insbesondere

- allen Mitarbeitern von Markt & Technik, die zur Verwirklichung dieses Buches beigetragen haben, vor allem Christine Baumann, die uns die Anregung zu diesem Buchprojekt und permanente Unterstützung während der Verwirklichung dieses Projektes gegeben hat.
- den Mitarbeitern des Commodore State Support in Frankfurt, hier besonders dem Leiter Herrn Härtel und Herrn Kakadures, die sicherlich an unseren umfangreichen Fragestellungen fast verzweifelt sind.
- Herrn Knobel von der Firma Electronic Medical, der uns bei der Entwicklung des Genlock-Interfaces mit seinem fachlichen Wissen im Bereich der Fernsehtechnik und mit Meßgeräten mit Rat und Tat zur Seite stand.
- unseren Eltern, die großes Verständnis gezeigt haben, daß so manche selbstverständliche Arbeit unerledigt blieb, weil wir noch einige Verbesserungen an dem Buch vornehmen wollten.

Marburg/Karlsruhe im Jahre 1988

Einführung

In dieser Einführung gehen wir auf die *C*-Programmierung und auf die Bedienung des *Seka*-Assemblers ein, den wir für die Entwicklung der Assembler-Programme verwendet haben. Natürlich können nicht alle Einzelheiten dargestellt werden, da dies den Rahmen des Buches sprengen würde. Als Einstieg dürften die Informationen allerdings ausreichend sein.

Die C-Programme, die in diesem Buch aufgeführt sind, sind mit dem Lattice-Compiler Version 3.10 erstellt worden. Aber mit den unten aufgeführten Batch-Files dürften sie auch auf den anderen Versionen, bzw. dem Aztec-Compiler fehlerfrei laufen.

Zur Vereinfachung des Kompiliervorganges bei C-Programmen haben wir ein Batch-File geschrieben, das alle Kompilier- und Linkphasen selbständig durchführt. Dieses *Batch-File* haben wir für eine Harddisk geschrieben, aber es ist ohne Änderungen auch für die Arbeit mit zwei Laufwerken geeignet.

Für die Lattice-Version 3.02 oder 3.03:

```
stack 20000
if not exists <prg>.c
  echo "File ist nicht vorhanden"
  skip end
  endif
echo "-- kompilieren --"
lcl -i:include/ -i:include/lattice/ <prg>.c
if not exists <prg>.q
  echo "Compiler-Fehler"
  quit 20
  endif
lc2 <prg>
alink : lib/lstartup.obj+<prg>.o library : lib/lc.lib+
:lib/amiga.lib to <prg> map nil:
delete <prg>.o
echo "-- Kompilier- und Linkvorgang ist zu Ende --"
lab end
```

Für die Lattice-Version 3.10:

```
stack 20000
if not exists <prg>.c
  echo "File ist nicht vorhanden"
  skip end
  endif
echo "-- kompilieren --"
LC1 -f -i:include/ -i:include/lattice/ <prg>.c
if not exists <prg>.q
  echo "Compiler-Fehler"
  quit 20
  endif
LC2 -cdb <prg>
BLINK FROM LIB:c.o+<prg>.o TO <prg> LIB LIB:lc.lib+lcmffp.lib+
  LIB:amiga.lib+LIB:lcm.lib
delete <prg>.o
echo "-- Kompilier- und Linkvorgang ist zu Ende --"
lab end
```

Für die Lattice-Version 4.00:

```
stack 20000
if not exists <prg>.c
  echo "File ist nicht vorhanden"
  skip end
  endif
echo "-- Kompilieren --"
LC1 -f -i:include/ -i:include/lattice/ <prg>.c
if not exists <prg>.q
  echo "Compiler-Fehler"
  quit 20
  endif
LC2 <prg>
BLINK FROM LIB:c.o+<prg>.o TO <prg> LIB LIB:lc.lib+lcmffp.lib+
  LIB:amiga.lib+LIB:lcm.lib
delete <prg>.o
echo "-- Kompilier- und Linkvorgang ist zu Ende --"
lab end
```

Anmerkung: Leider funktionierte bei uns nicht die Compiler-Option »-cdb« – Programmdaten nur im Chip-Mem – in Verbindung mit LC2V4.00. Abhilfe schafft ein Trick von Peter Wollschlaeger, den Sie im »Liesmich«-File seines Buches »Amiga Programmierpraxis Intuition« finden:

Variablen wie IntuitionBase sind systemglobal. Deklariert man Sie als extern, kann man z.B. den Data-Hunk ins Chip-Memory bringen mit:

```
lc -L -ad none
```

Den auftretenden Fehler »b ignored« können Sie ignorieren.

Für den Aztec-Compiler:

```
cc -t <prg> .c +l
ln <prg> .o -lm32 -lc32
echo "-- Kompilier- und Linkvorgang ist zu Ende --"
```

Dieses File muß mit dem Editor »ed« eingegeben und gespeichert werden. Dazu gehen Sie von der Workbench aus in das CLI. Dort erscheint »1>«. Nun kommen Sie mit »ed comp« in den Editor und können das Batch-File, das für Ihre Compiler-Version angegeben ist, eingeben. Wenn Sie fertig sind, können Sie es abspeichern, indem Sie »ESC« und anschließend »x« drücken. Das Batch-File steht nun unter dem Namen »comp« auf Ihrer Diskette, bzw. Harddisk. Wenn Sie später ein selbstgeschriebenes C-Programm compilieren wollen, starten Sie es mit dem CLI-Befehl »EXECUTE comp« und dem Programmnamen des C-Programms ohne das ».c«-Kürzel.

Beispiel: Ihr C-Programm steht unter dem Namen ›test.c‹ auf der Harddisk, bzw. Diskette. Kompilieren Sie es nun, indem Sie ›execute comp test‹ eingeben. Nach Beendigung des Kompiliervorgangs steht das ausführbare Programm unter dem Namen ›test‹ auf der Harddisk, bzw. Diskette.

Nun, was bewirkt dieses Batch-File für den Lattice. Zu Beginn wird der Programmname der Variablen »prg« übergeben und anschließend der Stack auf eine Größe von 20 000 Byte gesetzt, was nötig ist, da der Compiler eine Vielzahl von Daten zwischenspeichern muß. Anschließend überprüft es, ob das gewünschte Programm zum Kompilieren überhaupt existiert. Ist das Programm nicht vorhanden, steigt das Batch-File aus und druckt die Fehlermeldung »File ist nicht vorhanden«. Ist kein Fehler aufgetreten, so beginnen nun die Kompiliervorgänge »lc1« und »lc2«. »lc1« überprüft hauptsächlich die Syntax des Hauptprogramms und der eingeladenen Include-Files. »lc2« generiert anschließend den Programmcode. Tritt beim Kompiliervorgang »lc1« ein Fehler auf, so wird auch hier der Ablauf des Batch-Files gestoppt und eine Fehlermeldung »Compiler-Fehler« ausgegeben. Verliefen jedoch die Kompiliervorgänge ohne Fehler ab, beginnt das Programm mit dem Zusammenfügen, sprich »Linken«, der Bibliotheksmodule mit dem Programmcode. Eine Meldung teilt dem Benutzer anschließend mit, daß dieser Prozeß beendet ist. Danach kann das »kompilierte« und »gelinkte« Programm gestartet werden. Das Programm muß unter dem gewünschten Namen, mit angehängtem ».c« erstellt und abgespeichert worden sein, also beispielsweise »test.c«. Nach dem Kompilier- und Linkvorgang steht der startbare Programmcode in der Datei »test«. Dieser Programmcode läßt sich nun einfach durch Eintippen des Dateinamens starten. Bei den meisten Programmen bietet sich auch noch die Möglichkeit an, eine ».info«-Datei zu kopieren, beispielsweise »copy cli.info to test.info«. Dann können die Programme auch von der Workbench aus gestartet werden. Eine Ausnahme bilden hier die Programme, die »printf«, »scanf« oder DOS-Befehle benutzen und mit der Lattice-Version 3.02 oder 3.03 kompiliert wurden, da diese Befehle ihre Ein- und Ausgabe über

CLI abwickeln. Aus diesem Grund besitzen auch nicht alle Demonstrationsprogramme auf der mitgelieferten Diskette sog. Icons, sind also von der Workbench aus nicht sicht- oder startbar. Sie können nur von CLI aus gestartet werden.

Für alle nachfolgenden Erklärungen möchten wir Sie bitten, alle Schritte direkt am Computer nachzuvollziehen, da es dann leichter für Sie wird. Zu Beginn müssen Sie natürlich den Computer starten. Falls Sie es nicht schon zuvor getan haben, müssen Sie das Preferences-Programm starten und den CLI-Schalter auf »ON« setzen, da nur in diesem Fall CLI zu verwenden ist. Anschließend können Sie wieder Preference verlassen, am besten mit »Save«, da dann das CLI-Icon auch nach dem Einschalten des Computers erscheint. Nun starten Sie bitte CLI durch einen Doppelklick auf das CLI-Icon. Kurz darauf erscheint das CLI-Window. Ist es das einzige CLI-Window auf dem Bildschirm, so müßte das »1›«-Prompt darin erscheinen. Dahinter ist der Cursor zu erkennen. Nun können Sie sämtliche CLI-Befehle verwenden. Als Beispiel dafür tippen Sie bitte »dir« mit anschließendem »RETURN« ein. Sie sehen nun das Inhaltsverzeichnis der Hauptdiskette.

Nun wollen wir mit der Einführung in »C« beginnen. »C« ähnelt in vielen Punkten den Programmiersprachen PASCAL und MODULA, weshalb PASCAL- und/oder MODULA-Programmierer keine Schwierigkeiten haben dürften, auf »C« umzusteigen.

»C« wurde 1972 in den USA entwickelt, 1973/74 verbessert und anfangs vornehmlich unter dem Betriebssystem UNIX verwendet. Da diese Sprache möglichst flexibel sein sollte, wurden ihr nur sehr wenige Befehle fest implementiert. Darunter sind:

- if ⇨ Bedingte Anweisung
- switch ⇨ Bedingte Anweisungen
- for, while ⇨ Zähl- und bedingte Schleifen

Es sind noch einige Befehle mehr vorhanden, auf die wir allerdings nicht eingehen werden, da sie nicht von größerer Bedeutung sind. Diese Befehle genügen allerdings, um alle programmkontrollierenden Funktionen durchführen zu können, zumal sich die meisten Befehle in Include-Dateien oder in Bibliotheken befinden, die der Sprache »C« zu ihrer Leistungsfähigkeit und Flexibilität verhelfen.

Ein C-Programm setzt sich normalerweise aus drei Grundteilen zusammen. Im ersten Teil gibt der Programmierer an, welche Include-Dateien oder Bibliotheken er verwenden will und somit beim Kompilieren eingelesen werden müssen. Im anschließenden, zweiten Teil werden die globalen Variablen deklariert. Auf diese Variablen kann von jeder Routine des Programms aus zugegriffen werden. Der dritte Teil besteht aus dem eigentlichen Progamm. Dieser Teil gliedert sich allerdings wieder in zwei Teile auf. Da ist zum einen der Teil mit den Unterroutinen – In Pascal auch Procedures oder Functions genannt – und zum anderen der sogenannte »main«-Teil. Dieser Teil stellt die Hauptroutine des Programms dar, die beim Start des Programms aktiviert wird. Der

Unterschied zu Pascal besteht darin, daß Unterroutinen und Hauptprogramm keine festgelegte Reihenfolge haben müssen. Es können in »C« also auch Unterroutinen aufgerufen werden, die erst später im Programm folgen.

C-Programme wirken leider teilweise sehr undurchsichtig, was aber durch das Einfügen von Kommentaren und Unterroutinen wettgemacht werden. Solche Kommentare werden mit »/*« und »*/« geklammert. Aufpassen sollte man auf solche »Kleinigkeiten« wie Semikolons oder Kommata, da der Compiler oftmals solche Fehler nicht erkennt, sondern weiterkompiliert, was zum Absturz beim späteren Starten des Programms führen kann. Mit »C« zu arbeiten heißt also korrekt und sauber arbeiten, sonst kann für nichts garantiert werden.

Zusätzlich zu den oben genannten Befehlen besitzt »C« noch eine Reihe von Funktionen, von denen »printf« und »scanf« die wichtigsten sind. »printf« dient zur Ausgabe von Texten, Zahlen u.a. Mittels »scanf« können Texte und Zahlen eingelesen werden. Zu der Funktion »printf« wollen wir an dieser Stelle ein erstes Programm erstellen. Es soll nur den Text »Mein erstes C-Programm« ausgeben. Da Sie sich schon in CLI befinden, müssen Sie nur noch den Editor »ed« aktivieren. Zusätzlich muß noch der Name angegeben werden, den unser Programm haben soll, gefolgt von ».c«:

```
1 > ed test.c
```

Nun befinden Sie sich im Editor. Da wir nur den Befehl »printf« verwenden wollen, der von »C« bereitgestellt wird, müssen wir keine Include-Dateien einlesen oder Variablen deklarieren. Also können wir gleich mit dem Programm beginnen. Das Programm besteht in unserem Fall nur aus der Hauptroutine, die mit »main()« eingeleitet wird. Anschließend folgen die Anweisungen, die zur Hauptroutine gehören. Sie müssen mit den zwei geschweiften Klammern »{« und »}« geklammert werden. Zwischen diesen Klammern steht also das Hauptprogramm. In unserem Fall besteht es nur aus »printf("Mein erstes C-Programm/n");«. »printf« gibt, wie schon zuvor erwähnt, einen Text aus. Der Text steht anschließend in Klammern und von Hochkommata eingegrenzt. Ein Sonderfall ist noch mit eingebaut: »/n« bewirkt einen Zeilenvorschub, was nötig ist, da »printf« keinen automatischen Zeilenvorschub bewirkt. Neben »/n« gibt es unter anderem noch »/0«, was den ASCII-Code Null darstellt. Nun sieht unser Programm also folgendermaßen aus:

```
main()
{
printf("Mein erstes C-Programm/n");
}
```

Betätigen Sie nun die »ESC«-Taste und anschließend »X« und »RETURN«, um das Programm unter dem Namen »test.c« zu speichern. Um das Programm nun starten zu können, muß es zuerst mit dem Compiler in Maschinensprache übersetzt werden. Dies kann mit dem oben angegebenen Batch-File geschehen:

```
1 > EXECUTE comp test
```

Nach einer Weile ist der Kompilier-Vorgang beendet und es erscheint wieder das »1›«-Prompt. Nun kann das Programm mittels der Eingabe von »test« mit anschließendem »RETURN« gestartet werden. Das Ergebnis ist zwar nicht aufregend, aber es zeigt doch die Vorgehensweise beim Erstellen eines C-Programms.
Nun wollen wir einen Schritt weitergehen, wir wollen weitere Funktionen verwenden, die sich in einer Include-Datei auf der Diskette befinden. Diese Funktionen sind »getchar« und »putchar«, die sich in der Datei »stdio.h« befinden. Zusätzlich deklarieren wir zwei Variablen. Die Variable »global« kann in allen Routinen verwendet werden. »lokal« kann nur in der Hauptroutine verwendet werden. Wäre eine weitere Routine vorhanden, so könnten in ihr nur die Variable »global«, sowie ihre eigenen lokalen Variablen verwendet werden.

»getchar« und »putchar« haben im Prinzip die gleiche Funktion wie »scanf« und »printf«, nur daß sie für einzelne Zeichen ausgelegt sind. Um das Programm erstellen zu können, müssen Sie als erstes in den Editor mit »ed test2.c«. Das Programm sieht dann folgendermaßen aus:

```
#include ‹stdio.h›

char global;

main()
sz

  char lokal;

  lokal = getchar();
  global = lokal;
  putchar(global);
sz
```

Auch dieses Programm muß nach dem Speichern kompiliert werden. Dies geschieht mit »EXECUTE comp test2«.
An dieser Stelle wollen wir auf die möglichen Datentypen eingehen, mit denen Variablen deklariert werden können:

Datentyp	Wertebereich		Speicherlänge
int	–32 768	bis 32 767	2 BYTE
long int	–2*10 hoch 9	bis 2*10 hoch 9	4 BYTE
unsigned int	0	bis 65 535	2 BYTE
char	0	bis 255 (ASCII)	1 BYTE
FLOAT	±10 hoch –37	bis ±10 hoch 38	4 BYTE
DOUBLE	±10 hoch –307	bis ± 10 hoch 308	8 BYTE
BYTE	–128	bis 255	1 BYTE
WORD	–32 768	bis 32 767	2 BYTE
LONG	–2.15*10hoch9	bis 2.15*10 hoch 9	4 BYTE

UBYTE	0	bis 255	1 BYTE
UWORD	0	bis 65 535	2 BYTE
ULONG	0	bis 4.3*10 hoch 9	4 BYTE

Nun wollen wir noch ein Programm schreiben, das Unterroutinen verwendet. Solchen Routinen können Parameter übergeben werden, sie können aber auch Werte zurückgeben. Hier das Programm:

```
#include <stdio.h>

char eingabe;

routine(wert)                /* Routinenkopf mit Parameter */
char wert;                   /* Datentyp des Parameters */
{
  putchar(wert);
  eingabe = getchar();
  return(eingabe);
}
main()                       /* Hauptroutine */
{
  eingabe = getchar();
  eingabe = routine(eingabe); /* Routine aufrufen */
  putchar(eingabe);
}
```

Nun wissen Sie über die Struktur von C-Programmen Bescheid. In den nachfolgenden Teilen gehen wir näher auf die Programmierung ein.

Datentyp-Umwandlungen

»C« bietet die Möglichkeit, Daten innerhalb des Programms auf einfache Weise in andere Datentypen zu wandeln. Dazu braucht nur vor den zu wandelnden Wert, bzw. vor die Variable, in runden Klammern der Datentyp gesetzt werden, in den gewandelt werden soll. Wenn beispielsweise eine Integerzahl in einer Integervariablen gespeichert ist, aber einer FLOAT-Variablen zugewiesen werden soll, so geschieht das folgendermaßen:

floatvar = (FLOAT) intvar;

Dies gilt auch für Structures, die später erläutert werden. Als Besonderheit gilt an dieser Stelle, daß in solchen Fällen das Wort »struct« noch davor gesetzt werden muß. Beispiel:

```
struct testl /* erste Structure deklarieren */
{
  FLOAT float;
  int int;
};
```

```
struct test2 /* zweite Structure deklarieren */
{
  FLOAT float;
  int int;
};
main()
{
  struct test1 var1; /* Variablen vom Typ test1 */
  struct test2 var2; /* und test2 deklarieren */
  .
  .
  var2 = (struct test2) var1; /* Zuweisen und umwandeln, */
  .                           /* da nicht vom gleichen Typ */
  .
}
```

Zeiger

Ein sehr wichtiges Thema sind die Zeiger. Sie gibt es zwar auch in Pascal und Modula, doch ist ihre Verwendung in »C« besonders flexibel.
Ein Zeiger, auch Pointer oder Ptr genannt, ist eine Adreßvariable. Die Adresse, die sie enthält, ist das erste Byte einer Variablen. Man sagt auch, der Zeiger zeigt auf die Variable. Deklarieren kann man einen Zeiger folgendermaßen:

```
FLOAT *flt;
```

Wir haben also einen Zeiger auf eine Float-Variable deklariert. Durch den Stern »*« wird »flt« zum Zeiger. Es muß aber beachtet werden, daß durch diese Deklaration nur Speicherplatz für den Zeiger, nicht aber für die Variable bereitgestellt wird.
Zu den Zeigern gehört auch der Adreßoperator »&«. Er ermittelt die Adresse einer Variablen. Das bedeutet, wenn man einen Zeiger auf eine bestimmte Variable setzen will, so geht man folgendermaßen vor:

```
FLOAT *flt; /* Zeiger deklarieren */
FLOAT var; /* Variable deklarieren */

flt = &var; /* Zeiger auf Variable setzen */
```

Durch den Adreßoperator kann man also die Adresse einer Variablen ermitteln. Umgekehrt kann durch den Stern »*« auf den Speicherbereich zugegriffen werden, auf den *flt zeigt:

```
FLOAT *flt; /* Zeiger deklarieren */
FLOAT var; /* Variable deklarieren */

*flt = var; /* Variablenwert in Bereich kopieren,
  auf flt zeigen */
```

Bedingungen

In der Sprache »C« sind verschiedene Möglichkeiten vorhanden, zu testen, ob eine Bedingung wahr oder falsch ist.

Die erste Möglichkeit ist die Verwendung des »if«-Befehls. Er hat folgende Syntax:

```
if(BEDINGUNG)
  DANN;
else
  ANSONSTEN;
```

Alle kleingeschriebenen Worte sind in dieser Form anzugeben. Auf den »else«-Zweig kann verzichtet werden. »DANN« gibt den Befehl, bzw. die Befehle an, die ausgeführt werden sollen, wenn die Bedingung wahr ist. Wenn nur ein Befehl ausgeführt werden soll, so wird dieser normal angegeben:

```
if(x == y)
  printf("x ist gleich y");
```

Sollen mehrere Befehle ausgeführt werden, so müssen diese geklammert werden:

```
if(x == y)
{
  printf("x ist gleich y, /n");
  printf("also ist y auch gleich x");
};
```

Das Gleiche gilt auch für den Else-Zweig.

Die Bedingung besteht immer aus dem Vergleich zweier Werte miteinander. Folgende Vergleichsoperatoren stehen zur Verfügung:

>	–	Größer als
>=	–	Größer als oder gleich
<	–	Kleiner als
<=	–	Kleiner als oder gleich
==	–	Gleich
!=	–	Nicht gleich

Wenn x gleich 4 und y gleich 7 ist, dann ist also die Bedingung (x != y) wahr, da x ungleich y ist.

Eine weitere Möglichkeit, einen Programmteil nur unter bestimmten Bedingungen ablaufen zu lassen, stellt der Befehl »switch« dar. Er hat folgende Syntax:

```
switch(AUSDR)
{
  case AUSDR1 : DANN;
  case AUSDR2 : DANN;
  .
  .
  usw.
};
```

AUSDR ist ein Wert, der mit den Ausdrücken nach den »case«-Marken verglichen wird. Sind dann beide Werte gleich, so wird der Befehl, bzw. werden die Befehle nach dem Doppelpunkt ausgeführt.

Schleifen

Wiederholungen innerhalb eines Programms nennt man Schleifen. »C« kennt verschiedene Arten von Schleifen. Der erste Typ ist die Zählschleife:

```
for(INIT; BED; INC)
  BEFEHLE;
```

Bei INIT muß die Schleifenvariable, die für die Zählschleife benötigt wird, auf den Anfangswert gesetzt werden. Diese Schleifenvariable muß von einem ganzzahligen Typ sein.
BED stellt die Abbruchbedingung der Schleife dar. Wenn diese Bedingung nicht mehr erfüllt ist, wird die Schleife abgebrochen. Welche Bedingungen möglich sind, können Sie aus Kapitel 1.3 ersehen.
Da die Schleifenvariable nicht automatisch erhöht oder erniedrigt wird, müssen Sie selbst diese Aufgabe übernehmen. Dies geschieht bei INC.

Als Beispiel führen wir nun eine Zählschleife an, die von 0 bis 1000 zählt:

```
int zaehler;

for(zaehler = 0; zaehler <= 1000; zaehler++)
  printf("n%d",zaehler);
```

»zaehler ++« hat die gleiche Bedeutung wie »zaehler = zaehler+1«.

Eine weitere Möglichkeit, Schleifen zu bilden, ist die While-Schleife. Sie wird solange durchlaufen, bis die Bedingung nicht mehr gilt. Ihre Syntax:

```
while(BEDINGUNG)
  BEFEHL;
```

Die Bedingung entspricht der der If-Anweisung.

Ähnlich wie die While-Schleife funktioniert die »do..while«-Schleife. Die Besonderheit liegt darin, daß das Abbruchkriterium erst nach einmaligem Durchlaufen der Befehle innerhalb der Schleife geprüft wird:

```
do
  BEFEHL;
while(BEDINGUNG);
```

Auch in diesem Fall müssen die Befehle innerhalb der Schleife geklammert werden, wenn die Schleife aus mehr als einem Befehl besteht.

Strukturen

Von grundlegender Bedeutung sind die Strukturen, auch Structures oder Listen genannt. In ihnen können verschiedene Variablen unter einem Oberbegriff zusammengefaßt werden. Eine Structure, wie wir sie nachfolgend nennen wollen, wird von dem Wort »struct« eingeleitet. Ihm folgt der Name, den die Structure haben soll, gefolgt von den Einträgen, die in der Structure zusammengefaßt werden sollen. Solche Einträge können auch weitere Structures sein. Ein Beispiel:

```
struct Bsp
{
  FLOAT    flt;
  int        i;
  struct Test demostruct;
  struct Bsp  *ptr;
};
```

Die Structure Bsp besteht also aus den Einträgen flt, i, demostruct und dem Zeiger ptr, der auf eine weitere Structure vom Typ Bsp zeigt.
Bsp stellt nun einen neuen Datentyp dar. Um ihn verwenden zu können, muß eine Variable von diesem Typ deklariert werden. Dies geschieht folgendermaßen:

```
struct Bsp beispiel;
```

Will man nun auf die einzelnen Einträge zugreifen, so geschieht das folgendermaßen:
Auf flt, i und demostruct kann sehr einfach zugegriffen werden:

```
beispiel.flt = 2.45;
beispiel.i = 3;
```

also einfach durch einen Punkt zwischen dem Variablennamen und dem Eintrag, auf den zugegriffen werden soll.
Um auf ptr zugreifen zu können, muß ein »Pfeil« zwischengesetzt werden:

```
beispiel->ptr = ....;
```

Man muß nicht unbedingt eine Variable vom diesem Structuretyp deklarieren. Es kann auch ein Zeiger darauf verwendet werden, für den die gleichen Bedingungen gelten, wie für die Zeiger in Kapitel 1.2. Ein solcher Zeiger wird folgendermaßen deklariert:

```
struct Bsp *beispiel;
```

Die Bibliotheken

Die Hardware des Amiga ist von einer Vielzahl von leistungsstarken Software-Modulen umgeben. Durch diesen modularen Aufbau bieten sich ungeahnte Möglichkeiten. Das System wird somit flexibler und leistungsstärker. Module können hinzugefügt oder, falls notwendig, verändert werden.

Einen Teil dieser Amiga-System-Software-Module bilden die Libraries, zu deutsch (Software-)Bibliotheken. Das Amiga-System enthält bisher 16 Module. Hier eine Übersicht :

clist.lib: Enthält einige nützliche Routinen, die den Umgang und die Anwendung von Zeichenketten vereinfachen.

console.lib: Dieses Library enthält Programme für den Umgang mit der Tastatur, der sogenannten Console.

diskfont.lib: Das diskfont.lib ermöglicht die Verwendung der verschiedenen Schrifttypen,die sich auf der Workbench-Diskette befinden.

dos.lib: Durch dieses Library wird dem Amiga unter anderem der Zugriff auf die Diskette ermöglicht. Der Zugriff auf die Diskette ist dank dieses Libraries fast so einfach, wie von der Benutzerschnittstelle CLI aus.

exec.lib: Dieses Library bildet den System-Kern des Amiga. Dieser Kern entscheidet z.B. welche Tasks zum Laufen kommen (in der Computersprache bezeichnet man dies mit Scheduling) oder wieviel Speicherplatz für ein Programm bereitgestellt werden muß.

graphic.lib: Ohne Grafik geht heutzutage nichts mehr. Das graphic.lib ist ein sehr leistungsstarkes und umfangreiches Bibliotheksmodul, dessen Funktionen unter anderem durch den direkten Zugriff auf den Blitter und Copper phantastische Geschwindigkeitenin punkto Grafik sowie Animation ermöglichen.

icon.lib: Hier sind verschiedene, durchaus nützliche Utilities für den Umgang mit den, von der Workbench her bekannten Icons enthalten. Es ist ebenfalls eines der wenigen Libraries, die sich auf der Workbench-Disk befinden.

info.lib: Dieses Library wird dazu verwendet, um Information über Dateien, Datei- Verzeichnisse oder ganze Disketten zu bekommen.
Es wird kaum verwendet und befindet sich auf der Workbench-Diskette.

intuition.lib: Das intuition.lib ist eines der wichtigsten Libraries des Amiga. Ohne dieses Library wäre keine Bedienung mit der Maus oder die einfache Handhabung von Menüs denkbar.

janus.lib: Dies ist bisher das letzte Bibliotheks-Modul, das dem Amiga beigefügt wurde. Es befindet sich ebenfalls auf der Diskette und wird zur Steuerung der SideCar-Hardware benötigt.

layers.lib: In diesem Bibliotheks-Modul sind Routinen enthalten, die dem Anwender beispielsweise das Handling von überlappenden Display-Elementen erleichtern.

mathffp.lib: Mit diesem sogenannten FFP-Basic-Mathematik-Library können einfache mathematische Aufgaben, wie z.B. die Multiplikation oder Division, gelöst werden.

mathieeedoubbas.lib: Dies ist das erweiterte FFP-Basic-Mathematik-Library. Es befindet sich auf der Workbench-Diskette und enthält eine Vielzahl von mathematischen Funktionen, die Zahlen im IEEE-Standard mit doppelter Genauigkeit verarbeiten.

mathtrans.lib: Für schwierigere mathematische Aufgaben, wo Funktionen wie arcsin, arccos usw. Verwendung finden, enthält dieses Bibliotheks-Modul genügend Befehle. Da diese Funktionen nicht ständig verwendet werden, ist dieses Library auf der Workbench-Disk enthalten.

timer.lib: Wenn Sie zeitlich im Bilde sein wollen, bietet sich die Verwendung dieses Libraries an. Leider kann beim Amiga 1000 nur die Software-Uhr angesprochen werden. Bei den Versionen 500 und 2000 ist diese Uhr jedoch batteriegepuffert.

translator.lib: Das translator.lib hat die Aufgabe, Sätze, die in englisch verfaßt sind, für die Sprachausgabe vorzubereiten. Es findet kaum Verwendung und ist deshalb auf der Systemdisk enthalten.

Je nach Art des Programms, das der Programmierer entwickeln will, muß er selbständig entscheiden, welche Bibliotheks-Module er benötigt. Sicherlich werden Sie nun denken, je mehr Libraries verwendet werden, desto besser wird das Programm. Im Gegenteil! Für sehr gute Programme reichen schon 2 bis 3 Libraries aus.

Beim Umgang mit den Libraries müssen bestimmte Regeln eingehalten werden, damit die jeweiligen Funktionen ansprechbar sind. So hat es z.B.keinen Zweck, Funktionen eines Libraries aufzurufen, wenn das jeweilige Library nicht geöffnet wurde.

Bevor jedoch das jeweilige Library geöffnet wird, muß der »Basis« des Libraries ein Zeiger zugewiesen werden, hier am Beispiel des Intuition-Library demonstriert, der von OpenLibrary zurückgegeben wird:

```
IntuitionBase = (struct IntuitionBase *)
  OpenLibrary("intuition.library",0);
```

Anschließend enthält IntuitionBase die Einsprungadresse der Intuition-Library. Enthält diese Variable den Wert »NULL«, war es nicht möglich, das Library zu öffnen.
Ist der Wertungleich »NULL«, verlief alles normal und das Library konnte geöffnet werden.
Nachdem Sie ein Library geöffnet haben, muß es natürlich auch wieder geschlossen werden:

```
CloseLibrary(IntuitionBase);
```

schließt das jeweilige Library.
Hier zum besseren Verständnis nochmals ein Beispiel :

```
/* Öffnen und Schließen eines Bibliotheksmodules */
.
.
struct GfxBase *GfxBase; /* Zeiger für die Einsprungadresse */
main()
{
GfxBase = (struct GfxBase *)
  OpenLibrary("graphics.library",0); /* Library öffnen */
if (GfxBase == NULL)
}
  printf("Öffnen des graphics.library nicht möglich !/n");
  exit(FALSE);
{
/*
*/    Hier das jeweilige Programm eintragen
/* Zum Schluß Bibliotheks-Module schließen */
CloseLibrary(GfxBase);
}
```

IntuitionBase und GfxBase dürfen mit

```
struct IntuitionBase *IntuitionBase;
```

bzw.

```
struct GfxBase *GfxBase;
```

deklariert werden, da sie die Einsprungadressen der Intuition- und der Graphics-Library darstellen. Für alle anderen Libraries gilt folgende Deklaration:

Beispiel Diskfont-Library

```
DiskfontBase = OpenLibrary("diskfont.library",0);
if (DiskfontBase == NULL)
{
  printf("Öffnen des diskfont.library nicht möglich !/n");
  exit(FALSE);
}
/*
*/      Hier das jeweilige Programm eintragen
/* Zum Schluß Bibliotheks-Module schließen */
CloseLibrary(DiskfontBase);
}
```

Die Devices

Neben den Libraries, die für den Programmierer Erleichterungen und für das System eine große Flexibilität darstellen, steht dem Programmierer weiteres großes Hilfsmittel zur Verfügung: Die Devices.
Devices, zu deutsch Vorrichtungen, sind die Bindeglieder zwischen der (externen) Hardware und der Software des Amiga. Durch sie können Daten zur Hardware gesendet oder von ihr empfangen werden. Somit ist, z.B. durch das Verändern von Parametern der Trackdisk-Device, das Lesen von fremden Diskettenformaten, wie IBM-Format möglich. Der Amiga enthält 17 verschiedene Devices, die sich um die Vorrichtungen wie Tastatur, der Seriell- und Parallelschnittstellen und einiges mehr kümmern. Nicht alle werden ständig benötigt, sondern befinden sich im »Devs«-Directory auf der Workbench-Disk.

Die Devices des Amiga im Überblick:

audio.device: Mit ihr wird der »Sound« des Amiga gesteuert. Je nach Belieben richtet sie die 4 Audio-Kanäle des Amiga ein, bestimmt die Amplitute des Tons und vieles mehr.

bootblock.device: Testet, ob es sich um eine Kickstart- oder um eine DOS-Diskette handelt. Bei den neuen Amiga's ist diese Vorrichtung weggefallen, da bei ihnen keine Kickstartdiskette mehr erforderlich ist.

clipboard.device: Wird benötigt, um Daten zwischen zwei Anwendungen zu transferieren. Da dies nicht häufig vorkommt, befindet sich diese Device auf der Workbench-Disk.

console.device: Regelt die Ein- und Ausgabe des Systems über die Tastatur und den Bildschirm.

gameport.device: Gameport.device übernimmt die Steuerung der Ein- und Ausgabe über die GamePorts 1 und 2.

input.device: Diese Device regelt die gesamte Ein- und Ausgabe des Amiga. Es ist eine Kombination aus timer-, gameport- und keyboard.device.

inputevent.device: Inputevent.device erfaßt die Ereigniseingaben, wie z.B. Gadgets.

jdisk.device: Dies ist die neuste Device des Amiga. Sie übernimmt die Steuerung der Harddisk des Amiga, die sich auf der IBM-PC-kompatiblen Seite des Amiga2000 oder im SideCar befindet. Da sie sehr neu ist, befindet sie sich ebenfalls auf der Workbench-Disk.

keyboard.device: Hiermit wird der Zugriff auf die Tastatur des Amiga gesteuert.

keymap.device: Damit kann die Belegung der Tastatur verändert werden.

narrator.device: Narrator.device ist für die Steuerung der Sprachausgabe notwendig. Da sie nicht ständig benötigt wird, befindet sie sich auf der Workbench-Disk.

parallel.device: Hiermit kann der Parallelport gesteuert werden. Diese Device befindet sich ebenfalls auf der Workbench-Disk.

printer.device: Diese Device dient zur Kommando-Steuerung des Druckers, um z.B. einen Wagenvorlauf des Druckers zu bewirken. Printer.device befindet sich ebenfalls auf der Workbench-Disk.

prtbase.device: Prtbase.device übernimmt die Datendefinition der printer.-device.

serial.device: Diese dient zur Deklaration des seriellen Ports des Amiga. Sie befindet sich ebenfalls auf der Workbench-Disk.

timer.device: Mittels timer.device kann auf die Systemzeit zugegriffen werden.

trackdisk.device: Diese Device kontrolliert die Floppies des Amiga. Sie übernimmt Funktionen, wie das Lesen und Schreiben von Daten und einiges mehr.

Um mit einer Device arbeiten zu können, muß ein Port angelegt werden. Dies geschieht mit

```
printerPort = CreatePort("printer.port",0);
```

wobei printerPort zurückgegeben wird.
Danach muß die Device geöffnet werdem, in diesem Fall "printer.device":

```
fehler = OpenDevice("printer.device",0,&request,0);
```

Wenn Fehler gleich ungleich 0 ist, konnte die Device nicht geöffnet werden. &request ist der Pointer auf eine Structure der jeweiligen Device, die bestimmte »Routinen« wie z.B. das Drucken eines Screens enthalten oder auf die allgemeine Ein- und Ausgabe-Structure von EXEC.
Nachdem die Device und der Port geöffnet sind, kann die benötigte Ein- und Ausgabe-Structure initialisiert werden. Nach der Initialisierung wird die jeweilige »Funktion«, wie z.B. das Drucken eines Textes, mit

```
DoIO(&request);
```

gestartet.

&request ist der Pointer auf die Ein- und Ausgabe-Structure der jeweiligen »Funktion«.

Nachdem die Ein- und Ausgabe beendet ist, muß der Port und die Device wieder geschlossen werden. Dies kann mit

```
DeletePort(printerPort);
CloseDevice(&prefrequest);
```

erledigt werden.

Der Seka-Assembler

Zur Entwicklung der Assembler-Programme haben wir den Seka-Assembler verwendet. An dieser Stelle wollen wir nun auf die Bedienung dieses Assemblers eingehen.

Ist der Assembler gestartet worden, fragt er nach dem Speicherplatz, der ihm zur Verfügung gestellt werden soll. Anschließend befindet man sich in der Befehlsebene. Will man nun ein Programm eingeben, so kann dies durch Drücken der Escape-Taste geschehen. Durch sie gelangt man in den Editor, der zwar etwas umständlich und einfach ist, dem Zweck aber durchaus genügt. Durch nochmaliges Betätigen der Escape-Taste gelangt man wieder in die Befehlsebene.

Zum Editor ist lediglich noch zu sagen, daß beliebig mit dem Cursor »herumgefahren« werden kann und im Text manipuliert werden kann. Äußerst positiv fiel uns auf, daß TAB's als solche unterstützt wurden und nicht nur ein Einfügen von Leerzeichen zur Folge hatten. In der Befehlsebene stehen folgende Befehle zur Verfügung:

T	Cursor in die oberste Zeile setzen
T n	Setzt den Cursor in Zeile n
B	Setzt den Cursor an den Schluß des Textes
U	Cursor eine Zeile hochsetzen
U n	Cursor um n Zeilen hochsetzen

D	Cursor eine Zeile heruntersetzen
D n	Cursor um n Zeilen heruntersetzen
Z	Löscht die aktuelle Zeile
Z n	Löscht n Zeilen ab der aktuellen Zeile
E	Zeileneditor für aktuelle Zeile
E n	Zeileneditor für Zeile n
Ftext	Sucht ab der aktuellen Zeile nach dem eingegebenen Text. Dabei wird Groß-/Kleinschreibung und Leerzeichen berücksichtigt
F	Sucht weiter nach dem obigen Text
I	Fügt eine Zeile an der aktuellen Zeile ein
KS	Löscht den Quelltext
O	Macht die Löschung wieder rückgängig
P	Druckt die aktuelle Zeile aus
P n	Druckt n Zeilen ab der aktuellen Zeile aus
V	Gibt das Inhaltsverzeichnis aus
KF	Löscht eine Datei von Diskette
R	Liest ein Sourcefile von Diskette
RO	Liest ein Objectfile (assembliertes File) von Diskette
RI	Liest eine Datei in den Speicher. Es wird nach der Anfangs- und der Endadresse des Bereiches gefragt, in den geladen werden soll
RX	Hat die gleiche Funktion wie RI, nur das von der seriellen Schnittstelle gelesen wird
RL	Liest ein mit Seka erstelltes Linkfile in den Speicher
W	Schreibt den Sourcecode auf Diskette
WI	Speichert einen Speicherbereich auf Diskette ab
WX	Funktion wie WI, nur das über die serielle Schnittstelle übertragen wird
WL	Schreibt ein Linkfile auf Diskette

A Assembliert den Quellcode. Folgende Optionen stehen zur Verfügung:

V Ergebnisse werden auf dem Bildschirm ausgegeben

P Ausgabe wird auf den Drucker umgeleitet

H Nach jeder Seite wird die Ausgabe gestoppt und auf einen Tastendruck gewartet

O Alle Branch-Befehle werden optimiert

L Es wird ein linkfähiger Programmcode erzeugt, der mit WI abgespeichert werden kann

G Starten des assemblierten Programms

Glabel Starten des assemblierten Programms ab dem Label label

Jlabel Wie Glabel, nur das der Aufruf über einen JSR-Befehl bewirkt wird.

Dies war nun eine Auflistung der Befehle, die Seka zur Verfügung stellt. Wir hoffen, daß Sie sich mit diesem Assembler anfreunden können, denn er hat große Stärken, wenn es darum geht, schnell eine Maschinenroutine auszuprobieren.

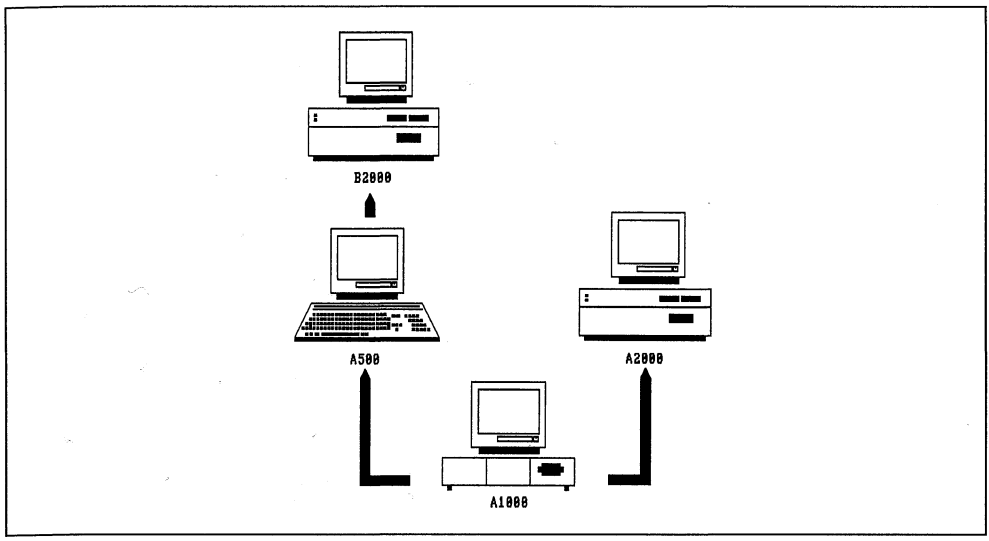
Anwendern, die größere Projekte in Assembler bearbeiten müssen, sei das DevPac von HiSoft/Markt & Technik empfohlen.

Kapitel 1

Die Amiga-Serie

Die Amiga-Serie besteht zur Zeit aus zwei grundlegend verschiedenen Amigas. Auf der einen Seite haben wir die Amiga 500 und 1000 für den »Heim«-Bereich, auf der anderen Seite den A2000 für den »Geschäftsbereich«. Beide Seiten sind untereinander kompatibel. Der große Vorteil des A2000, im Vergleich zum A500 bzw. A1000, besteht jedoch in seinem offenen Aufbau. Er kann durch Erweiterungskarten MS-DOS oder sogar UNIX-fähig oder durch Einsatz eines anderen Prozessors beschleunigt werden. Für den Einstieg in die MS-DOS-Welt mit dem A500 und A1000 besteht die Möglichkeit, eine Hardware-Emulation, das SideCar, anzuschließen. Beim A500 stößt man dabei zunächst auf ein großes Problem beim Anschließen, da der 86-Pin-Erweiterungsstecker um 180 Grad gedreht wurde.

Der technische Fortschritt hat auch bei den Amiga's keinen Halt gemacht. So ist das Motherboard der neuen Generation, das des Amiga 500 und Amiga B2000, durch Einsatz der FatAgnus und des Garry stark geschrumpft. Somit konnten auch Kosten und die damit verbundenen Preise der Amigas gesenkt werden. Auch das SideCar wurde in eine kleine Platine umgesetzt. Die Produktion des Amiga 1000 und des Original-SideCars wurden somit unrentabel, weshalb sie eingestellt wurde.



Die Amiga-Familie

Es steht also eine reichhaltige Amiga-Palette zur Verfügung, die bei uns auf dem Markt erhältlich ist, bzw. war. Hier ein »geschichtlicher« Überblick über die Amiga-volution:

Ende 85: Erste, aus den USA importierte Amiga 1000 werden verkauft. Sie sind daran erkennbar, daß sie ein zusätzliches Netzteil benötigen, eine NTSC-Version der Agnus besitzen und ein anderer Monitor mitgeliefert wurde.

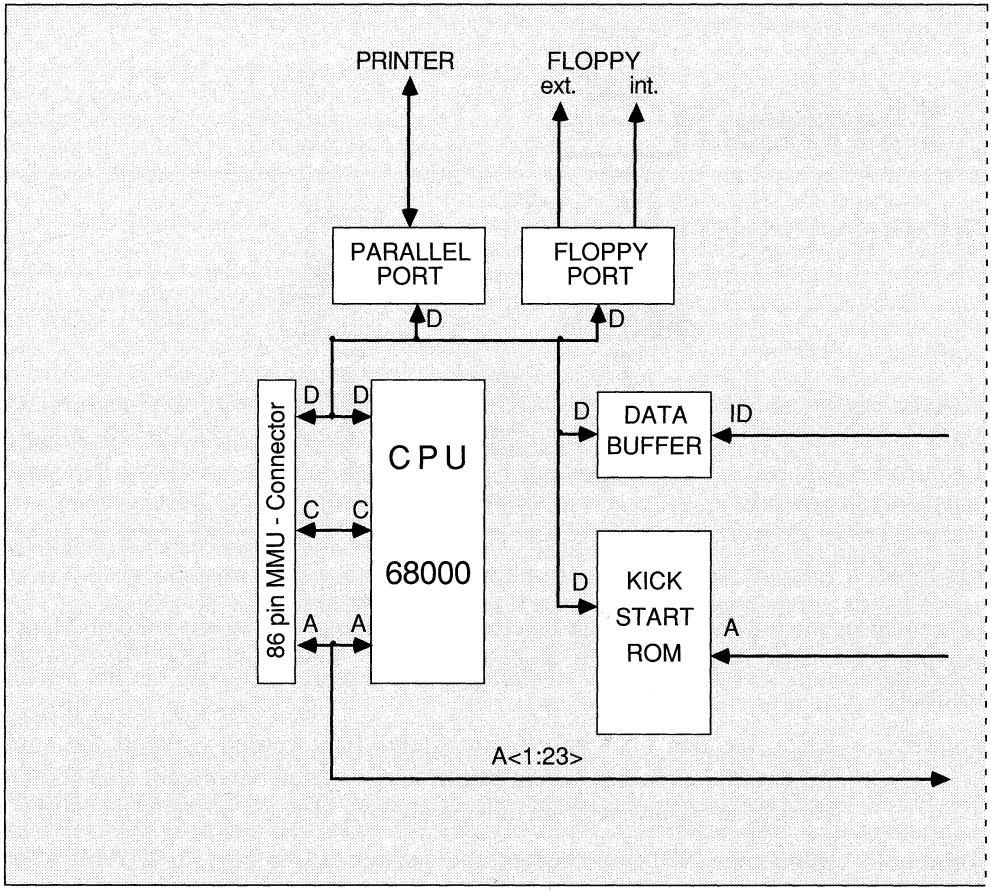
Anfang 86: Der Amiga 1000 als eingedeutschte Version mit Netzteil für 220 Volt, aber noch mit NTSC-Agnus.

Mitte 86: Nun erhält der Amiga 1000 auch eine PAL-Agnus, besitzt aber noch einen NTSC-Composite-Video-Ausgang.

Ende 86: Der Amiga 1000 verfügt nun auch über eine PAL-Version des Composite-Video-Ausgangs. Die Hucke-Pack-Platine, in die das Kickstart geladen wird, ist auf die Grundplatine verpflanzt worden. Das SideCar wird auf den Markt gebracht.

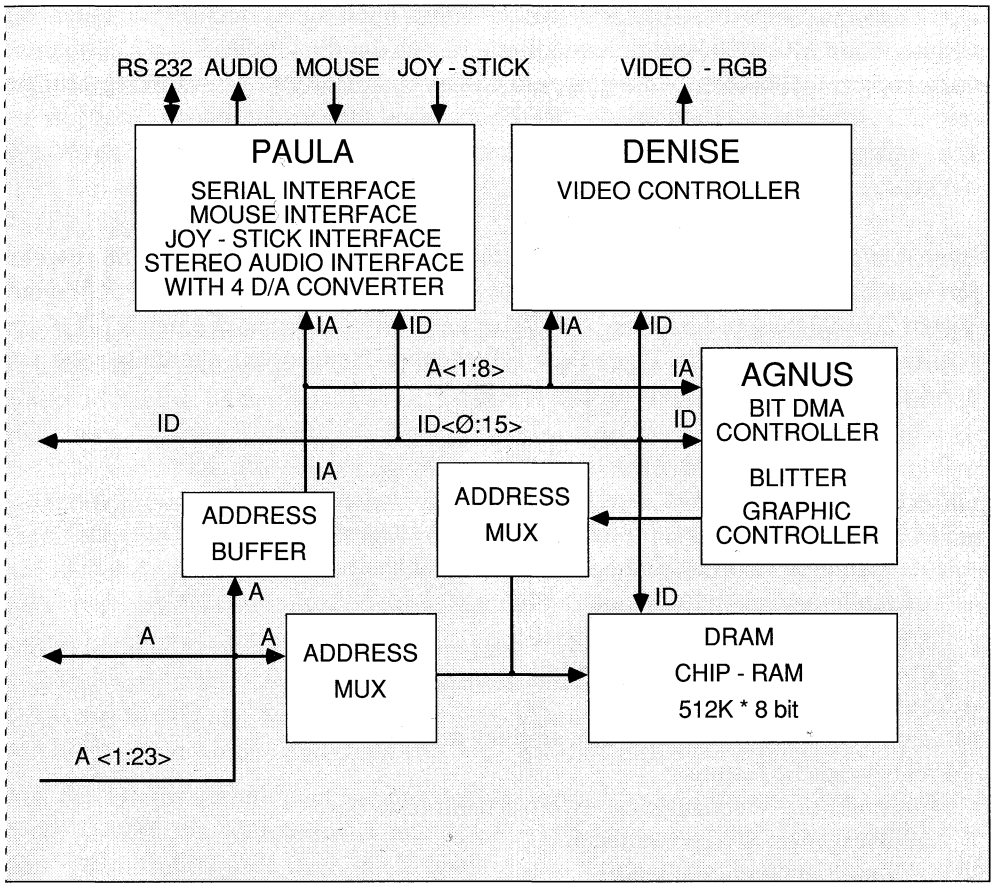
Anfang 87: Die Amiga-Offensive: Der A1000 erhält Nachwuchs in Form des Amiga 500 und des Amiga A2000. Gleichzeitig werden die ersten Erweiterungskarten für den Amiga A2000 präsentiert: PC-Karte und RAM-Erweiterungen.

Mitte 87: Der Amiga A2000 wird runderneuert. Seine Platine wird durch Einsatz der FatAgnus und des Garry, die schon im Amiga 500 eingesetzt wurden, erheblich verkleinert. Sein neuer Name ist Amiga B2000.



Z 1.1: *Amiga 1000-Blockdiagramm (Teil 1)*

Die Frage ist nun, wie es mit der Entwicklung weitergeht. Commodore hält sich in dieser Frage eher bedeckt, auch wenn zur Zeit Gerüchte über einen Super-Amiga im Umlauf sind. Aber diese Strategie verfolgte Commodore auch vor der Vorstellung der A1000, A500 und A2000. Während sich der Amiga 1000 seinen Weg zum Renner unter den Computern bahnte, hatte Commodore schon den Amiga 2000 und Amiga 500 im Schrank stehen. Im Bereich des Amiga 500 wird sich wohl zunächst nicht sehr viel tun, was wohl auch nicht nötig ist, da der A500 sich zum 64er-Nachfolger avanciert. Im 2000er-Bereich werden demnächst einige Neuerungen auf dem Markt vorgestellt, die die Leistung des jetzigen A2000 um ein Vielfaches übertreffen werden.



***Z 1.1:** Amiga 1000-Blockdiagramm (Teil 2)*

1.1: Der Amiga 1000

Der Amiga 1000 ist der Ursprung aller Amigas. Er wurde von einer Joystick-Firma entwickelt, die Commodore aufkaufte und somit an die Rechte dieses fantastischen Computers gelangte. Er sollte neue Maßstäbe im Computerbereich, speziell im Business-Bereich, setzen. Seine fantastischen Fähigkeiten ließen ihn jedoch schon früh als einen Super-Spielcomputer erscheinen. Neben den großartigen Soundfähigkeiten, überzeugt er die Fachwelt besonders im Grafikbereich. Custom-Chips sind mit Erscheinen dieses Rechners das Schlagwort. Sie verleihen diesem Rechner ungeahnte Möglichkeiten und nehmen dem Hauptprozessor einen Großteil der Arbeit ab. Ein Blockdiagramm veranschaulicht das Zusammenwirken dieser Komponenten (Zeichnung 1). (Siehe auch Bild 5 im Farbteil).

Diese Custom-Chips im A1000 erhielten die Spitznamen Agnus, Denise und Paula. Sie entlasten den MC 68000 enorm und kümmern sich um die Grafik, Sound, DMA und noch einiges mehr. Mausbedienung, ein 16-Bit-Prozessor und Multitasking gehören seit der Markteinführung dieses Rechners zum Standard. Jedoch kann bisher kein anderer Computerhersteller mit seinen Produkten diesem Computer das Wasser reichen.

Beim Amiga 1000 gehen wir in diesem Buch von der Version, die ab Mitte '86 ausgeliefert wurde, aus. Diese hat eine PAL-Agnus, konnte somit vertikal 256 bzw. 512 Punkte anstatt 200/400 Punkte darstellen, hatte aber unter Umständen noch einen NTSC Compostite-Videoausgang. Ein Piggy-Pack (Huckepack-Platine) war ebenfalls noch vorhanden. Hier seine Merkmale im Überblick:

- Prozessor MC 68000 mit 7.16 Mhz getaktet.
- 256 Kbyte internes RAM, erweiterbar bis 1 Mbyte intern.
- 256 Kbyte RAM (WOM) für System-Software (Kickstart).
- Eingebautes 3,5 Zoll Floppy, Speicherkapazität brutto 880 Kbyte.
- Anschlußmöglichkeit für 3 weitere Floppies.
- Programmierbare serielle Schnittstelle.
- Programmierbare paralle Schnittstelle.
- Mitgelieferte Maus.
- Game-Port-Anschlüsse für Maus, Joystick usw.
- Frei bewegliche Tastatur.
- Multifunktions-Videoausgang, extern synchronisierbar.
- Audioausgänge, Stereo zweikanalig.
- Vier Ton- und Geräuschkanäle.
- Erweiterungsport.
- Grafikauflösung 320 x 256, 320 x 512, 640 x 256, 640 x 512, mit speziellen Grafikmodi bis zu 4096 verschiedene Farben gleichzeitig
- 3 Custom-Chips: Agnus, Denise, Paula kontrollieren beispielsweise die DMA

Dies war sicherlich nur eine grobe Zusammenfassung der Fähigkeiten des Amiga 1000. Besonders zu erwähnen wäre noch, daß sein Betriebssystem, das *»Kickstart«*, gesondert geladen werden muß, bevor mit der Workbench das eigentliche »System« aktiviert wird. Dies hat den Vorteil, daß man nicht nur eine andere Kickstart-Software (Kickstart 1.1 oder 1.2) verwenden, sondern auch seine eigene Version schreiben kann.

Vom äußeren hebt er sich erheblich von den Designs des IBM-PCs, oder C64 ab. Intern zeigt sich der A1000 solide aufgebaut. Ist das *Piggy-Pack* abgehoben, so erkennt man die Custom-Chips in voller Pracht. In Bild 7 (Farbteil) sehen Sie das Motherboard in voller Größe.

Eine Besonderheit bei den älteren Amiga-Rechnern ist,wie schon erwähnt, das Piggy-Pack (eine Huckepack-Platine). In das RAM, das sich auf dieser Platine befindet, wird nach dem Einschalten des Rechners die Kickstart-Software geladen. Zudem enthält sie 3 wichtige PAL's (Programmierbare Logik-Chips), die für die Erzeugung der Refresh-Signale der RAM-Chips, für das Chip-Select-Signal und noch einiges mehr zuständig sind. Bei dem neueren Amiga 1000 ist diese Platine in das Motherboard integriert. Bild 28 (Farbteil) zeigt das Piggy-Pack.

1.2: Der Amiga 500

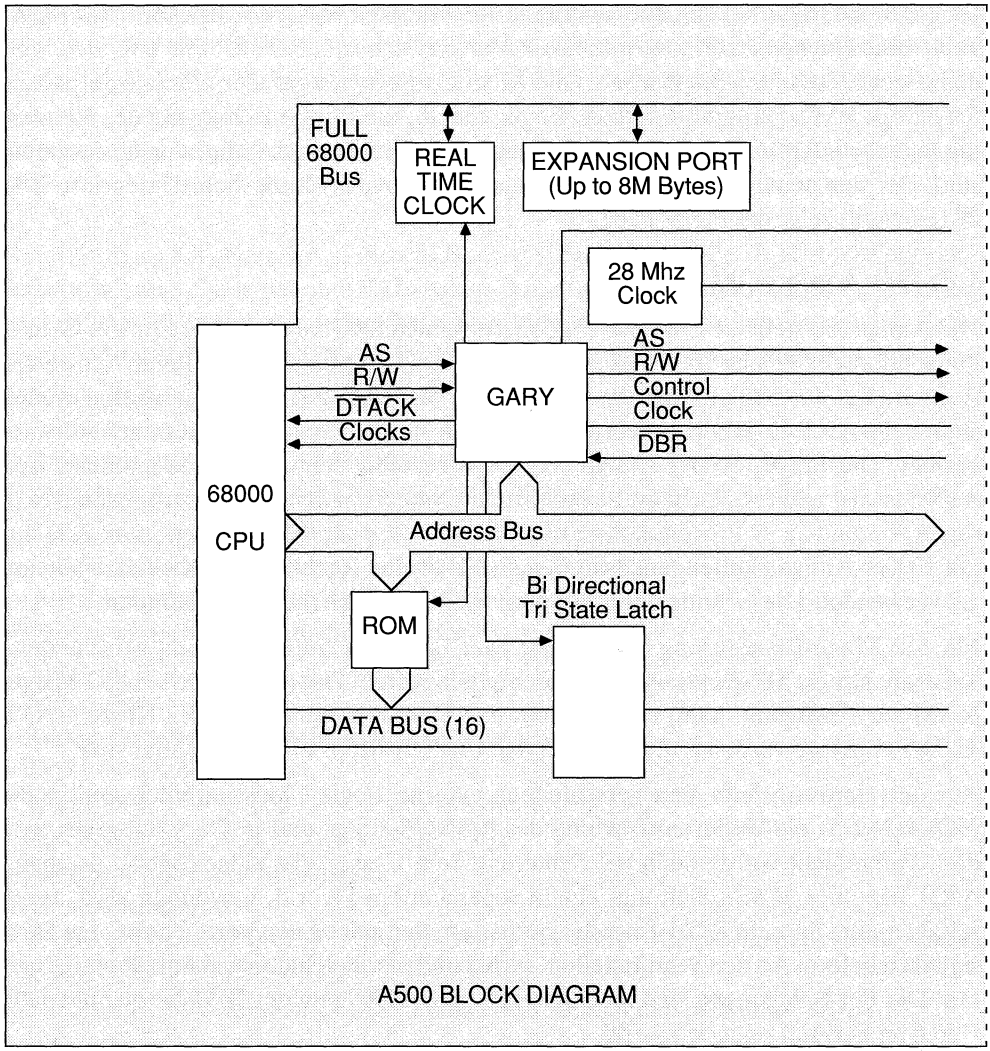
Der Amiga 500 stellt die low-cost-Version des Amiga 1000 dar. Er basiert auf der Grundkonzeption des Amiga 1000 und ist fast vollständig kompatibel zur älteren Schwester. Während bei der 1000er-Serie das Kickstart in das Piggy-Pack geladen werden mußte, steht beim A500 ein ROM zur Verfügung, das die neuste Version der Kickstart-Software enthält. Die Frage, ob der Amiga 500 softwaremäßig 100prozentig kompatibel zum A1000 ist, ist nicht so leicht zu beantworten. Ohne 512-Kbyte-Erweiterung kann man sagen, sie sind zu 99 Prozent kompatibel. Es gibt ein paar wenige Programme, die nur auf 1000er-Amiga's laufen, auch wenn man bei beiden Rechnern die Kickstart-Version 1.2 verwendet. Die Gründe dafür konnten wir selbst noch nicht herausfinden.

Mit 512-Kbyte-Erweiterung können ein paar Probleme auftauchen, da verschiedene Software nur für 512-Kbyte-Amigas geschrieben wurde. Hat jedoch ein Amiga 1 MByte RAM zur Verfügung, kann es passieren, daß Daten in den oberen 512 Kbyte *(FAST-MEM)* abgelegt werden und somit von den Custom-Chips nicht erreichbar sind.

Auf der Hardwareseite sind verschiedene externe Logik-Chips entfallen und in die *FAT-AGNUS*, die Weiterentwicklung des AGNUS-Chips und in Gary integriert worden. Vereinfacht wurde auch die Schaltung zum Composite-Video-Signal, das beim A500 aus einer Mischschaltung, einem sogenanntem Hybrid, gewonnen wird. Beim A1000 mußte hier ein RGB-Encoder herhalten, der jedoch den Vorteil hatte, ein Farbsignal zu liefern. An den Schnittstellen, seriell und parallel, hat sich ebenfalls einiges getan. Die Pin-Belegungen sind im Gegensatz zum A1000 nun der IBM-Norm angepaßt. Bild 6 im Farbteil zeigt den A 500.

Daß der Amiga 500 stark zusammengeschrumpft ist, ist auch deutlich an seinem Blockdiagramm (Z 1.2-1) zu erkennen. Viele verschiedene Komponenten wurden zusammengefaßt. Sein Aufbau ist somit noch einfacher und kostensparender geworden.

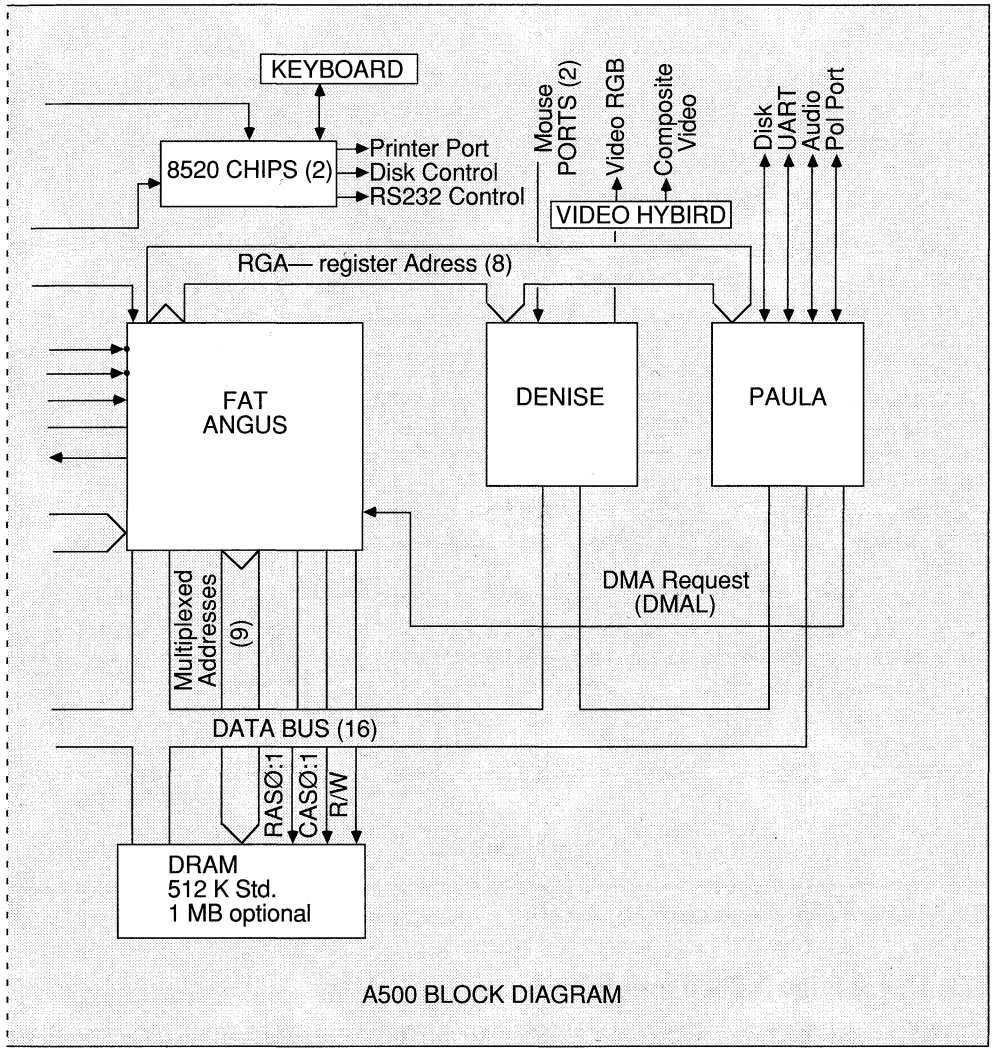
Äußerlich hat sich dieser Amiga sehr verändert. Die portable Tastatur wurde in das Gehäuse eingebaut, das Floppy, welches beim A1000 von vorne zugängig war, wurde auf die rechte Seite verpflanzt, was den Nachteil hat, daß der 86-Pin-Erweiterungsport auf die linke Seite weichen mußte. Eine 100prozentige Hardwarekompatibilität zum A1000 ist somit nicht mehr gegeben, denn Geräte wie das SideCar können ohne zusätzlichen Adapter nicht angeschlossen werden, da der Erweiterungsport im Vergleich zum A1000 um 180 Grad gedreht wurde.



***Z 1.2-1:** Amiga 500-Blockdiagramm (Teil 1)*

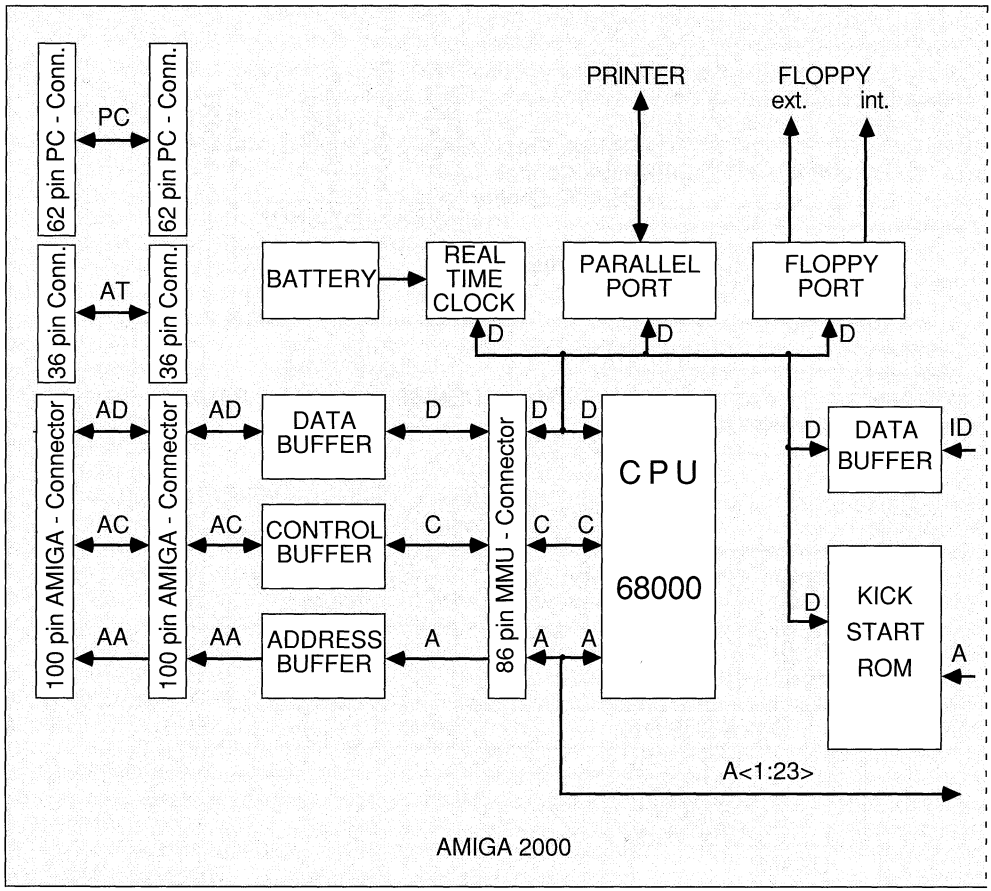
Hier die Merkmale der Amiga 500 im Überblick:

- Prozessor MC 68000 mit 7.16 Mhz getaktet.
- 512 Kbyte internes RAM, erweiterbar bis 1 MByte intern.
- 256 Kbyte ROM.
- Eingebautes 3,5 Zoll Floppy, Speicherkapazität brutto 880 Kbyte.
- Anschlußmöglichkeit für 3 weitere Floppies.
- Programmierbare serielle Schnittstelle.
- Programmierbare paralle Schnittstelle.
- Mitgelieferte Maus.



Z 1.2-1: *Amiga 500-Blockdiagramm (Teil 2)*

- Game-Port-Anschlüsse für Maus, Joystick usw.
- Multifunktions-Videoausgang, extern synchronisierbar.
- Audioausgänge, Stereo zweikanalig.
- Vier Ton- und Geräuschkanäle.
- Erweiterungsport im Vergleich zum A1000 um 180 Grad gedreht.
- Grafikauflösung 320 x 256, 320 x 512, 640 x 256, 640 x 512, mit speziellen Grafikmodi mit 4096 verschiedenen Farben gleichzeitig.
- 4 Custom-Chips Agnus, Denise, Paula, Gary sorgen beispielsweise für die DMA-Kontrolle.

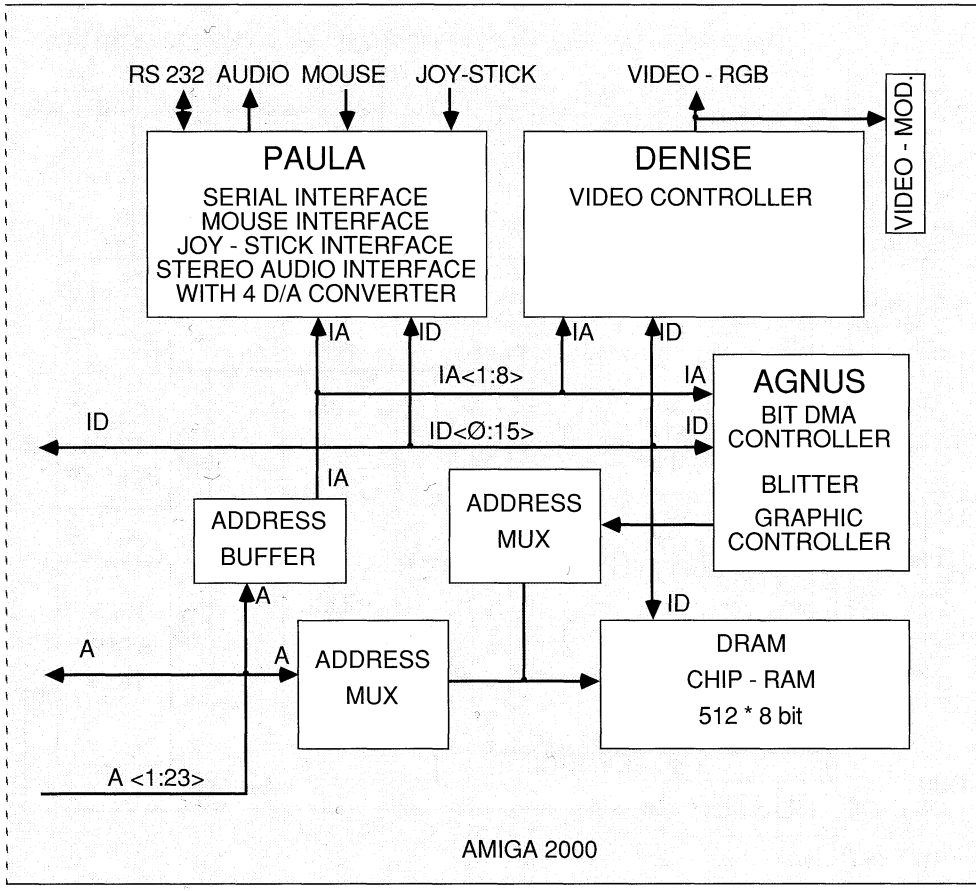


Z 1.3-1: *Das A2000-Blockdiagramm (Teil 1)*

1.3: Der Amiga A2000 und B2000

Die Weiterentwicklung des A1000 ging nicht nur in Richtung des low-cost-Bereichs A500, sondern auch in den professionellen Bereich. Für diesen Bereich wurde der neue Amiga 2000 sehr offen gestaltet. Nach altem Apple-Konzept wurde die Grundplatine mit verschiedenen Slots bestückt, die kompatibel zum 100-Pin-*Zorro-Bus*, AT- bzw. PC-Slot und zum 86-Pin-Amiga-Stecker sind. Eine Vielzahl von Steckkarten, wie PC/XT-Emulatur / AT-Emulator-Card und Coprozessor-Karte, lassen den Amiga für neue Anwendungsgebiete interessant werden. Bild 14 im Farbteil zeigt das Slot-Prinzip des 2000ers.

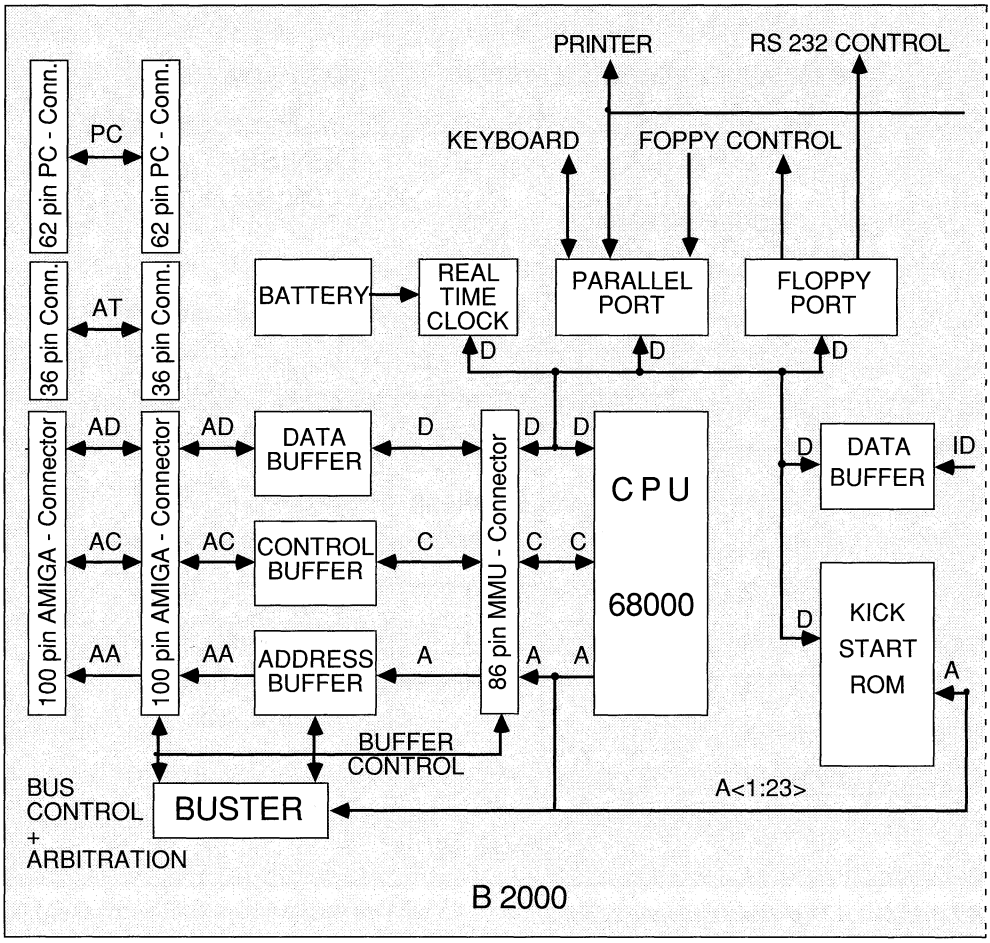
Der erste A2000 entstand aus dem A1000. Es wurde, grob beschrieben, ein A1000 Motherboard mit einer Buskontroll-Logik versehen, die für die Verwaltung der Amiga-Slots zuständig ist. Für PC/AT-Anwendungen wurden zu den 5-AMIGA-Slots noch 4



Z 1.3.1: *Das A2000-Blockdiagramm (Teil 2)*

PC/AT-kompatible Slots hinzugefügt, die der 2000er Serie den Weg in den Business-Bereich ebnen sollen. Zudem enthält der 2000er einen *Video-Slot* zum internen Einstecken eines Genlock-Interfaces oder eines UHF-Modulators. Hinzugefügt wurde auch eine batteriegepufferte Uhr. Ansonsten blieb die Technik gleich, was das Blockdiagramm der Zeichnung Z 1.3-1 beweist:

Mit Veränderung des A1000 zum A500 wurde auch der A2000 im Aufbau grundlegend verändert. Als Basis dient nun bei dem neuen 2000er, B2000 genannt, eine 500er Grundplatine. Die Kontroll-Logik für die Slots wurde hier wiederum in einen Custom-Chip, dem »Buster« gepackt. Somit entstand ein kompakter B2000, dem gleich ein 1 Mbyte RAM und ein erweiterter Video-Slot für erweiterte interne Video-Karten verpaßt wurde.

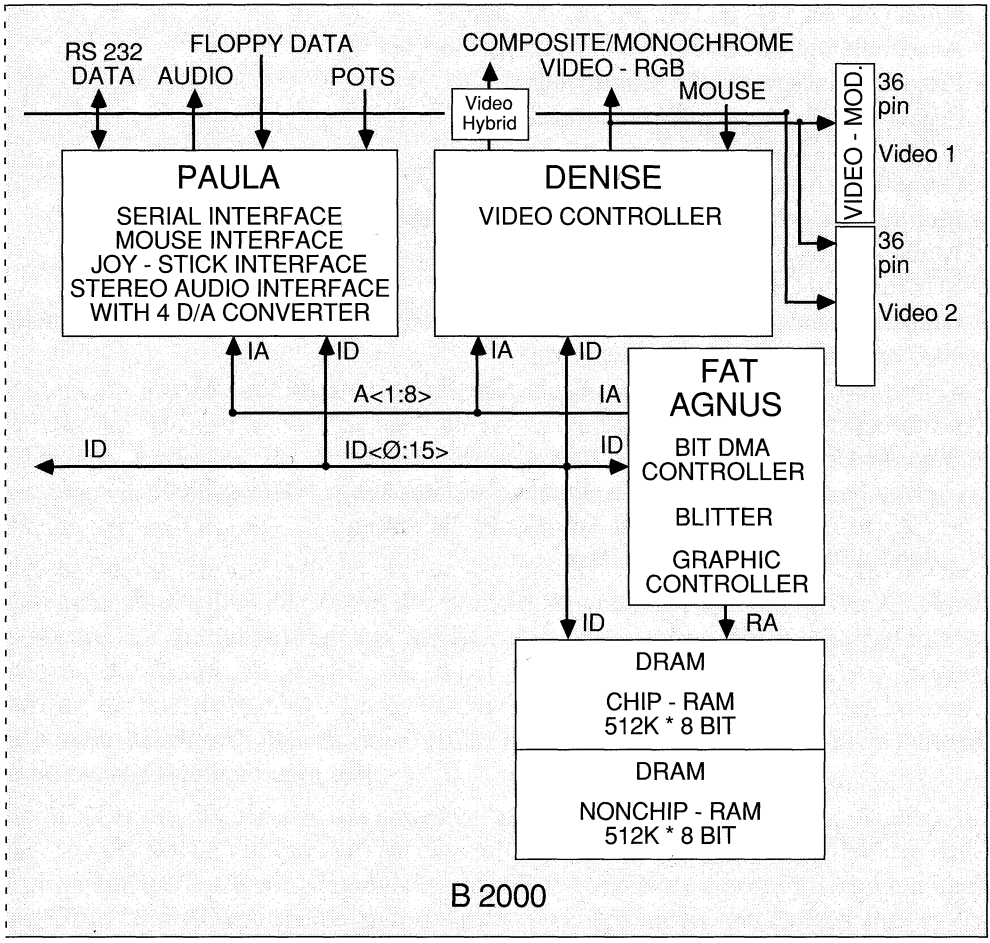


Z 1.3-2: *Das B2000-Blockdiagramm (Teil 1)*

Von dieser B2000-Version werden wir auch in diesem Buch ausgehen, da sie die neuste und auf absehbare Zeit die meist verbreiteste Version sein wird. Vom optischen Aufbau der Grundplatine hat sich einiges geändert, da, wie schon erwähnt, verschiedene Kleinteile gegen neue und erweiterte Custom-Chips ausgetauscht wurden. Bild 8 (Farbteil) zeigt Ihnen die Platine des B 2000.

Hier die Merkmale und Fähigkeiten eines Amiga B2000 auf einen Blick:

- Prozessor MC 68000 mit 7.16 Mhz getaktet.
- 1 Mbyte internes RAM, intern aufrüstbar bis 9.5 Mbyte.
- 256 Kbyte ROM.
- Eingebautes 3,5 Zoll Floppy, Speicherkapazität brutto 800 Kbyte, vorgesehene Ein-



Z 1.3-2: *Das B2000-Blockdiagramm (Teil 2)*

schübe für ein 5 1/4 und ein 3 1/2 Zoll Floppy.
- Anschlußmöglichkeit für 3 weitere Amiga-Floppies.
- Programmierbare serielle Schnittstelle.
- Programmierbare parallele Schnittstelle.
- Mitgelieferte Maus.
- Game-Port-Anschlüsse für Maus, Joystick usw.
- Multifunktions-Videoausgang, extern synchronisierbar.
- Audioausgänge, Stereo zweikanalig.
- Vier Ton- und Geräuschkanäle.
- Grafikauflösung 320 x 256, 320 x 512, 640 x 256, 640 x 512, mit speziellen Grafikmodi 4096 verschiedene Farben gleichzeitig.
- 4 Custom-Chips Agnus, Denise, Paula, Gary beispielsweise für DMA.
- Externe Tastatur PC-kompatibel.
- Insgesamt 7 Systemsteckplätze, davon
  - einen 86-Pin-Expansionsport, kompatibel zum Amiga 500 bzw. 1000,
  - 5 Zorro-Bus-Steckplätze mit AutoConfig-Funktion,
  - 4 PC/AT-kompatible Steckplätze.

Kapitel 2

Der Bootvorgang

Unter dem *Booten* des Amiga wird die Initialisierung des Amigas und seiner Hardware verstanden. Dies ist, wie wir meinen, ein sehr wichtiges Thema, da hier grundlegende Vorgänge stattfinden, die dem Hardwarefreak die Lösung zu einigen Problemen bieten, denn das »Hochfahren« des Rechners bewirkt neben der Initialisierung eine sogenannte System-Diagnose, die das Innenleben des Amiga »durchcheckt«.

Dieses Booten des Amiga findet in zwei Schritten statt. Zu Amiga-1000er-Zeiten waren dies beim Einschalten der Aufruf des Boot-ROM und nach dem Laden der *Kickstart*-Software die Initialisierung von EXEC, des sogenannten Multitasking-Betriebssystems des Amiga. Vorgesehen waren bei der 1000er-Serie auch mögliche Kickstart-ROMs, die unterhalb der Ur-Lader-*ROMs* (Boot-ROMs) eingesetzt werden konnten. Ein nachträglicher Umbau zu einem ROM 1000er ist also möglich. Bild 24 im Farbteil zeigt zwei ROM's im Amiga 1000.

Beim 500er und 2000er wird das Laden der Kickstart-Software nicht mehr benötigt, da hier sowohl Boot als auch *EXEC* in einem ROM (Festwert-Speicher) vorhanden sind. Ein großer Teil der Boot-Routinen ist eigentlich überflüssig geworden, man hat aber auf Grund der Kompatibilität zum 1000er nur wenig gändert und einfach alles in ein ROM gepackt. Beim Einschalten des Rechners verhält er sich dann einfach so, als ob die Kickstart-Diskette schon geladen ist. Bild 18 (Farbteil zeigt das ROM im 500er/2000er.

Kickstart im ROM hat sicherlich ein paar Nachteile. So können z.B. keine einfachen Manipulationen am System vorgenommen werden. Der große Vorteil, der sich hingegen ohne Laden der Kickstart bietet, ist, daß sofort beim Einschalten der Amiga bereit ist und man nicht erst noch die Kickstart-Disk suchen muß.

2.1: Funktion des Boot-ROMs

Das Boot-ROM übernimmt das eigentliche »Hochfahren« des Amiga. Dieses Boot-ROM beginnt bei $F80000 und endet bei $F81E86. Es wird immer dann aufgerufen, wenn CTRL Amiga Amiga bzw. Commodore Amiga betätigt oder das Netzteil eingeschaltet wurde. Beim Amiga 1000 blinkt hier, nach diesem Kalt- bzw. Warm-Start die Power-LED fünfmal, anschließend verfärbt sich der Monitor dunkelgrau.

Nach dieser Verfärbung des Bildschirms wird die *Power-LED* dunkel aufleuchten und die *DMA* und *Interrupts* werden abgeschaltet. Durch eine *Checksum*-Prüfung des Bereiches $FC0000 bis $FFFFFF wird getestet, ob sich die Kickstart schon im RAM- (bzw. ROM-) Bereich findet. Ist dies der Fall, so wird mit der Initialisierung von *EXEC* fortgefahren, wenn nicht, wird das Amiga-System getestet, da man davon ausgehen muß, daß es sich um einen Kaltstart handelt. Zunächst werden die RAM-Bereiche überprüft, hier zuerst das Kickstart-RAM. Tritt hier ein Fehler auf, so verfärbt sich der Monitor blaugrün und das Boot-ROM wird erneut gestartet. Anschließend findet ein Check des unteren RAM-Bereiches statt. Ein Fehler bewirkt die Grün-Färbung des Bildschirms und ebenfalls einen Neustart des Boot-ROMs. Danach beginnt ein Check der allgemeinen Hardware des Amiga, so wird kurz eine Tonfolge über die 4 Audio-Kanäle des Amiga ausgegeben. Anschließend werden die Ausnahmevektoren (2 bis 47) auf eine kleine Routine gesetzt, die aktiviert wird, falls ein Busfehler oder Interrupt eintritt. Tritt ein solcher Fehler auf, so verfärbt sich der Monitior gelb und es wird der Neustart des Boot-ROMs eingeleitet. Verlief bis hierhin alles klar, so wird das interne Laufwerk eingeschaltet und bei eingelegter Disk der Block 0 eingelesen. Befindet sich hier in dem ersten Long-Word die Kennung »KICK«, so wird von Spur 0 an steigend die eingelegte Diskette eingelesen und von dort aus in das Kickstart-RAM geschrieben. Wird die Kickstart-Disk nicht erkannt oder tritt ein Fehler beim Lesen auf, so erscheint eine Hand mit der Kickstart-Disk. Konnte das Kickstart eingelesen werden, so verfärbt sich der Bildschirm schwarz. Das Urlader-ROM wird abgeschaltet und EXEC gestartet.

Die Veränderung der Farben beim Booten ist eine sehr gute Systemdiagnose. Möchte man intern seine Amigas mit mehr RAM aufrüsten, kann so festgestellt werden, ob der Eingriff gelungen ist. Bei der Entwicklung der 512 Kbyte-Erweiterung zum Amiga 500 hatten wir z.B. einen Schluß auf dem Adreßbus der RAM-Erweiterung, der Bildschirm verfärbte sich grün beim Einschalten des Amiga. Der Fehler konnte mit dieser Hilfe schnell analysiert und beseitigt werden.

2.2: Die Initialisierung des Systems

Nachdem das Kickstart geladen bzw. gefunden wurde und das Urlader-ROM sich abgeschaltet hat, beginnt die Initialisierungsphase des Systems. Für diese Initialisierung ist EXEC zuständig. Am Anfang der Initialisierung wird, wie bei dem Starten der Ur-Lader-Software im Boot-ROM, die LED dunkel, die DMA und Interrupts werden gesperrt, der Bildschirm verfärbt sich dunkelgrau und die Ausnahmevektoren werden hier ebenfalls auf eine Fehlerroutine gesetzt, falls eine Ausnahmebedingung, wie z.B. ein Busfehler, auftritt. Danach wird gepüft, ob EXEC schon initialisiert ist und mittels Check-Summen-Prüfung wird getestet, ob es in Ordnung ist. Da der Abfangvektor *ColdCapture* durch den Speichertest des Boot-ROM auf 0 gesetzt wurde, wird mit der

eigentlichen Datenprüfung, nach dieser Prüfung der Initialisierung von EXEC, fortgefahren. Der ColdCapture-Vektor kann jedoch auch vom Anwender auf die Startadresse eines eigenen Programmes gesetzt werden. EXEC wird dieses Programm automatisch durch kurzes Verzweigen auf diesen Vektor starten, bevor mit der eigentlichen Datenprüfung fortgefahren wird. Zunächst wird getestet welcher, ob *Fast-* oder *Chip-Memory* und wieviel Speicher zur Verfügung steht. Befindet sich im Bereich von $C00000 bis $DC0000 *Fast-Memory*, so wird der EXEC-Datenbereich bei $C00000 angelegt, ansonsten bei $676. Danach wird der maximale Speicher durch Abfragen des Bereiches von 0 bis 2 Mbyte festgestellt. Ist dieser Bereich kleiner 256 Kbyte, so verfärbt sich der Bildschirm grün und es wird mit einer Endlosschleife, die ständig die Power-LED blinken läßt, fortgefahren. Wenn dieser Bereich größer oder gleich 256 Kbyte ist, verfärbt sich der Bildschirm mittelgrau. Anschließend beginnt die Einrichtung des EXEC-Datenbereiches. Die Prozessorbestückung wird ermittelt und verschiedene Listen und Header werden installiert. Die EXEC-Sprungliste zu den verschiedenen Befehlen wird eingerichtet. Die Ausnahmevektoren 2 bis 47 werden belegt, der Interrupt-Server, sowie der Task-Kontroll-Block und das Debugger-System installiert. Nun ist der Initialisierungsvorgang von EXEC fast am Ende, es wird nur noch der Supervisor-Modus abgeschaltet und die Tasks werden zum Scheduling freigeben. Die Anfangsadresse der Libraries wird in einer Liste gesammelt und im EXEC-Datenbereich abgelegt.

Anschließend wird die Power-LED auf hell umgeschaltet. An dieser Stelle befindet sich wieder ein Abfangvektor. Dieser Vektor wird mit *CoolCapture* bezeichnet. Ist dieser Vektor im RAM-Base gesetzt, so findet hier nochmals eine Verzweigung statt. Nach dieser möglichen Verzweigung werden alle residenten Module durch die EXEC-Routine InitCode initialisiert. Das letzte residente Modul, das initialisiert wird, ist die DOS-Bootstrap-Routine. Hierbei werden, falls eine Diskette im internen Laufwerk 0 eingelegt ist, die Sektoren 0 und 1 gelesen, in denen sich die DOS-Initialisierungsroutine befindet. Anschließend wird überprüft, ob die Kennung »DOS« vorhanden und die Checksumme fehlerfrei ist. Wird diese Erkennungsmarke nicht gefunden, oder die Checksumme ist fehlerhaft, so erscheint eine Hand, die das Einlegen der Workbench-Disk verlangt. Ist hingegen die Diskette erkannt worden und die Checksumme fehlerfrei, so wird die gelesene DOS-Initialisierungsroutine gestartet. Diese Routine öffnet das DOS-Library und übergibt die Kontrolle an Amiga-DOS. Mißlingt diese Übergabe beispielsweise wegen eines fehlerhaften Programms, so gibt InitCode die Kontrolle an EXEC zurück. Falls der WarmCapture-Vektor gesetzt ist, wird an dieser Stelle verzweigt, andernfalls wird der residente Debugger-ROM-Wack aufgerufen. Dieser kann jedoch nur mit einem Terminal an der seriellen Schnittstelle betrieben werden.

2.2.1: Die Abfangvektoren

Von großem Nutzen für Anwender können die Abfangvektoren *ColdCapture* und *CoolCapture* sein, da hier EXEC bei der Initialisierung des Systems auf ein eigenes Programm verzweigen kann. Der Vektor *WarmCapture* hingegen ist von keiner großer Bedeutung, da er nur bei einem Fehlschlagen der DOS-Inititalisierung aufgerufen wird.

Durch Verändern der Cold- bzw. CoolCaptureVektoren kann der Anwender sein Programm direkt in die Initialisierung des Systems einbinden, wodurch die Programmierung von Vieren oder Reset-Fester-RAM-Disks möglich ist. Bevor jedoch unser Programm aufgerufen werden kann, muß der Vektor auf die Startadresse des jeweiligen Programms gesetzt werden, das aufgerufen werden soll. Der erste Vektor, der aufgerufen wird, ist der ColdCapture. Er befindet sich bei $2A(AbsExecBase). Ist dieser Vektor ungleich Null, wird das adressierte ColdCapture-Programm mit »JMP« von EXEC angesprungen. Da der Stack noch nicht initialisiert wurde, kann das Programm nicht mit »JSR« aufgerufen werden. Um dennoch von dem ColdCapture-Programm zurück zum EXEC zu kommen, wurde die Rückkehradresse in das Adreßregister A5 abgelegt. Eine Rückkehr von dem aufgerufenen Cold-Capture-Programm zum EXEC ist dadurch nur mit »JMP (a5)« möglich. Hier sollte niemals »RTS« verwendet werden. Vor dem Sprung in die ColdCapture-Routine löscht EXEC den gesetzten Vektor. Dies hat zur Folge, daß bei nochmaligem Reset das Programm nicht mehr aufgerufen wird. Soll dies nicht der Fall sein, muß am Anfang des Programms der Vektor nochmals neu gesetzt werden.

Bei dem CoolCapture-Vektor, er ist zu finden bei $2E(AbsExecBase), ist die Initialisierung des Systems fast am Ende. Im Gegensatz zum ColdCapture wird der CoolCapture mit JSR aufgerufen, sobald der Vektor ungleich Null ist. Hier kann problemlos mit »RTS« zum EXEC zurückgekehrt werden. Dieser Vektor wird nicht gelöscht, ein Neusetzen ist somit nicht notwendig.

Leider genügt es nicht, einfach die Startadresse eines beliebigen Programms in einen der beschriebenen Vektoren einzutragen. Eine Checksumme sorgt hier für die notwendige Verwirrung. Es braucht jedoch keine eigene Routine für die Checksummenberechnung entwickelt zu werden. Hierzu kann man einfach die Checksummenberechnung von EXEC ($160 bis $182(EXEC)) verwenden.

Bei dem Programmieren von Reset-Routinen sollte man etwas vorsichtig sein, denn einmal gesetzt und aufgerufen, kann es bei fehlerhaften Programmen dazu führen, daß nur noch Ausschalten weiterhilft.

Der Programmierung von resetfesten Programmen steht nun nichts mehr im Wege. Bei unserem resetfesten Programm wird der ColdCapture-Vektor auf den Beginn einer Copperanimation gesetzt, die nach dem Betätigen der Maustaste endet und anschließend zum System zurückkehrt. Der CoolCapture-Vektor dient hier zum Neu-Initiali-

sieren des Speichers für die Reset-Routine. Dies ist notwendig, da beim Reset die Memory-List-Header neu initialisiert werden und somit der Speicher für unser Programm frei wird. Dies hätte zur Folge, daß die Reset-Routine von anderen Programmen überschrieben werden kann. So jedoch bleibt das Programm solange erhalten, bis der Rechner ausgeschaltet wird.

```
 1 ; *********************
 2 ;
 3 ; Demonstration für
 4 ; resetfeste Programme
 5 ; last update 10/03/88
 6 ; von Frank Kremser und Jörg Koch
 7 ; © Markt & Technik 1988
 8 ;
 9 ; *********************
10 ;
11 ; Dies ist eine Demonstration, wie man resetfeste Programme kon-
12 ; struiert. Das Programm setzt die zwei Resetvektoren › ColdStart ‹
13 ; und › CoolStart ‹. Von ColdStart aus wird das Hauptprogramm ge-
14 ; startet, das neue Copperlisten aufstellt. Wurde dann eine Maus-
15 ; taste gedrückt, so fährt der Rechner weiter fort mit der
16 ; Initialisierung. Dabei wird dann auch die Memory-Liste erstellt.
17 ; Anschließend wird dann über den CoolStart eine kurze Routine
18 ; aufgerufen, die den Speicherplatz, der vom Programm benötigt
19 ; wird, in der Memory-Liste als belegt einträgt. Damit ist das
20 ; Programm dann weitgehend vor dem Überschreiben geschützt. Das
21 ; bedeutet, das Programm läßt sich nur noch durch Ausschalten
22 ; des Rechners beseitigen.
23 ; Mittels dieser Routine könnte man also ohne größere Probleme
24 ; einen AMIGA-Virus schaffen, wovon wir aber ausdrücklich abraten
25 ; wollen.
26 ;
27 ; ******************************
28
29          move.l   4,a6
30          move.l   #10000,d0      ;10000 Byte für dieses Programm
31          lea      start(pc),a1   ;Ab Programmstart bereitstellen
32          jsr      -$0cc(a6)      ;(AllocAbs)
33 start:   move.l   4,a6           ;Startadresse von EXEC
34          lea      reset(pc),a0   ;Startadresse des Hauptprogramms
35          move.l   a0.$2a(a6)     ;in den ColdCapture-Vektor speichern
36          lea      mem(pc),a0     ;und Adresse der Mem-Allocation-Rout.
37          move.l   a0,$2e(a6)     ;in den CoolCapture-Vektor speichern
38
39          move.w   #0,$24(a6)     ;anschließend neue Checksumme für den
40          moveq    #0,d1          ;untersten Datenbereich erstellen,
41          lea      $22(a6),a0     ;damit EXECc keinen Fehler erkennt
42          moveq    #$18,d0
43 sum:     add.w    (a0)+,d1
```

```
44          dbra    d0,sum
45          not.w   dl
46          move.w  dl,$24(a6)
47          moveq   #0,dl
48          lea     $22(a6),a0
49          moveq   #$18,d0
50 sum2:    add.w   (a0)+dl
51          bra     d0,sum2
52          not.w   dl              ;Ende der Initialisierung
53          rts
54
55 mem:     move.l  #10000,d0       ;10000 Byte für dieses Programm
56          lea     start(pc).al    ;Ab Programmstart bereitstellen
57          jsr     -$0cc(a6)       ;(AllocAbs)
58          rts                     ;Ende der Speicherallokierung
59
60 ; ******************************
61 ;
62 ; Dies war der erste Teil des Programmes.
63 ; Ab hier kann ein eigenes Programm eingefügt werden.
64 ;
65 ; ******************************
66
67 reset:  movem.l d0-d7/a0-a6,-(a7) ;Register retten
68          lea.l   $50000,a0       ;Bitplane ab $50000
69          move.l  #6645,d0        ;6645 Byte
70 clear:  clr.l   (a0)+            ;löschen
71          dbf     d0,clear        ;dekrementiere, teste
72                                  ;d0 = 0, nein dann clear
73
74          jsr     start           ;Resetvektoren neu setzen
75          lea.l   coltab(pc),al   ;Zeiger auf Farbtabelle;
76          bsr     copperinit      ;Cooperliste initialisieren
77          jsr     -132(a6)        ;---> Fortbid();
78                                  ;Multitasking abschalten
79          lea.l   $dff000,a5      ;a5 = Customchipbase
80          move.w  #$03e0,$96(a5)  ;DMA - Control write
81                                  ;die Bits DMA enable,
82                                  ;Bit-Plane DMA enable,
83                                  ;Coprocessor DMA enable,
84                                  ;Blitter DMA enable,
85                                  ;sowie Sprite DMA enable,
86                                  ;werden im DMA-Register
87                                  ;gelöscht
88
89          move.l  #$55000,$80(a5) ;COP1LCH := $55000
90                                  ;indirekte JMP - Adresse des
91                                  ;Copper location register
92                                  ;wird auf die Adresse
93                                  ;$55000 gesetzt (Copperliste)
```

```
 94          clr.w   $88(a5)             ;COPJMP1 loeschen
 95          move.w  #$1100,$100(a5)     ;BLCON0 = #$1100
 96          clr.l   $102(a5)            ;löschen von BLCON1
 97          clr.l   $108(a5)            ;löschen von BL1MOD
 98
 99          move.w  #$8380,$96(a5)      ;DMA control write
100                                     ;die Bits DMA enable,
101                                     ;Bit-Plane DMA enable,
102                                     ;Coprocessor DMA enable
103                                     ;werden gesetzt
104
105          move.l  #0,a2               ;Initialisierung von a2
106 main:    andi.b  #64,$bfe001         ;Ist Bit 6 gesetzt?
107          beq     ende                ;Wenn ja, dann war Maustaste
108                                     ;gedrückt, Programm beenden
109          lea.l   coltab(pc),a1       ;Zeiger auf Colortabelle
110          add.l   #1,a2               ;a2 um 1 erhöhen,
111          cmp.l   #32,a2              ;vergleichen ob gleich 32
112          bne     cont                ;wenn nicht, dann fortfahren
113          move.l  #0,a2               ;wenn nein, dann a2 zurückset.
114          adda.l  a2,a1               ;a2 zu a1 hinzuaddieren
115          bsr     copperinit          ;und neue Copperliste erstel.
116          move.l  #$ff,d3             ;Wert für Warteschleife setzen
117 loop:    tst.l   (a6)                ;Dummybefehl zur Verzögerung
118          dbra    d3,loop             ;d3 erniedrigen und ggf. fortf.
119          bra     main                ;Das ganze nocheinmal
120
121 ende:    lea.l   gfxname(pc),a1      ;Zeiger auf gfxname in a1
122          jsr     -408(a6)            ;--> OldOpenLibrary();
123          move.l  d0,a4               ;alte Copperliste holen
124          move.l  38(a4),$80(a5)      ;und in COP1LCH setzen
125          clr.w   $88(a5)             ;COPJMP1 löschen
126          move.w  #$8060,$96(a5)      ;Sprite und Blitter DMA
127                                     ;setzen
128          jsr     -138(a6)            ;--> Permit();
129                                     ;Multitasking ein
130
131          movem.l (a7)+,d0d7/a0-a6    ;Register zurückgeben
132          move.l  #reset.$2a(a6)      ;Resetvektor nocheinmal set.
133          jmp     (a5)                ;Rückkehr
134
135 copperinit:
136          clr.l   d1                  ;d1 löschen
137          lea.l   $55000,a0           ;Startadresse für
138                                     ;Cooperliste
139          move.l  #$00e00005,(a0)+    ;BitPlane1 Zeiger setzen
140          move.l  #$00e20000,(a0)+    ;
141          move.l  #$01800f00,(a0)+    ;Color0 auf rot setzen
142          move.l  #80,d0              ;Ab Zeile 80 Farbrotation
143 loop1:   move.b  d0,(a0)+            ;Wait-VP
```

```
144         move.b   #1,(a0)+           ;Wait - HP + Bit 0 gesetzt
145         move.l   #$fffe0180,(0)+    ;Wait + Color0
146         clr.l    d6
147         move.b   (a1,d1),d6         ;neue Farbe aus Tabelle holen
148         eori     #$0ff0,d6          ;bearbeiten
149         move     d6,(a0)+           ;u. in Copperli. f. Color 0 eintr.
150         addq     #1,d1              ;Farbindex um 1 erhöhen
151         cmpi     #32,d1             ;Wenn 32 Farben gezeigt,
152         bne      loop2
153         clr.l    d1                 ;dann Index zurücksetzen
154 loop2: add.l    #1,d0              ;Zeilenindex um 1 erhöhen
155         cmp.l    #255,d0            ;und mit 255 vergleichen
156         beq      loop3
157         bra      loop1
158 loop3: move.l   #$ff01fffe,(a0)+   ;warten, bis Zeile 255 erreicht
159         move.l   #$01800f00,(a0)+   ;dann Color0 auf rot setzen
160         move.l   #$fffffffe,(a0)+   ;Endekennzeichnung für Copper
161         rts                         ;Rückkehr
162
163 ;Farbtabelle, doppelt so lang, wie Farbanzahl,
164 ;da Rotation
165 even
166 coltab: dc.b    $00,$10,$20,$30,$40,$50,$60,$70,$80,$90
167         dc.b    $a0,$b0,$c0,$d0,$e0,$f0,$f0,$e0,$d0
168         dc.b    $c0,$b0,$a0,$90,$80,$70,$60,$50,$40,$30
169         dc.b    $20,$10,$00,$00,$10,$20,$30,$40,$50,$60
170         dc.b    $70,$80,$90,$a0,$b0,$c0,$d0,$e0,$f0,$f0
171         dc.b    $e0,$d0,$c0,$b0,$a0,$90,$80,$70,$60,$50
172         dc.b    $40,$30,$20,$10,$00
173
174 gfxname:dc.b "graphics.library",0
```

Kapitel 3

Der MC68000

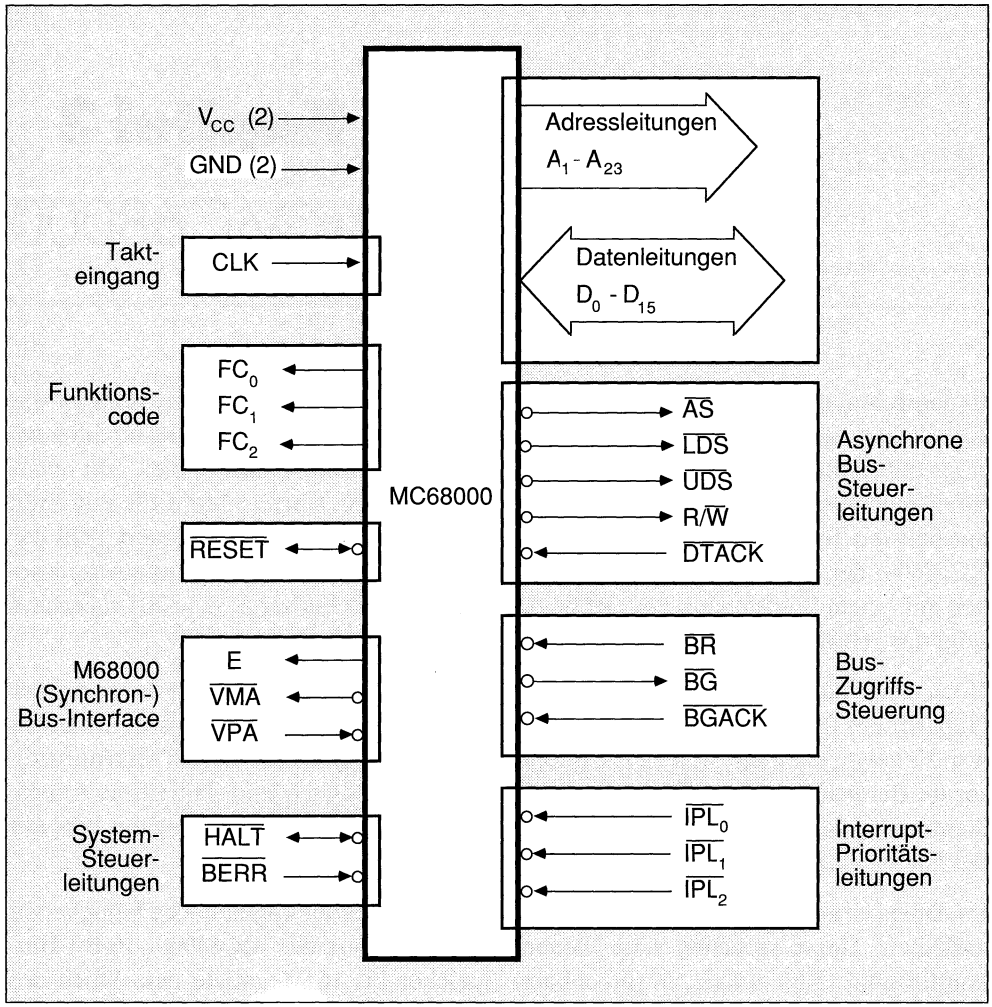
10 Jahre müssen wir zurückblicken, um die Anfänge dieses Mikroprozessors an das Tageslicht zu holen. Wir schrieben das Jahr 1977, als man begann, Software auf Siliziumchips mit minimalen Abmaßen zu bringen. Auch Motorola versuchte dies. Mit neuer MOS-Technologie und auf den Grundlagen des MC6800, startete Motorola mit ihren Spezialisten Tom Gunter und Co. das Projekt MC68000. Daraus entstand 1979 der XC68000, der Mikroprozessor der 80er Jahre, der insgesamt 68000 Transistorfunktionen auf einem Siliziumchip der Größe 6,2 mal 7,1 mm beherbergt. Bild 20 im Farbteil zeigt das „Herz" des Amigas.

Doch erst 1984 kam dieser Mikroprozessor zu großem Glanz, in dem Platinencomputer Gepard, der als Zusatz zum damals populären Apple oder als Einzelgerät erhältlich war. Es zeigte erstmals, welche faszinierenden Möglichkeiten mit diesem Mikroprozessor im Bereich Grafik und Berechnungen von Daten möglich sind. Die Firma Amiga, damals eine Joystick-Firma, sah in diesem Prozessor die Zukunft und begann 1984 mit der Entwicklung eines Supercomputers: dem Amiga.

Im Amiga ist er einer der »vier Großen«, die diesem Rechner unglaubliche Fähigkeiten verleihen. Heute sprechen viele Fakten für den Einsatz des MC68000: 16-Bit-Bus, komfortabler Befehlssatz, ansprechbarer Speicher bis 16 Megabyte usw. Doch der Trend geht schon zum Einsatz von noch leistungsfähigeren Prozessoren, wie dem MC68020, dem MC68030 oder sogar dem, noch in Entwicklung befindlichem MC68040.

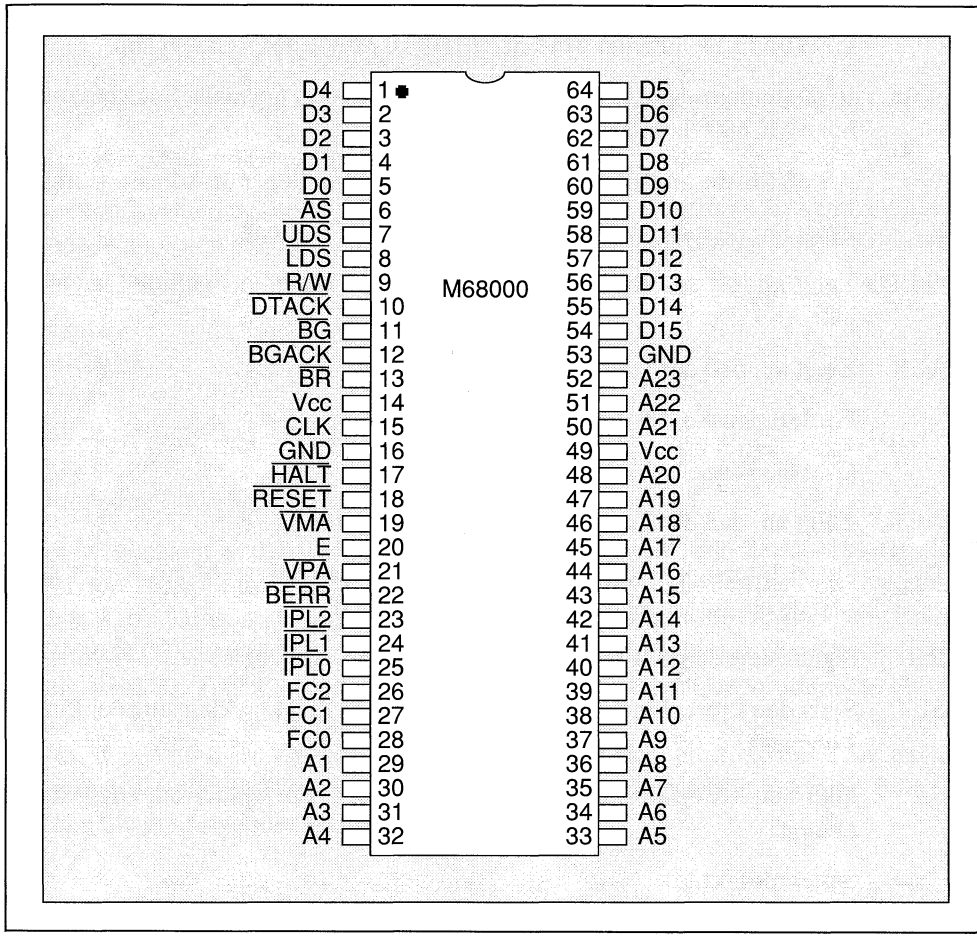
3.1: Der MC68000 im Detail

Der MC68000 verfügt über 24 *Adreßleitungen*, wobei A0 nicht als solche gekennzeichnet ist, 16 *Datenleitungen*, 3 Leitungen zur Interrupt-Prioritäts-Mitteilung usw. Die Anschlüsse im Überblick zeigt folgendes Blockschaltbild (Z 3.1-1):



Z 3.1-1: *Einteilung der Leitungen des M68000*

Wie die Anschlüsse bei dem Originalgehäuse verteilt sind, zeigt Abbildung 3.1-2:



Z 3.1-2: *Pinbelegung des Dual-In-Line-Gehäuses*

Die Signale des MC68000:

A1–A23 23 Adreßleitungen zum direkten Ansprechen von 8 Mbyte. A0 könnte aus UDS und LDS erzeugt werden, so daß 16 Mbyte ansprechbar sind.

D0–D15 16 Datenleitungen. Daraus resultiert, daß der normale Speicherzugriff beim MC68000 ein Word-Zugriff ist.

AS Adreß-Strobe gibt an, daß die auf dem Adreßbus liegende Adresse gültig ist.

R/W Zeigt an, ob ein Lese- oder ein Schreibzugriff vorliegt.

UDS/LDS Zeigt an, ob das obere/untere Byte auf dem Datenbus gültige Daten enthält.

DTACK Zeigt an, daß der Datentransfer beendet ist.

BR Fordert den Adreß- und den Datenbus an.

BG Gibt die Busse frei.

BGACK Zeigt an, daß die Busfreigabe empfangen wurde.

IPL0–2 Diese Interruptleitungen lösen eventuell einen Interrupt aus, wenn nicht bereits ein Interrupt höherer Priorität vorliegt.

BERR Signalisiert dem MC68000, daß ein Busfehler vorliegt.

RESET Setzt den Prozessor zurück, wenn mindestens 10 Taktzyklen lang der Pin auf Low liegt.

HALT Hält den MC68000 so lange an, bis an diesem Pin wieder ein High-Signal anliegt.

E Synchrontakt.

VMA Gültige Speicheradresse liegt vor.

VPA Gültige Peripherieadresse liegt vor.

FC0–2 Funktioncode-Ausgänge zeigen an, in welchem Zustand sich der MC68000 gerade befindet.

CLK Systemtakt

VCC Versorgungsspannung +5V

GND Masse

3.2: Die Exceptions

Der MC68000 hat drei definierte Zustände:

1. Normalzustand, d.h. Programmausführung.

2. Haltezustand, d.h. am HALT-Eingang des Prozessors liegt ein Low-Signal an.

3. Ausnahmezustand (Exceptions).

Im dritten Zustand behandelt der Prozessor bestimmte Routinen. Er kann auf die verschiedensten Arten dazu gebracht werden, solche Routinen abzuarbeiten. Diese Möglichkeiten sind beispielsweise:

Interrupts
- Busfehler
- Reset
- Durch die Befehle TRAP, TRAPV, CHK und DIV
- Adreßfehler
- Durch den Trace-Modus

Jeder Möglichkeit ist dabei ein bestimmter Vektor zugeteilt, der die Adresse der auszuführenden Exception-Routine enthält. Diese Vektoren befinden sich im Speicherbereich $000 bis $3FF und können auch vom Benutzer gesetzt werden, wobei die Routine dann mit RTE verlassen werden muß. Beim Amiga jedoch besteht dazu noch ein Hindernis, denn das EXEC arbeitet in diesem Bereich mit einer Checksumme, so daß eine einfache Änderung nichts nutzt. Zusätzlich muß noch eine neue Checksumme berechnet werden. Wie diese Checksummenberechnung funktioniert, ersehen Sie aus dem Programmbeispiel zu Kapitel 2. Welche Exceptions nun welche Vektoren benutzen, ersehen Sie aus folgender Tabelle:

Adr	Nr.	Funktion
000	00	Supervisor-Stackpointer
004	01	Anfangsadresse für Programmcounter nach RESET
008	02	Busfehler
00C	03	Adreßfehler
010	04	nicht implementierter Befehl
014	05	Division durch Null
018	06	Befehl CHK
01C	07	Befehl TRAPV
020	08	Privilegverletzung
024	09	Traceroutine
028	0A	Emulator für 1010-Befehlscode
02C	0B	Emulator für 1111-Befehlscode

Adr	Nr.	Funktion
30	0C	
–	–	reserviert
038	0E	
03C	0F	nicht initialisierter Interrupt
040	10	
–	–	reserviert
05C	17	
060	18	falscher Interrupt
064	19	Interruptvektor Ebene 1
068	1A	Interruptvektor Ebene 2
06C	1B	Interruptvektor Ebene 3
070	1C	Interruptvektor Ebene 4
074	1D	Interruptvektor Ebene 5
078	1E	Interruptvektor Ebene 6
07C	1F	Interruptvektor Ebene 7
080	20	
–	–	Trap-Befehlsvektoren
0BC	2F	
0C0	30	
–	–	reserviert
0FC	3F	
100	40	
–	–	Anwender-Interruptvektoren
3FC	FF	

Kapitel 4

Die Custom-Chips

Der Amiga besitzt neben dem MC68000 noch weitere »intelligente« Bausteine, die wesentliche Aufgaben übernehmen. Diese Bausteine sind:

Amiga 1000: *Agnus, Denise* und *Paula*

Amiga 500: *FatAgnus*, Denise, Paula und *Garry*

Amiga A2000: Agnus, Denise und Paula

Amiga B2000: FatAgnus, Denise, Paula, Garry und *Buster*

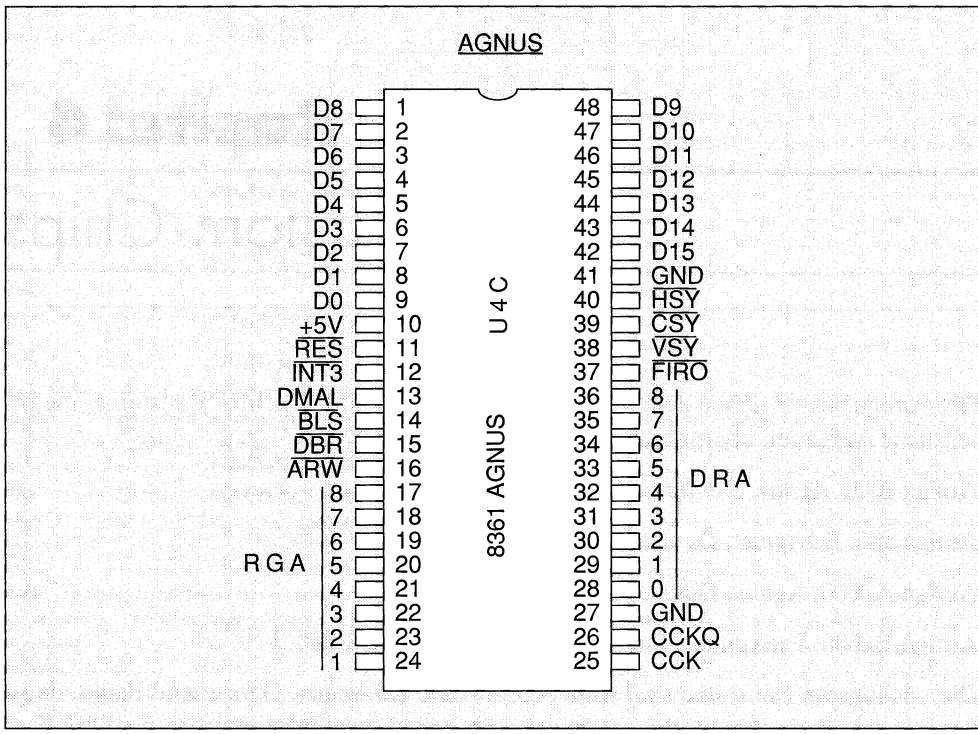
Die wichtigsten Bausteine sind wohl Agnus, bzw. FatAgnus, Denise und Paula, da sie Funktionen übernehmen, die sofort erkennbar sind, wie beispielsweise die Disk-Kontrolle oder die Sprite-Darstellung. Bild 23 (Farbteil) zeigt die Custom-Chips des A 1000.

Garry und Buster sind lediglich Bauteile, die verschiedene Steuerschaltungen ersetzen, aber nicht softwaremäßig angesprochen werden können.

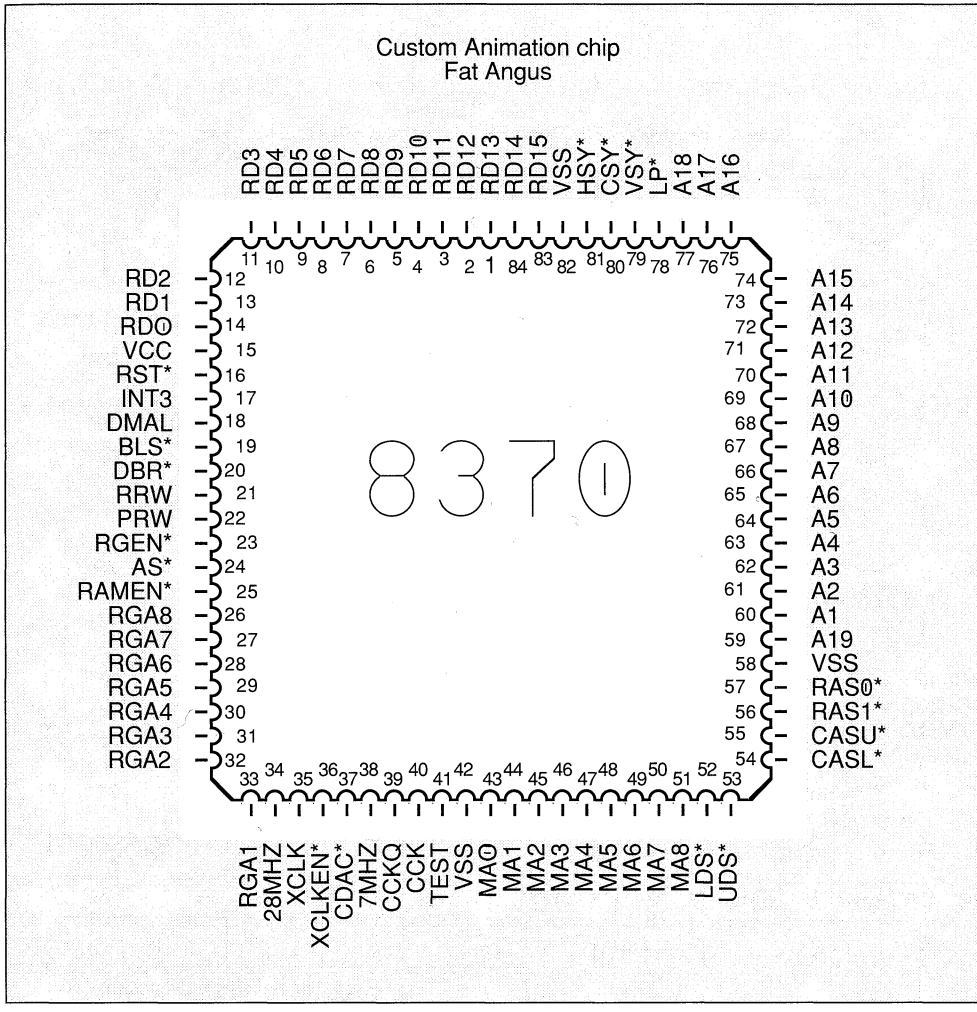
4.1: Agnus und FatAgnus

Im Amiga 1000 wurde als einer der Custom-Chips Agnus eingesetzt. Dieser Chip wurde in den späteren Amiga-Produkten, ausgenommen Amiga A2000, durch eine Nachfolgeversion, FatAgnus genannt, ersetzt. Dieser neue Chip enthält einige Steuerschaltungen, die zuvor noch auf der Platine zu finden waren. Zudem enthält er die komplette *Refresh-Logik* für 1 Mbyte, so daß die RAM-Erweiterung beim A500 sehr einfach gehalten werden konnte, bzw. auf dem Amiga B2000 1 Mbyte direkt auf der Platine integriert werden konnten. Außerdem erzeugt er alle *Takte*, die daß System benötigt, aus dem Grundtakt von 28,63636 MHz, was zuvor von einer externen Schaltung übernommen wurde. Vergleiche Bild 21 und 22 im Farbteil.

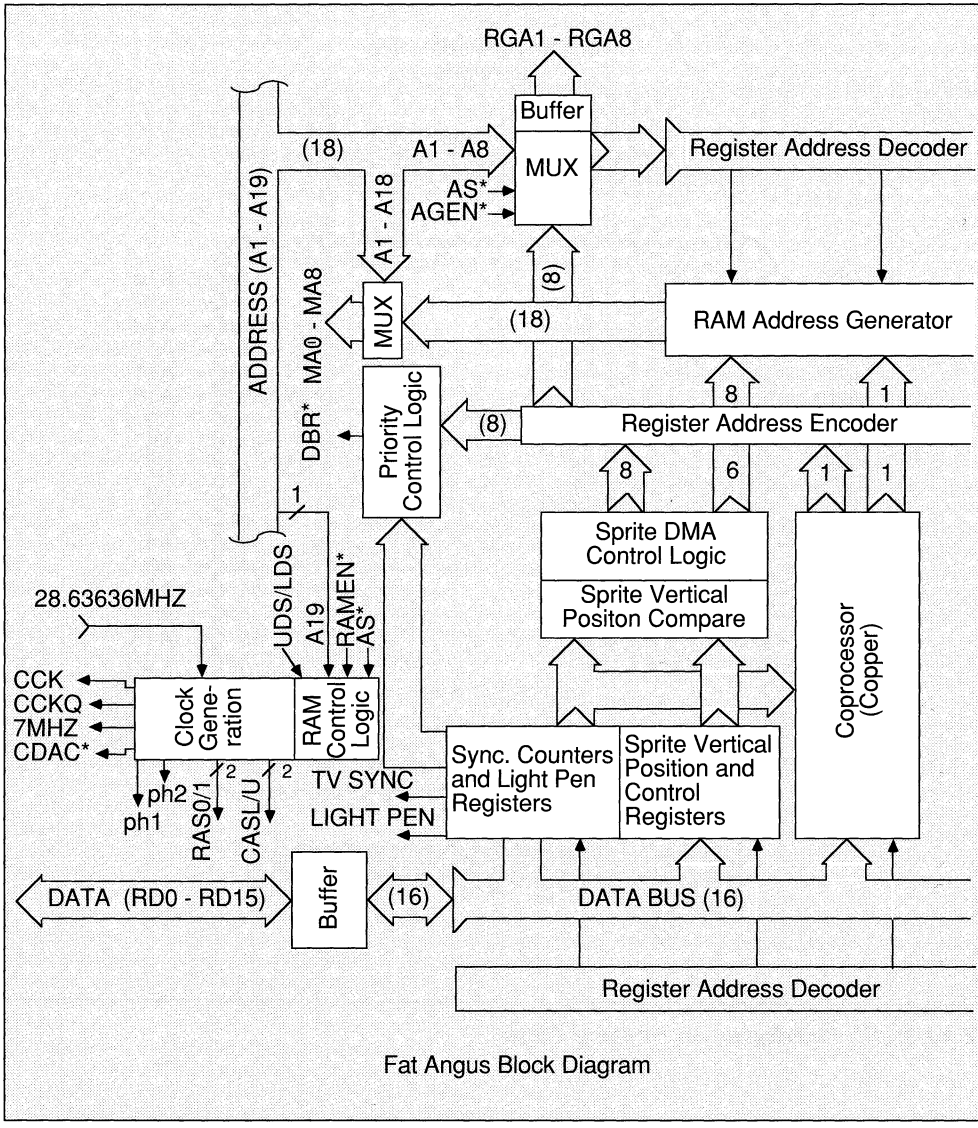
Die folgenden Abbildungen zeigen die Pinbelegungen von Agnus und FatAgnus, sowie das Blockschaltbild zu FatAgnus:



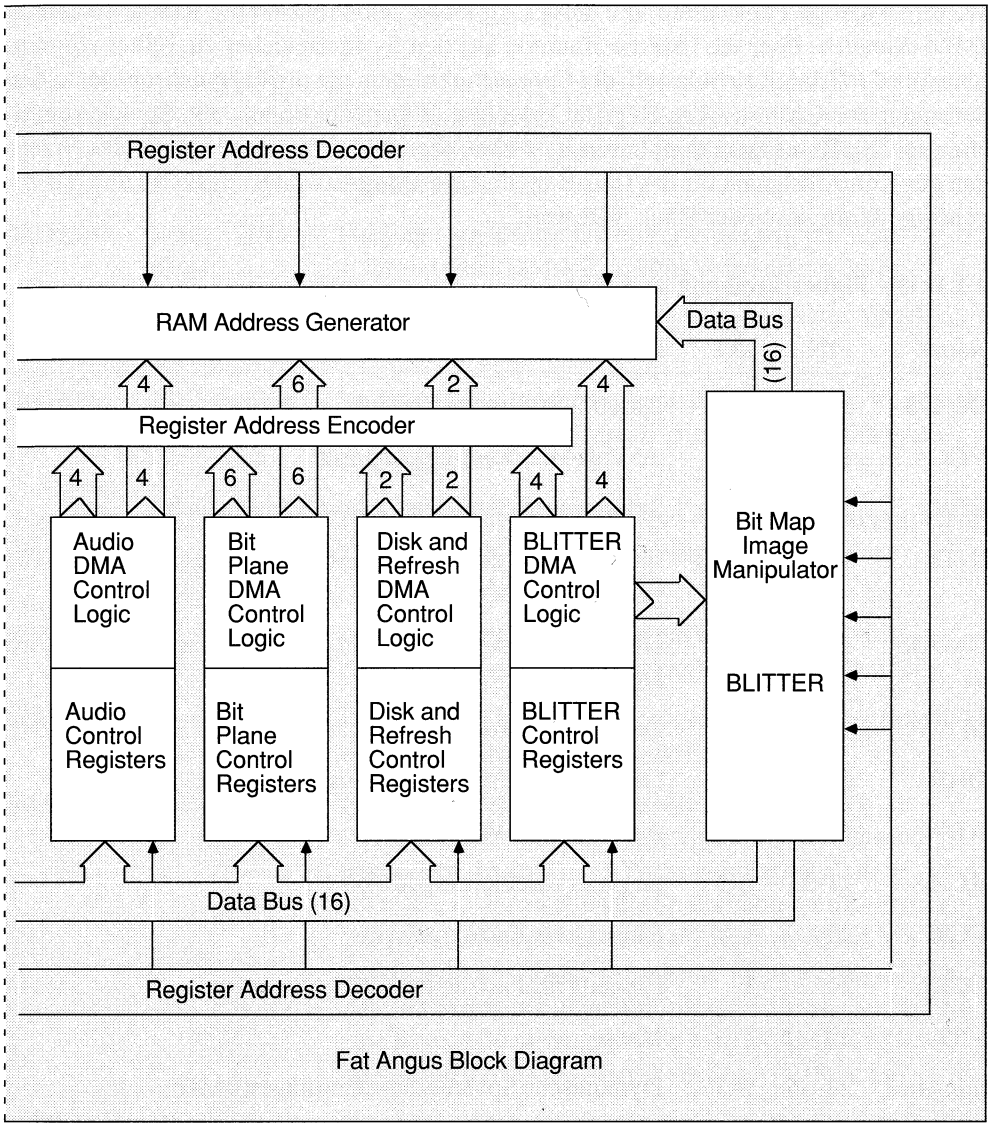
Z 4.1-1: *Die Pinbelegung des Agnus-Chips*



Z 4.1-2: *Die Pinbelegung des FatAgnus-Chips*



Z 4.1-3: *Das Blockschaltbild von FAT-Agnus (Teil 1)*



Z 4.1-3: *Das Blockschaltbild von FATAgnus (Teil 2)*

Weitere wichtige Funktionen, die diese Chips übernehmen, sind die Kontrolle von 25 *DMA-Kanälen*, über die externe Bauteile auf den Systemspeicher zugreifen können, ohne die *CPU* damit zu belasten, die *Copper*funktionen, die displaysynchronisierte Änderungen der Custom-Chip-Register erlauben, *Blitter*funktionen, um Speicherbereiche des ChipRams zu manipulieren, fast ohne die CPU zu bremsen, sowie das Erzeugen der Kontrollsignale für das ChipRam und die 1Meg-Erweiterungskarte und das zugehörige *Multiplexen* der Ram-Adressen.

4.1.1: Die Pinbeschreibung zu Agnus

Name	PIN	I/O	Beschreibung
D0–D8	1–9	I/O	Datenbusleitungen 0 bis 8
VCC	10	I	+5Volt-Versorgungsspannung
RES	11	I	Systemreset. Setzt den Custom-Chip zurück.
INT3	12	O	Interrupt-Level 3 Ausgang. Fordert bei der MC68000 CPU einen Interrupt an.
DMAL	13	I	DMA-Request-Leitung. Ein direkter Speicherzugriff wird angefordert.
BLS	14	I	Der Blitter wird verlangsamt (BLITTER- SLOWDOWN)
DBR	15	O	Datenbus-Anfrage an die Busverwaltung.
ARW	16	O	Agnus-RAM-Write-Signal
RGA8–1	17–24	I/O	Registeradreßleitungen 8 bis 1
CCK	25	I	Eingang für Farbsignaltakt
CCKQ	26	I	Eingang für Farbsignaltakt-Verzögerung
VSS	27	I	Masse
DRA0–8	28–36	I/O	Dynamische Speicheradreßleitungen 0 bis 8
LP	37	I	LightPen-Eingang
VSY	38	I/O	Vertikales Synchronsignal
CSY	39	O	Synchronsignal für Composite-Ausgang
HSY	40	I/O	Horizontales Synchronsignal
VSS	41	I	Masse
D15–D9	42–48	I/O	Datenbusleitungen 15 bis 9

4.1.2: Die Pinbeschreibung zu FatAgnus

Name	PIN	I/O	Beschreibung
A19–A1	59–77	I	Adreßbusleitungen. Die Leitungen A1 bis A8 werden auch von der CPU benutzt, um die internen Register zu adressieren.
RD15–0	1–14 83/84	I/O	Dies sind gepufferte Datenbusleitungen, die auch während eines DMA-Zugriffes verwendet werden.
AS	24	I	Dieses Signal zeigt an, daß die Daten auf dem Adreßbus verwendbar sind.
RGEN	23	I	Ist dieses Signal zusammen mit AS aktiviert, so werden die Daten auf dem Adreßbus als interne Registeradressen verstanden.
RAMEN	25	I	Ist dieses Signal zusammen mit AS aktiviert, so werden die Daten des Adreßbusses gemultiplext und auf dem Adreßbus MA bereitgestellt.
PRW	22	I	Ist dieses Signal low, so liegt ein Write-Zugriff vor. Bei einem High-Signal liegt ein Read-Zugriff vor.
RRW	21	O	Dieses Signal kennzeichnet einen Read-/Write- Zugriff an weitere Systemteile. Sonst wie bei PRW.
MA0–8	43–51	O	Auf diesem Adreßbus liegen die gemultiplexten Adreßdaten an. Diese Daten werden in zwei Phasen übergeben. In der ersten Phase werden die Zeilen-adressen, in der zweiten die Spaltenadressen übergeben. Die Adressen beziehen sich auf 256Kbyte-DRAM's, wobei nur die unteren 512 Kbyte angesprochen werden können. Allerdings liegen nur gemultiplexte Adressen an, wenn auf eine Ram-Adresse zugegriffen wird (RAMEN ist low), oder wenn ein DMA-Zugriff vorliegt (DBR ist low).
LDS	52	I	Dies ist der Lower-Data-Strobe der CPU. Entsprechend dem Eingangssignal wird CASL gesetzt.
UDS	53	I	Dies ist der Upper-Data-Strobe der CPU. Entsprechend dem Eingangssignal wird CASU gesetzt.
CASL	54	O	Wird entsprechend LDS gesetzt und beeinflußt die Spalten-Adressierung der DRAMs. Ist dieses Signal aktiv, entspricht das einem Zugriff auf das Lowbyte des Datenwortes.

Name	PIN	I/O	Beschreibung
CASU	55	O	Wird entsprechend UDS gesetzt und beeinflußt die Spalten-Adressierung der DRAMs. Ist dieses Signal aktiv, entspricht das einem Zugriff auf das Highbyte des Datenwortes.
RAS0	57	O	Dieses Signal wird aktiv, wenn auf die unteren 512 Kbyte zugegriffen wird.
RAS1	56	O	Dieses Signal wird aktiv, wenn auf die oberen 512 Kbyte zugegriffen wird, die FatAgnus verwalten kann.
DBR	20	O	Dieses Signal ist aktiv, wenn ein DMA-Zyklus bevorsteht.
RGA8–1	26–33	O	Dies ist ein Register-Adreß-Bus. Über diesen Bus kann FatAgnus auf Register weiterer Custom-Chips zugreifen.
HSY	81	I/O	Ist dieses Signal als Eingang geschaltet (GenlockVideo on), so kann über diesen Pin der horizontale Rasterstrahl-Zähler extern synchronisiert werden. Ansonsten liegt hier der Horizontalsynchronimpuls des Systems an.
VSY	79	I/O	Ist dieses Signal als Eingang geschaltet (GenlockVideo on), so kann über diesen Pin der vertikale Rasterstrahl-Zähler extern synchronisiert werden. Ansonsten liegt hier der Vertikalsynchronimpuls des Systems an.
CSY	80	O	An diesem Pin liegen die Synchronisationssignale für den Composite-Video-Ausgang an. HSY, VSY und CSY sind NTSC-Kompatibel.
LP	78	I	Wird dieses Signal auf low gesetzt, so zeigt dies an, daß die Lightpen-Position mit der des Rasterstrahls übereinstimmt.
RST	16	I	Setzt FatAgnus zurück.
INT3	17	O	Fordert einen Interrupt der Stufe 3 bei der CPU an. Ein solcher Interrupt wird immer dann angefordert, wenn der Blitter einen Datentransfer beendet hat und für neue Aufgaben bereitsteht.
DMAL	18	I	Dieses Signal wird aktiv gesetzt, wenn eine externe Komponente einen Audio- und/oder Disk-DMA-Zugriff benötigt.
BLS	19	I	Ist dieses Signal aktiv, so wird der Blitter gestoppt, so daß die CPU diesen Zyklus verwenden kann.

Name	PIN	I/O	Beschreibung
28MHZ	34	I	An diesem Pin muß der Systemtakt von 28,63636 MHz anliegen. Dieser Takt gilt als aktiv, wenn XCLKEN High ist.
XCLK	35	I	An diesem Pin kann ein alternativer Systemtakt anliegen, der als Takt verwendet wird, wenn XCLKEN Low ist. Ein solcher alternativer Takt ist nötig, wenn das System mit einer externen Videoquelle o.ä. synchronisiert werden soll.
XCLKEN	36	I	Dieses Signal kontrolliert, welcher Systemtakt verwendet werden soll.
CCK	40	O	Dies ist der Takt, der als Farbträgersignal dient.
CCKQ	39	O	Dies ist der gleiche Takt wie CCK, allerdings um 90 Grad nachhängend.
7MHZ	38	O	Dies ist der Takt, der beispielsweise von der CPU verwendet wird.
CDAC	37	O	Dies ist der gleiche Takt wie 7MHZ, nur um 90 Grad vorlaufend.
TEST	41	I	Ist dieses Signal aktiv, so wird der Prozessorzyklus unterbrochen und die internen Register können in jedem CCK-Zyklus angesprochen werden (Copper-Befehlsliste).

4.1.3: Der Copper

Agnus und FatAgnus enthalten einen CoProzessor, der wohl das interessanteste Baustück des Amiga-Systems darstellt. Er ist mit dem *Rasterstrahl* synchronisiert. Durch diese Synchronisierung ist dieser Copper in der Lage, Register der Custom-Chips in Abhängigkeit der Rasterstrahlposition zu modifizieren. Dazu muß eine sogenannte *Copper-Liste* aufgestellt werden, die im ChipMem abzulegen ist, damit die DMA diese Daten fortlaufend an den Copper übertragen kann.

Der große Vorteil des Copper's besteht darin, daß ohne großen Aufwand Videomanipulationen möglich sind, die ohne ihn gar nicht, oder nur schwer möglich wären. Beispiele dafür lassen sich in großer Zahl finden. Hier nur zwei davon:

- Die Möglichkeit, mehrere Screens auf einmal auf einem Bildschirm übereinander darzustellen, wird erst durch den Copper ermöglicht, da dieser an den jeweiligen Positionen die Chip-Register auf die Videodaten eines neuen Screens setzt (Bilddaten, Farbregister, Sprites usw.).

- Durch den Copper ist es möglich, Sprites mehrmals auf einem Bildschirm zu zeigen, indem, nach der Beendigung der Darstellung des Sprites, die zugehörigen Register zu diesem Sprite auf neue Daten gesetzt werden.

Im Normalzustand des Rechners steuert der Copper alle Displayfunktionen. Das heißt, das eine Copperliste besteht, die alle Farben, Screendaten usw. fortlaufend setzt. Das hat natürlich auch zur Folge, daß es keinen Zweck hat, eine neue Hintergrundfarbe zu setzen, indem einfach ein neuer Wert in das entsprechende Farbregister geschrieben wird, da der Copper bei einer normalen Copperliste auch Befehle vorfindet, um die Farben zu setzen. Man muß also direkt die Copperliste manipulieren. In diese Copperliste können natürlich auch eigene Befehle eingeschrieben werden, bzw. es kann eine ganz neue Copperliste gesetzt werden. Wenn allerdings eine ganz neue Liste gesetzt wird, bedeutet dies, daß die bisherige Darstellung verschwindet, da sie ja nicht mehr durch den Copper gesetzt wird. Welche Register vom Copper umgesetzt werden können, entnehmen Sie bitte dem Anhang. Der Copper versteht drei Befehle, aus denen eine Copperliste zusammengesetzt wird. Diese Befehle sind:

WAIT: Wartet, bis der Rasterstrahl eine bestimmte Bildschirmposition erreicht hat. Diese Position kann auch in X-Richtung bestimmt werden.

MOVE: Setzt ein Chip-Register auf einen spezifizierten Wert.

SKIP: Überspringt den nächsten Copperbefehl, wenn eine spezifizierte Rasterstrahlposition schon erreicht wurde.

Jeder Befehl besteht aus zwei 16-Bit-Worten. Die Copperliste wird sequentiell gelesen, wobei die zwei Befehlsworte immer zusammen gelesen werden.

Der MOVE-Befehl hat folgende Syntax:

Erstes Befehlswort:

Bit 0 Ist immer auf 0 zu setzen.

Bit 8–1 Enthält die Registeradresse des Chip-Registers, das gesetzt werden soll. Das Color0-Register hat beispielsweise die Adresse $DFF180, es ist der Wert $180 einzutragen.

Bit 15–9 Diese Bits werden nicht benutzt, sollten aber auf 0 gesetzt werden.

Zweites Befehlswort:

Bit 15–0 Diese 16 Bits enthalten den Wert, der in das Chip-Register, das im ersten Befehlswort spezifiziert wurde, geschrieben werden soll.

Der WAIT-Befehl hat folgende Syntax:

Erstes Befehlswort:

Bit 0 Ist immer auf 1 zu setzen.

Bit 15–8 Vertikale Rasterstrahlposition, auf die gewartet werden soll.

Bit 7–1 Horizontale Rasterstrahlposition, auf die gewartet werden soll.

Zweites Befehlswort:

Bit 0 Ist immer auf 0 zu setzen.

Bit 15 Blitter-Finished-Disable-Bit. Wenn über den Copper Blitterregister gesetzt werden sollen, muß erst gewartet werden, bis dieser seine Operationen beendet hat. Um auf den Blitter zu warten, muß dieses Bit auf 0 gesetzt werden. Normalerweise ist es auf 1 gesetzt.

Bit 14–8 Diese Bits bestimmen, welche Bits der vertikalen Position beim Vergleich mit der Rasterstrahlposition zu verwenden sind. Soll ein Bit verwendet werden, so ist hier das entsprechende Bit zu setzen.

Bit 7–1 Für diese Bits gilt das Gleiche, wie für die Bits 14 bis 8, nur für die horizontale Position.

Der SKIP-Befehl hat folgende Syntax:

Erstes Befehlswort:

Bit 0 Ist immer auf 1 zu setzen.

Bit 15–8 Vertikale Rasterstrahlposition, auf die gewartet werden soll.

Bit 7–1 Horizontale Rasterstrahlposition, auf die gewartet werden soll.

Zweites Befehlswort:

Bit 0 Ist immer auf 1 zu setzen (einziger Unterschied zu WAIT).

Bit 15 Blitter-Finished-Disable-Bit. Wenn über den Copper Blitterregister gesetzt werden sollen, muß erst gewartet werden, bis dieser seine Operationen beendet hat. Um auf den Blitter zu warten, muß dieses Bit auf 0 gesetzt werden. Normalerweise ist es auf 1 gesetzt.

Bit 14–8 Diese Bits bestimmen, welche Bits der vertikalen Position beim Vergleich mit der Rasterstrahlposition zu verwenden sind. Soll ein Bit verwendet werden, so ist hier das entsprechende Bit zu setzen.

Bit 7–1 Für diese Bits gilt das Gleiche, wie für die Bits 14 bis 8, nur für die horizontale Position.

Die Werte für die horizontalen Rasterstrahlpositionen können Werte zwischen $00 und $E2 annehmen. Das bedeutet, es können alle 4 LoRes- oder alle 8 HiRes-Pixel abgefragt werden.

Für die vertikale Position stehen Werte zwischen $00 und $FF, also 255, zur Verfügung. Da der Screen aber 262 Zeilen besitzt, gibt es Probleme, wenn die untersten Zeilen per Copper angesprochen werden sollen. Hier eine Lösungsmöglichkeit:

- Warten, bis Zeile 255 erreicht.

- Dann die folgenden Zeilen als Zeilen 0 bis 6 ansprechen.

Hat man eine Copperliste erstellt, so trägt man den Zeiger auf diese Liste entweder in die Register COP1LCH/COP1LCL oder aber in die Register COP2LCH/COP2LCL ein. Anschließend muß noch ein beliebiger Wert in Register COPJMP1 oder in COPJMP2 geschrieben werden. COPJMP1 bewirkt den Neustart des Coppers mit der Copperliste aus den Register COP1LCH/COP1LCL, während COPJMP2 einen Neustart mit der Liste aus COP2LCH/COP2LCL bewirkt.

Wie man eine Copperliste erstellt, ersehen Sie aus den folgenden Programmen.

```
 1 /*******************************
 2
 3 1. Cooper-Demonstration
 4 last update 16/02/88
 5 von Frank Kremser und Jörg Koch
 6 © Markt & Technik 1988
 7
 8 /*******************************
 9
10 Diese Demonstration ändert die Copper-Liste so ab, daß die Workbench
11 Flagge zeigt. Zusätzlich werden noch die Zeichenfarben geändert.
12
13 /*******************************
14
15 #include <exec/types.h>              /* Include-Files laden */
16 #include <exec/tasks.h>
17 #include <exec/memory.h>
18 #include <exec/interrupts.h>
19 #include <exec/execbase.h>
20 #include <exec/io.h>
21 #include <exec/libraries.h>
22 #include <exec/devices.h>
23 #include <exec/ports.h>
24 #include <exec/lists.h>
25 #include <exec/nodes.h>
26 #include <graphics/gfxmacros.h>
```

```
27 #include <graphics/copper.h>
28 #include <graphics/view.h>
29 #include <hardware/custom.h>
30
31
32 struct IntuitionBase * IntuitionBase; /* Zeiger für Libraries */
33 struct GfxBase *GfxBase;
34
35 extern struct Custom custom;     /* Externe Structure bereitstellen */
36                                     /* Sie enthält eine Vielzahl von */
37                                                 /* Systemvariablen */
38                                      /* siehe in hardware/custom.h */
39
40 main() /* HAUPTPROGRAMM */
41 {
42  struct UCopList *clist;
43  struct View *view;
44
45  GfxBase = OpenLibrary ("graphics.library",0); /* Libraries öffnen */
46  IntuitionBase = OpenLibrary(»intuition.library«,0);
47
48                  /* Speicher für eigene Copper-Liste bereitstellen */
49  clist = AllocMem(sizeof(struct UCopList), MEMF_PUBLIC:MEMF_CLEAR);
50  view = ViewAddress();
51  view->View->UCopIns = clist; /* Neue Copper-Liste eintragen */
52
53           /* Neue Copper-Liste erstellen */
54  CWAIT(clist,0,0);  /* Warten, bis Copper oberste Bildpos. erreicht */
55  CMOVE(clist,custom.color[0],0x000); /* Dann Farben ändern */
56  CMOVE(clist,custom.color[1],0x777);
57  CMOVE(clist,custom.color[3],0xFFF);
58
59  CWAIT(clist,85,0);     /* Warten, bis Copper Zeile 85 erreicht */
60  CMOVE(clist,custom.color[0],0xF00); /* Dann Farben ändern */
61  CMOVE(clist,custom.color[1],0xFFF);
62  CMOVE(clist,custom.color[3],0x000);
63
64  CWAIT(clist,171,0); /* Warten, bis Copper Zeile 171 erreicht */
65  CMOVE(clist,custom.color[0],0xFE0); /* Dann Farben ändern */
66  CMOVE(clist,custom.color[1],0x000);
67  CMOVE(clist,custom.color[3],0xF00);
68  CEND(clist);       /* Eintragen, daß das Ende der Liste erreicht ist */
69
70  RethinkDisplay(); /* Neue Displaydaten zur Darstellung bringen */
71
72  CloseLibrary(IntuitionBase); /* Libraries schließen */
73  CloseLibrary(GfxBase);
74 }
```

```
/*****************************

2.Copper-Demonstration
last update 16/02/88
von Frank Kremser und Jörg Koch
© Markt & Technik 1988

*****************************

Diese Demonstration zeigt ein Rechteck, das sich laufend in der Größe
ändert. Dabei wird aber nicht etwa in den Screen gezeichnet, sondern
es wird an den entsprechenden Stellen die Hintergrundfarbe geändert.
Diese Aufgabe übernimmt der Copper.

*****************************/

#include <exec/types.h>        /* Include-Files laden */
#include <exec/tasks.h>
#include <exec/memory.h>
#include <exec/interrupts.h>
#include <exec/execbase.h>
#include <exec/io.h>
#include <exec/libraries.h>
#include <exec/devices.h>
#include <exec/ports.h>
#include <exec/lists.h>
#include <exec/nodes.h>
#include <graphics/gfxmacros.h>
#include <graphics/copper.h>
#include <graphics/regions.h>
#include <graphics/gels.h>
#include <graphics/gfxbase.h>
#include <graphics/gfx.h#
#include <graphics/clip.h>
#include <graphics/view.h>
#include <graphics/rastport.h>
#include <graphics/layers.h>
#include <intuition/intuition.h>
#include <hardware/custom.h>
#include <hardware/blit.h>

struct IntuitionBase *IntuitionBase; /* Zeiger für Library-Pointer */
struct GfxBase *GfxBase;

extern struct Custom custom; /* Externe Structure, siehe 1. Copperdemo */

struct NewWindow nw = /* NewWindow-Structure für eigenes Window */
 {
  0,
  0,
```

```
  316,
  10,
  2,
  1,
  CLOSEWINDOW,
  WINDOWDRAG|WINDOWCLOSE|SMART_REFRESH,
  NULL,
  NULL,
  "Copperdemo",
  NULL,
  NULL,
  0,
  0,
  0,
  0,
  WBENCHSCREEN
};

main() /* HAUPTPROGRAMM */
{
 struct Window *window;
 struct ViewPort *vp;
 struct UCopList *ucop[26];

 UWORD i,j,w = 0;

 GfxBase=OpenLibrary("graphics.library",0); /* Libraries öffnen */
 IntuitionBase=OpenLibrary("intuition.library",0);

 window=OpenWindow(&nw); /* Window öffnen */
 vp=ViewPortAddress(window); /* und Viewport ermitteln */

 for(j=10;j<=35;j++) /* 26 Copperlisten für die verschiedenen */
   {                 /* Größen des Rechteckes erstellen */
    ucop[j-10]=AllocMem(sizeof(struct UCopList),MEMF_CHIP|MEMF_CLEAR);
    for (i=(j*2);i>0;i--)       /* Speicher für die Copperliste */
     {
      CWAIT(ucop[j-10],127-i,140-(j*2));
      CMOVE(ucop[j-10],custom.color[0],0x0F00);
      CWAIT(ucop[j-10],127-i,140+(j*2));
      CMOVE(ucop[j-10],custom.color[0],0x0000);
     }
    for (i=0;i<(j*2);i++)
    {
     CWAIT(ucop[j-10],127+i,140-(j*2));
     CMOVE(ucop[j-10],custom.color[0],0x0F00);
     CWAIT(ucop[j-10],127+i,140+(j*2));
     CMOVE(ucop[j-10],custom.color[0],0x0000);
     }
    CEND(ucop[j-10]); /* Copperliste beenden */
```

```
101     }
102
103    j = 0;
104    while(!GetMsg(window->UserPort)) /* Solange, bis Closegadget */
105     {
106      WaitTOF();       /* Warten, bis oberer Bildschirmrand erreicht */
107      if(w==0) j+=3;
108      if(w==1) j-=3;
109      if(j==24) w = 1;
110      if(j==0) w = 0;
111
112      vp->UCopIns=ucop[j]; /* dann neue Copperliste setzen */
113      RethinkDisplay();
114     }
115    for(j=0;j<=25;j++) /* Alle 26 Copperlisten löschen */
116     {
117      vp->UCopIns=ucop[j];
118      FreeVPortCopLists(vp);
119      RemakeDisplay();
120     }
121    CloseWindow(window); /* Window schließen */
122    CloseLibrary(IntuitionBase); /* Libraries schließen */
123    CloseLibrary(GfxBase);
124  }
125
126
```

```
  1 /*******************************
  2
  3 3.Copper-Demonstration
  4 last update 16/02/88
  5 von Frank Kremser und Jörg Koch
  6 © Markt & Technik 1988
  7
  8 *******************************
  9
 10 Diese Copper-Demonstration setzt laufend neue Copperlisten, so daß ein
 11 Screen mit laufenden Farben entsteht.
 12
 13 *******************************/
 14
 15 #include <exec/types.h>       /* Include-Files laden */
 16 #include <exec/tasks.h>
 17 #include <exec/memory.h>
 18 #include <exec/interrupts.h>
 19 #include <exec/execbase.h>
 20 #include <exec/io.h>
 21 #include <exec/libraries.h>
 22 #include <exec/devices.h>
 23 #include <exec/ports.h>
```

```
24 #include <exec/lists.h>
25 #include <exec/nodes.h>
26 #include <graphics/gfxmacros.h>
27 #include <graphics/copper.h>
28 #include <graphics/regions.h>
29 #include <graphics/gels.h>
30 #include <graphics/gfxbase.h>
31 #include <graphics/gfx.h>
32 #include <graphics/clip.h>
33 #include <graphics/view.h>
34 #include <graphics/rastport.h>
35 #include <graphics/layers.h>
36 #include <intuition/intuition.h>
37 #include <hardware/custom.h>
38 #include <hardware/blit.h>
39
40 struct IntuitionBase *IntuitionBase; /* Zeiger für Libraries */
41 struct GfxBase *GfxBase;
42
43 extern struct Custom custom; /* Externe Structure, siehe 1.Copperdemo */
44
45 UWORD colors[] = /* Farben, die zur Animation verwendet werden*/
46  {
47   0xce3,0xae3,0x8e3,0x7e3,0x5e3,0x4e3,0x3e4,0x3e5,0x3e7,0x3e8,
48   0x3ea,0x3eb,0x3ec,0x3ee,0x3de,0x3ce,0x3ae,0x39e,0x37e,0x34e,
49   0x33e,0x43e,0x63e,0x73e,0x83e,0xa3e,0xb3e,0xc3e,0xe3e,0xe3d,
50   0xe3b,0xe3a,0xe39,0xe37,0xe36,0xe34,0xe33,0xe53,0xe63,0xe83,
51   0xe93,0xea3,0xeb3,0xec3,0xee3,0xde3,0xbe3,0x8e3,0x7e3,0x4e3,
52   0x3e4,0x3e5,0x3e6,0x3e8,0x3e9,0x3ea,0x3ec,0x3ed,0x3ee,0x3de,
53   0x3be,0x3ae,0x38e,0x37e
54  };
55
56 struct NewWindow nw = /* NewWindow-Structure für eigenes Window */
57  {
58   0,
59   0,
60   640,
61   10,
62   2,
63   1,
64   CLOSEWINDOW,
65   WINDOWCLOSE | SMART_REFRESH,
66   NULL,
67   NULL,
68   "Dies ist eine Copper-Demonstration, die den Workbench-Screen 'animiert'",
69   NULL,
70   NULL,
71   0,
72   0,
73   0,
```

```
 74   0,
 75 WBENCHSCREEN
 76  };
 77
 78
 79 main()
 80 {
 81  struct Window *window;
 82  struct ViewPort *vp;
 83  struct UCopList *ucop;
 84
 85  void *dspins,*sprins,*clrins;
 86
 87  UWORD i, j=1;
 88
 89  GfxBase=OpenLibrary("graphics.library",0); /* Libraries öffnen */
 90  IntuitionBase=OpenLibrary("intuition.library",0);
 91
 92  window=OpenWindow(&nw); /* Window öffnen */
 93  vp=ViewPortAddress(window); /* und Viewport ermitteln */
 94
 95  while(!GetMsg(window->UserPort)) /* Solange, bis CloseGadget */
 96   {                                    /* Speicher für Copperliste *
 97    ucop=AllocMem(sizeof(struct UCopList),MEMF_CHIP | MEMF_CLEAR);
 98
 99    for(i=0;i<64;i++) /* Copperliste mit 64 ver. Farben erstellen */
100     {
101      CWAIT(ucop,i*4,0);
102      CMOVE(ucop,custom.color[0],colors[(i+j)%64]);
103     }
104
105    CEND(ucop);
106    j++;
107                    /* Alte Instruktionslisten sichern */
108    dspins=vp->DspIns; sprins=vp->SprIns; clrins=vp->ClrIns;
109
110    Forbid(); /* Intuition vorübergehend 'abschalten' */
111
112    vp->DspIns=vp->SprIns=vp->ClrIns=0; /* Listen zurücksetzen */
113    FreeVPortCopLists(vp); /* Alte Copperliste löschen */
114
115                 /* Alte Instruktionslisten wieder setzen */
116    vp->DspIns=dspins; vp->SprIns=sprins; vp->ClrIns=clrins;
117    vp->UCopIns=ucop; /* Neue Copperliste setzen */
118
119    Permit(); /* Intuition wieder einschalten */
120    RethinkDisplay(); /* Änderungen in die Darstellung übernehmen */
121   }
122
123  /* Alle Listen löschen und neu erstellen */
```

```
124  FreeVPortCopLists(vp);
125  RemakeDisplay();
126
127  CloseWindow(window); /* Window schließen */
128  CloseLibrary(IntuitionBase); /* Libraries schließen */
129  CloseLibrary(GfxBase);
130 }
```

```
 1 ;*******************************
 2 ;
 3 ; 1.Copper-Demonstration
 4 ; last update 10/03/88
 5 ;von Frank Kremser und Jörg Koch
 6 ; © Markt & Technik 1988
 7 ;
 8 ;*******************************
 9 ;
10 ;Diese Demonstration setzt eine neue Copperliste, so daß ihr
11 ;Amiga 'Flagge' zeigt.
12 ;
13 ;*******************************
14
15       DMACON = $dff096
16
17       ExecBase     = 4
18       Permit       = -138
19       Forbid       = -132
20       OpenLibrary  = -408
21       CloseLibrary = -414
22
23            move.l  ExecBase,a6
24            lea     GfxName,a1        ;GfxLibrary öffnen
25            jsr     OpenLibrary(a6)
26            move.l  d0,GfxBase
27
28            move.l  ExecBase,a6
29            jsr     Forbid(a6)        ;Multitasking abschalten
30
31
32            lea.l   $50000,a0         ;Bitplane ab $50000
33            move.l  #6645,d0          ;6645 Longwords
34 clear:     clr.l   (a0)+             ;löschen
35            dbf     d0,clear          ;dekrementiere, teste
36                                      ;d0 = 0, nein dann clear
37
38
39            move.l  GfxBase,a0
40            add.l   #$32,a0           ;Zeiger auf LOFlist
41
```

```
42              move.w  #$0080,DMACON   ;Copper DMA stoppen
43              move.l  (a0),OldCopper  ;Zeiger auf alte Copperliste
44              move.l  #Copper1,(a0)   ;Neue Liste setzen
45              move.w  #$80A0,DMACON   ;und Copper starten
46
47 wait:        btst    #6,$bfe001      ;Wurde Maustaste gedrückt,
48              bne     wait            ;Wenn nein, dann weiter warten
49
50              move.l  GfxBase,a0
51              add.l   #$32,a0         ;Zeiger auf LOFlist
52
53              move.w  #$0080,DMACON   ;Copper und Sound abschalten
54              move.l  OldCopper,(a0)  ;Alte Copperliste setzen
55              move.w  #$8180,DMACON   ;Copper einschalten
56
57              move.l  ExecBase,a6
58              jsr     Permit(a6)      ;Multitasking einschalten
59
60              move.l  GfxBase,a1      ;GfxLibrary schließen
61              jsr     CloseLibrary(a6)
62              rts                     ;Rückkehr
63 Copper1:     dc.l    $00e00005       ;BitPlanel Zeiger setzen
64              dc.l    $00e20000
65              dc.l    $01800000       ;Color0 auf schwarz setzen
66              dc.l    $7001fffe       ;Warten, bis Zeile $70 erreicht
67              dc.l    $01800f00       ;Color0 auf rot setzen
68              dc.l    $d001fffe       ;Warten, bis Zeile $d0 erreicht
69              dc.l    $01800ff0       ;Color0 auf gelb setzen
70              dc.l    $fffffffe       ;Endekennzeichnung für Copperliste
71
72 GfxName:     dc.b    'graphics.library',0
73 even
74 GfxBase:     blk.l   1,0
75 OldCopper:
76              blk.l   1,0
77
```

```
 1 ;*******************************
 2 ;
 3 ;2.Copper-Demonstration
 4 ;last update 10/03/88
 5 ;von Frank Kremser und Jörg Koch
 6 ;© Markt & Technik 1988
 7 ;
 8 ;*******************************
 9 ;
10 ;Diese Demonstration setzt laufend neue Copperlisten, so daß der
11 ;Eindruck entsteht, die Farben wandern. Dieser Programmteil
12 ;taucht auch in dem Programm 'Resetfest.s' auf.
13 ;
14 ;*******************************
15
16       DMACON      =$dff096
17       ExecBase    =4
18       Permit      =-138
19       Forbid      =-132
20       OpenLibrary =-408
21       CloseLibrary=-414
22
23             move.l  ExecBase,a6
24             lea     GfxName,a1        ;GfxLibrary öffnen
25             jsr     OpenLibrary(a6)
26             move.l  d0,GfxBase
27
28             move.l  ExecBase,a6
29             jsr     Forbid(a6)        ;Multitasking abschalten
30
31
32             lea.l   $50000,a0         ;Bitplane ab $50000
33             move.l  #6645,d0          ;6645 Longwords
34 clear:      clr.l   (a0)+             ;löschen
35             dbf     d0,clear          ;dekrementiere, teste
36                                       ;d0=0, nein dann clear
37
38             lea.l   coltab(pc),a1     ;Zeiger auf Farbtabelle;
39             bsr     copperinit        ;Copperliste initialisieren
40
41             move.l  GfxBase,a0
42             add.l   #$32,a0           ;Zeiger auf LOFlist
43
44             move.w  #$0080,DMACON     ;Copper DMA stoppen
45             move.l  (a0),OldCopper    ;Zeiger auf alte Copperliste
46             move.l  #$55000,(a0)      ;Neue Liste setzen
47             move.w  #80A0,DMACON      ;und Copper starten
48
49             move.l  #0,a2             ;Initialisierung von a2
50 main:       btst    #6,$bfe001        ;Wurde Maustaste gedrückt,
```

```
 51            beq     ende                ;Wenn ja, dann Programm beenden
 52
 53            lea.l   coltab(pc),a1       ;Zeiger auf Colortabelle
 54            add.l   #1,a2               ;a2 um 1 erhöhen,
 55            cmp.l   #32,a2              ;vergleichen ob gleich 32
 56            bne     cont                ;wenn nicht, dann fortfahren
 57            move.l  #0,a2               ;wenn nein, dann a2 zurücksetzen
 58 cont:      adda.l  a2,a1               ;a2 zu a1 hinzuaddieren
 59            bsr     copperinit          ;und neue Copperliste erstellen
 60            move.l  #$ff,d3             ;Wert für Warteschleife setzen
 61 loop:      tst.l   (a6)                ;Dummybefehl zur Verzögerung
 62            dbra    d3,loop             ;d3 erniedrigen und ggf. fortfahren
 63            bra     main                ;Das ganze nocheinmal
 64
 65 ende:      move.l  GfxBase,a0
 66            add.l   #$32,a0             ;Zeiger auf LOFlist
 67
 68            move.w  #$0080,DMACON       ;Copper und Sound abschalten
 69            move.l  OldCopper,(a0)      ;Alte Copperliste setzen
 70            move.w  #$8180,DMACON       ;Copper einschalten
 71
 72            move.l  ExecBase,a6
 73            jsr     Permit(a6)          ;Multitasking einschalten
 74
 75            move.l  GfxBase,a1          ;GfxLibrary schließen
 76            jsr     CloseLibrary(a6)
 77            rts                         ;Rückkehr
 78
 79 copperinit:
 80            clr.l   d1                  ;d1 löschen
 81            lea.l   $55000,a0           ;Startadresse für
 82                                        ;Copperliste
 83            move.l  #$00e00005,(a0)+    ;BitPlanel Zeiger setzen
 84            move.l  #$00e20000,(a0)+
 85            move.l  #$01800f00,(a0)+    ;Color0 auf rot setzen
 86            move.l  #80,d0              ;Ab Zeile 80 Farbrotation
 87 loop1:     move.b  d0,(a0)+            ;Wait - VP
 88            move.b  #1,(a0)+            ;Wait - HP + Bit 0 gesetzt
 89            move.l  #$fffe0180,(a0)+    ;Wait + Color0
 90            clr.l   d6
 91            move.b  (a1,d1),d6          ;neue Farbe aus Tabelle holen
 92            eori    #$0ff0,d6           ;bearbeiten
 93            move    d6,(a0)+            ;und in Copperliste für Color0 eintr.
 94            addq    #1,d1               ;Farbindex um 1 erhöhen
 95            cmpi    #32,d1              ;Wenn 32 Farben gezeigt,
 96            bne     loop2
 97            clr.l   d1                  ;dann Index zurücksetzen
 98 loop2:     add.l   #1,d0               ;Zeilenindex um 1 erhöhen
 99            cmp.l   #255,d0             ;und mit 255 vergleichen
100            beq     loop3
```

```
101             bra     loop1
102 loop3:      move.l  #$ff01fffe,(a0)+ ;warten, bis Zeile 255 erreicht
103             move.l  #$01800f00,(a0)+ ;dann Color0 auf rot setzen
104             move.l  #$fffffffe,(a0)+ ;Endekennzeichnung für Copper
105             rts
106
017 ;Farbtabelle, doppelt so lang, wie Farbanzahl,
108 ;da Rotation
109 even
110 coltab:     dc.b    $00,$10,$20,$30,$40,$50,$60,$70,$80,$90
111             dc.b    $a0,$b0,$c0,$d0,$e0,$f0,$f0,$e0,$d0
112             dc.b    $c0,$b0,$a0,$90,$80,$70,$60,$50,$40,$30
113             dc.b    $20,$10,$00,$00,$10,$20,$30,$40,$50,$60
114             dc.b    $70,$80,$90,$a0,$b0,$c0,$d0,$e0,$f0,$f0
115             dc.b    $e0,$d0,$c0,$b0,$a0,$90,$80,$70,$60,$50
116             dc.b    $40,$30,$20,$10,$00
117
118 GfxName:    dc.b    'graphics.library',0
119 even
120 GfxBase:    blk.l   1,0
121 OldCopper:
122             blk.l   1,0
```

4.1.4: Der Blitter

Der Blitter ist ein sehr machtvoller Speichermanipulations-Komplex, der hauptsächlich für Grafikanwendungen verwendet wird. Er benötigt bis zu vier DMA-Kanäle gleichzeitig, um die Daten zu manipulieren. Diese Manipulationen laufen durch den Blitter erheblich schneller ab, als sie mit dem MC68000 programmiert werden könnten. Der Blitter kann

- Speicherbereiche kopieren,
- rechteckige Bildbereiche kopieren,
- bis zu drei verschiedene Speicherbereiche miteinander logisch verknüpfen usw.

Die drei DMA-Kanäle für die Speicherbereiche, die miteinander verknüpft werden können, werden mit Source-A, -B und -C bezeichnet. Der vierte DMA-Kanal, über den die verknüpften Daten geschrieben werden, wird mit Destination-D benannt. Damit nicht alle vier Kanäle gleichzeitig verwendet werden müssen, können sie mit den Bits 8 bis 11 des BLTCON0-Registers einzeln aktiviert, bzw. deaktiviert werden.

Um nun einen Speicherbereich zu kopieren, setzt man die Register BLTAPTH/ BLTAPTL auf die Startadresse des zu kopierenden Speicherbereiches und BLTDPTH/ BLTDPTL auf die Startadresse des Bereiches, in den kopiert werden soll. Anschließend setzt man noch das Register BLTSIZE, das unten noch beschrieben wird. Nun

braucht man nur noch die Kanäle für Source-A und Destination-D im BLTCON0-Register zu aktivieren und die Blitter-DMA zu starten. Schon wird der angegebene Speicherbereich kopiert.

Will man Datenbereiche manipulieren, so können sie entweder aufsteigend oder absteigend eingelesen werden. Dies ist beispielsweise von großer Bedeutung, wenn sich beim Datentransfer der Source-Bereich mit dem Destinations-Bereich überschneidet. Aufsteigend wird eingelesen, wenn Bit 1 des BLTCON1-Registers gelöscht ist.

Der Blitter kann lineare Speicherbereiche genauso bearbeiten, wie »rechteckige«. Grafiken sind beispielsweise rechteckig, sind aber im Speicher linear abgelegt. Um nun aus dieser Grafik einen rechteckigen Teilbereich herauszukopieren, muß zum einen in BLTxPTH/BLTxPTL die Startadresse des Teilbereiches angegeben sein und in BLTSIZE muß in den Bits 0 bis 5 die Breite des Bereiches und in den Bits 6 bis 15 die Höhe des Bereiches angegeben sein. Zudem muß man am Ende einer Zeile dieses Teilbereiches eine bestimmte Anzahl von Words hinzuaddieren, um auf die Startadresse der nächsten Zeile des Teilbereiches zu kommen. Dazu dienen die BLTxMOD-Register. Diese enthalten die Anzahl der Words, die hinzuaddiert werden sollen. Will man aber einen linearen Speicherbereich bearbeiten, so gibt man als Höhe oder als Breite den Wert 1 an, setzt den anderen Wert auf die Länge des Bereiches und setzt die BLTxMOD-Register auf 0.

Die oben beschriebenen Handlungen bewirken lediglich ein Kopieren eines Speicherbereiches. Der Blitter bietet aber noch die Möglichkeit, die Speicherbereiche, die mit Source-A bis -C bezeichnet sind, miteinander zu verknüpfen und dann zu speichern.

Um festzulegen, auf welche Weise die drei Speicherbereiche miteinander verknüpft werden sollen, müssen die Bits 0 bis 7 im BLTCON0-Register, die auch als LF-Kontroll-Byte bezeichnet werden, gesetzt werden.

Im folgenden werden die Source-A- bis -C-Bereiche nur noch mit A, B und C bezeichnet. Bei den logischen Verknüpfungen bedeutet ein Kleinbuchstabe, daß eine 0 logisch wahr ist, und ein Großbuchstabe, daß eine 1 logisch wahr ist. Beispielsweise ist der Ausdruck AbC nur dann logisch wahr, wenn ein Bit aus Source-A gleich 1, eines aus B gleich 0 und eines aus C gleich 1 ist. Dann wird auch das Bit in Destination-D auf 1 gesetzt. Die Bits des LF-Kontroll-Bytes haben folgende Bedeutungen:

Logik:	ABC	ABc	AbC	Abc	aBC	aBc	abC	abc
Bit:	7	6	5	4	3	2	1	0

Wird das LF-Byte auf %10000000 gesetzt, wird ein Bit in Destination-D nur dann auf 1 gesetzt, wenn die Bits in A, B und C gleich 1 sind. Wird das LF-Byte hingegen auf %10000001 gesetzt, so wird D nur dann gleich 1, wenn A, B und C alle gleich 1 oder gleich 0 sind.

Eine weitere Möglichkeit, die der Blitter zur Verfügung stellt, ist der *Barrel-Shifter.* Dieser Shifter ermöglicht es, die Daten, die über die Source-A- und -B-Kanäle gelesen werden, um 0 bis 15 Bits zu shiften. Hierzu wird nicht mehr Zeit benötigt, als ohne dieses Shiften, wodurch ein sehr schnelles Bit-Scrolling ermöglicht wird. Der Wert, um den »geshiftet« werden soll, ist für Source-A in den Bits 12 bis 15 von BLTCON0 zu setzen, und für Source-B in den Bits 12 bis 15 von BLTCON1.

Die nächste Funktion, die der Blitter noch zur Verfügung stellt, nennt sich *»Masking«.* Nicht immer sind die Speicherbereiche, die bearbeitet werden sollen, genau nach Words ausgerichtet. Um auch diese Bereiche bearbeiten zu können, bietet der Blitter die Möglichkeit, bis zu 16 Bits links und rechts auszumaskieren. BLTAFWM beinhaltet die 16 Bit, die links auszumaskieren sind. BLTALWM die, die rechts auszumaskieren sind. Beginnt eine Zeile beispielsweise mit %1100110011001100 und ist BLTAFWM auf %0000000011111111 gesetzt, so bleibt nur noch %0000000011001100 übrig. Das Gleiche gilt für den rechten Rand.

Der Blitter bietet auch die Möglichkeit, beliebig geformte Bereiche zu füllen. Dazu muß BLTxPTH/BLTxPTL auf die rechte, untere Ecke des Ausschnittes gesetzt werden, in dem sich der zu füllende Bereich befindet und BLTxMOD muß ebenfalls entsprechend gesetzt werden. Es wird lediglich ein Source-Kanal benötigt. Der Destination-D-Kanal muß auf die gleiche Adresse gesetzt werden, wie der Source-Kanal. Anschließend wird festgelegt, daß Ausschnitt abzählend durchlaufen werden soll, dazu Bit 1 von BLTCON1 auf 1 setzen. Nun muß noch angegeben werden, ob der Bereich außerhalb des umrahmten, oder der innerhalb gefüllt werden soll. Wird Bit 2 von BLTCON1 auf 0 gesetzt, wird innerhalb gefüllt.

```
 1 /*******************************
 2
 3 Blitter-Demonstration
 4 last update 16/02/88
 5 von Frank Kremser und Jörg Koch
 6 © Markt & Technik 1988
 7
 8 *******************************
 9
10 Diese Demonstration kopiert den Workbench-Screen laufend auf einen
11 eigenen Screen, wobei er ständig etwas verschoben kopiert wird.
12
13 *******************************/
14
15 #include <exec/types.h>          /* Include-Files laden */
16 #include <exec/tasks.h>
17 #include <exec/libraries.h>
18 #include <exec/memory.h>
19 #include <exec/devices.h>
```

```
20 #include <devices/keymap.h>
21 #include <graphics/copper.h>
22 #include <graphics/display.h>
23 #include <graphics/gfxbase.h>
24 #include <graphics/text.h>
25 #include <graphics/view.h>
26 #include <graphics/gels.h>
27 #include <graphics/regions.h>
28 #include <graphics/sprite.h>
29 #include <hardware/blit.h>
30 #include <intuition/intuition.h>
31 #include <intuition/intuitionbase.h>
32
33 struct  GfxBase         *GfxBase;           /* Lib Zeiger */
34 struct  IntuitionBase   *IntuitionBase;
35
36 struct  Screen *screen;        /* Screen-Structure-Zeiger */
37 struct  Window *window;
38 struct  RastPort *rpl, *rp2;
39
40 struct NewScreen ns =          /* Die New-Screen Structure */
41  {
42   0,                          /* Linke Ecke */
43   0,                          /* Obere Ecke */
44   640,                        /* Breite */
45   256,                        /* Hoehe */
46   2,                          /* Tiefe */
47   0,                          /* DetailPen */
48   1,                          /* BlockPen */
49   HIRES,                      /* ViewModes */
50   CUSTOMSCREEN,               /* Type */
51   NULL,
52   NULL,
53   NULL,
54   NULL
55  };
56
57 struct NewWindow nw =          /* NewWindow-Structure */
58  {
59   0,
60   0,
61   640,
62   10,
63   2,
64   1,
65   NULL,
66   NULL,
67   NULL,
68   NULL,
69   "Dies ist eine Blitter-Demonstration, die den Workbench-Screen 'animiert'",
```

```
 70   NULL,
 71   NULL,
 72   0,
 73   0,
 74   0,
 75   0,
 76   WBENCHSCREEN
 77  };
 78
 79
 80 main() /* HAUPTPROGRAMM */
 81 {
 82  int x,y = 0;
 83
 84  /* öffnen der Libs */
 85  if((IntuitionBase = (struct IntuitionBase *)
 86  OpenLibrary("intuition.library", 0)) == 0) exit();
 87
 88  if((GfxBase = (struct GfxBase *)
 89  OpenLibrary("graphics.library", 0)) == 0) exit();
 90                      /* Screen und Window öffnen */
 91  if((screen = (struct Screen*) OpenScreen(&ns)) == NULL) exit();
 92  if((window = OpenWindow(&nw)) == NULL) exit();
 93
 94  rpl = &screen->RastPort;
 95  rp2 = &window->WScreen->RastPort;
 96        /* RastPort des WBScreens ermitteln */
 97
 98  SetDrMd(rpl,JAM1);
 99  SetAPen(rpl,0);
100
101  ClipBlit(rp2,0,0,rpl,0,0,640,256,0xc0); /* Screen kopieren */
102  for(x=5;x<640; x+=5) /* Screen laufend etwas verschoben kopieren */
103   {
104    y+=2;
105    ClipBlit(rp2,0,0,rpl,x,y,640-x,256-y,0xc0);
106    RectFill(rpl,0,0,x,255); /* und freibleibende Flächen löschen */
107    RectFill(rpl,0,0,639,y);
108   }
109  ClipBlit(rp2,0,0,rpl,0,0,640,256,0xc0);
110  for(x=5;x<640; x+=5) /* Der gleiche Effekt mit ScrollRaster */
111  ScrollRaster(rpl,-5,-2,0,0,639,255);
112
113  CloseScreen(screen);                    /* Screen und Libs */
114  CloseWindow(window);
115
116  CloseLibrary(GfxBase);                  /* schliessen */
117  CloseLibrary(IntuitionBase);
118 }
119
```

```
 1 ;*******************************
 2 ;
 3 ; 1. Blitter - Demonstration
 4 ; last update 10/03/88
 5 ;von Frank Kremser und Jörg Koch ;
 6 © Markt & Technik 1988
 7 ;
 8 ;*******************************
 9 ;
10 ;Diese Demonstration scrollt einen Text von rechts nach links ein,
11 ;wobei der Befehl ScrollText verwendet wird. Dieser wiederrum
12 ;greift auf die Blitterfunktionen zurück.
13 ;
14 ;*******************************
15
16      scroll              =$55000; Zwischenspeicher fuer Laufschrift
17
18      Text                =-60
19      SetFont             =-66
20      CloseFont           =-78
21      Move                =-240
22      InitBitMap          =-390
23      InitRastPort        =-198
24      ScrollRaster        =-396
25      ClearScreen         =-48
26
27      AllocAbs            =-204
28      OpenLibrary         =-408
29      CloseLibrary        =-414
30      Forbid              =-132
31      Permit              =-138
32
33      OpenFont            =-30
34
35      ExecBase            =$04
36
37           movem.l d0-d7/a0-a6,-(a7)  ;Register retten
38           move.b  #$02,rows
39           move.l  ExecBase,a6
40           lea     $50000,a1          ;ab Adresse $50000
41           move.l  #8000,d0           ;8000 Byte als belegt kennzeichnen
42           jsr     AllocAbs(a6)
43
44           lea     gfxname,a1
45           jsr     OpenLibrary(a6)    ;GfxLibrary öffnen
46           move.l  d0,gfxbase         ;Basisadresse merken
47
48           lea     diskfontname,a1
49           jsr     OpenLibrary(a6     ;DiskfontLibrary öffnen
50           move.l  d0,fontlbase       ;Basisadresse merken
```

```
51
52              move.l  fontlbase,a6
53              lea     textattr,a0
54              jsr     OpenFont(a6)        ;Zeichensatz laden
55              move.l  d0,fontbase         ;Basisadresse Zeichensatz merken
56
57              lea     waitab,a0           ;Copperliste mit WAIT's ergänzen
58              move.l  #$df09,d1           ;ab Zeile $df spalte $09
59              move.l  #13,d0              ;14 WAIT's einfügen
60
61 coloop2:     move.w  d1,(a0)+            ;WAIT in Copperliste schreiben
62              move.w  #$fffe,(a0)+
63              move.w  #$0182,(a0)+        ;Vordergrundfarbe
64              move.w  #$0fff,(a0)+        ;auf schwarz setzen
65              add.w   #$0100,d1           ;WAIT-Wert um eine Zeile erhöhen
66              dbra    d0,coloop2          ;schon alle WAIT's eingefügt?
67
68              move.l  ExecBase,a6
69              jsr     Forbid(a6)          ;Multitasking abschalten
70
71              move.l  gfxbase,a0          ;Zur Basisadresse von GfxLibrary
72              add.l   #$32,a0             ;$32 addieren => Anfangsadr.
73                                          ;der alten CopperListe
74              move.w  #$0080,$dff096      ;Copper DMA stoppen
75              move.l  (a0),oldcopper      ;Alte Adresse merken
76              move.l  #newcopper,(a0)     ;Neue CopperListe setzen
77              move.w  #$8080,$dff096      ;Copper DMA starten
78
79              move.l  gfxbase,a6
80              lea     bitmap,a0           ;Zeiger auf Structure
81              move.l  #$01,d0             ;1 BitPlane
82              move.l  #352,d1             ;Breite 352 Punkte
83              move.l  #200,d2             ;Höhe 15 Zeilen
84              jsr     InitBitMap(a6)      ;Default Werte schreiben
85              move.l  #$50000,planel      ;Anfangsadresse des Grafik-
86                                          ;speichers ergänzen
87              lea     RastPort,a1         ;Zeiger auf Structure
88              jsr     InitRastPort(a6)    ;Rastport DefaultWerte
89              move.l  #bitmap,r_bitmap    ;Bitmap-Structure in Rastport eintr.
90              lea     RastPort,a1         ;Neuen Zeichensatz
91              move.l  fontbase,a0         ;installieren
92              jsr     SetFont(a6)
93
94              lea     RastPort,a1         ;Grafikspeicher löschen
95              jsr     ClearScreen(a6)
96
97              move.l  #scrollmsg,mesptr   ;Anfangsadresse d.ScrollText
98
99              move.l  $6c,oldirq          ;Alten IRQ-Vektor retten
100             move.l  #newirq,$6c         ;IRQ-Vektor auf eigene Routine setzen
```

```
101
102             bra.l   wait                    ;Weiter zur Mausabfrage
103
104 newirq:     movem.l d0-d7/a0-a6,-(sp)  ;Register retten
105 ;scrolltext im Grafikspeicher um einen Punkt nach links scrollen
106             move.l  gfxbase,a6
107             lea     RastPort,a1         ;RastPort, der gescrollt werden soll
108             move.l  #0,d2               ;linke, obere Koordinate des
109             move.l  #179,d3             ;zu verschiebenden Rechtecks
110             move.l  #352,d4             ;rechte, untere Koordinate
111             move.l  #198,d5             ;des Rechtecks
112             move.l  #$01,d0             ;1 Punkt in x-Richtung verschieben
113             clr.l   d1                  ;keinen Pkt in y-Richtung verschieben
114             jsr     ScrollRaster(a6)    ;Scrolling über Blitter ausführen
115
116             sub.b   #$01,rows           ;schon 1 Zeichen (16 Punkte)
117             bne.s   continuel           ;gescrollt?
118             move.b  #16,rows            ;Wenn ja, dann
119             bsr.s   PrintChar           ;neues Zeichen ausgeben
120 continuel:
121             lea     coltab,a0           ;Farbtabelle verschieben
122             lea     coltab+2,a1
123             move.w  #28,d0
124             move.w  coltab,d1
125 verloopl:
126             move.w  (a1)+,(a0)+
127             dbra    d0,verloopl
128             move.w  d1,coltab+56
129
130             move.l  #13,d0              ;14 Farben in Copperliste
131             lea     coltab,a0           ;schreiben
132             lea     waitab+6,a1
133 coloopl:    move.w  (a0)+,(a1)
134             add.l   #$08,a1             ;Abstand zu nächstem WAIT 4 Worte
135             dbra    d0,coloopl
136
137             movem.l (sp)+,d0-d7/a0-a6 ;Register zurückholen
138             dc.w    $4ef9               ;Interruptroutine mit Sprung zu
139 oldirq:     dc.l                        ;altem IRQ-Vektor beenden
140
141 printchar:
142             move.l  gfxbase,a6
143             lea     RastPort,a1         ;vorhandenes Bitmuster im Scroll-
144             jsr     ClearScreen(a6)     ;zwischenspeicher löschen
145             lea     RastPort,a1         ;im Zwischenspeicher Grafikcursor
146             move.l  #320,d0             ;nach x-position 320
147             move.l  #193,d1             ;und y-pos      14
148             jsr     move(a6)            ;bewegen
149             lea     RastPort,a1         ;in Zwischenspeicher
150             move.l  mesptr,a0           ;Zeichen ab Adresse (mesptr)
```

```
151             move.l  #1,d0               ;1 zeichen
152             jsr     text(a6)            ;an Position des Grafikcursors printen
153             add.l   #$01,mesptr         ;1 zum Textzeiger addieren
154             cmp.l   #ende,mesptr        ;Ende des Textes erreicht?
155             bne.s   return              ;Wenn nein, dann zurück
156             move.l  #scrollmsg,mesptr   ;Ansonsten Zeiger zurücksetzen
157 return:    rts
158
159 wait:       btst    #6,$bfe001          ;Warten, bis linke Maustaste gedrückt
160             bne.s   wait
161             move.l  oldirq,$6c          ;alten IRQ-Vektor zurückschreiben
162             move.l  gfxbase,a6
163             move.l  fontbase,a1
164             jsr     CloseFont(a6)       ;Font schließen
165             move.l  execbase,a6
166             move.l  fontlbase,a1
167             jsr     CloseLibrary(a6)    ;DiskfontLibrary schließen
168             move.l  gfxbase,a1
169             jsr     CloseLibrary(a6)    ;GfxLibrary schließen
170             move.l  gfxbase,a0          ;zur Basisadresse der GfxLibrary $32
171             add.l   #$32,a0             ;addieren erg. Zeiger auf Copperliste
172             move.w  #$0080,$dff096      ;Copper DMA stoppen
173             move.l  oldcopper,(a0)      ;alte CopperListe setzen
174             move.w  #$8080,$dff096      ;Copper DMA starten
175             move.l  ExecBase,a6
176             jsr     Permit(a6)          ;Multitasking einschalten
177             movem.l (a7)+,d0-d7/a0-a6   ;Register zurücksetzen
178             rts                         ;Rückkehr
179 newcopper:                              ;Neue Copperliste
180             dc.w    $0180,$0000
181             dc.w    $0182,$0ddd
182             dc.w    $008e,$2c81
183             dc.w    $0090,$f4c1
184             dc.w    $0092,$0038
185             dc.w    $0094,$00d0
186             dc.w    $0108,$0004
187             dc.w    $010a,$0004
188             dc.w    $0102,$0000
189             dc.w    $0104,$0000
190             dc.w    $0100,$1200
191             dc.w    $00e0,$0005
192             dc.w    $00e2,$0000
193
194 waitab:     blk.w   56,0                ;Speicher für WAIT Befehle
195             dc.w    $ffff,$fffe         ;Ende der Copperliste
196
197 even
198 scrollmsg:
199             dc.b    "Dies ist eine Blitterdemo, die ScrollRaster verwendet",0
200 even
```

```
ende:       dc.b
even
gfxbase:    dc.l    0
bitmap:     blk.w   4,0
planel:     blk.l   10,0
rastport:
            blk.l   1,0
r_bitmap:
            blk.l   26,0
oldcopper:
            dc.l    0
gfxname:    dc.b    "graphics.library",0
diskfontname:
            dc.b    "diskfont.library",0
even
fontname:           ;Name des Zeichensatzes
            dc.b    "opal.font",0
even
textattr:                           ;Struktur des Zeichensatzes
                                    ;wird bei OPENFONT übergeben
            dc.l    fontname        ;Zeiger auf Fontname
            dc.w    12              ;Höhe der Schrift in Punkten
            dc.w    0               ;Flags
fontlbase:
            dc.l    0
fontbase:
            dc.w    0
rows:       dc.b    0
even
mesptr:     dc.l    0
                                                ;Laufschriftfarben
coltab:     dc.w    $0ff0,$0cf0,$09f0,$06f0,$03f0,$00f0,$00f3,$00f6
            dc.w    $00f9,$00fc,$00ff,$00cf,$009f,$006f,$003f,$000f
            dc.w    $030f,$060f,$090f,$0c0f,$0f0f,$0f0c,$0f09
            dc.w    $0f06,$0f03,$0f00,$0f30,$0f60,$0f90,$0fc0
```

4.1.5: DMA-Kontroll-Logik

Agnus und FatAgnus kontrollieren die 25 DMA-Kanäle, die für eine Vielzahl von Funktionen des Systems von »lebenswichtiger« Bedeutung sind. Die beiden Chips enthalten zwei Register, über die die DMA-Hardware softwaremäßig gesteuert werden kann. Diese Register heißen DMACONR und DMACON. DMACONR ist lesbar und enthält die DMA-Zustände. Dabei bedeuten gesetzte Bits, daß der zugehörige DMA-Kanal aktiv ist. Ist ein Bit nicht gesetzt, ist der zugehörige DMA-Kanal nicht aktiv. Will man den Zustand eines oder mehrerer DMA-Kanäle verändern, so kann dies über das Register DMACON geschehen.

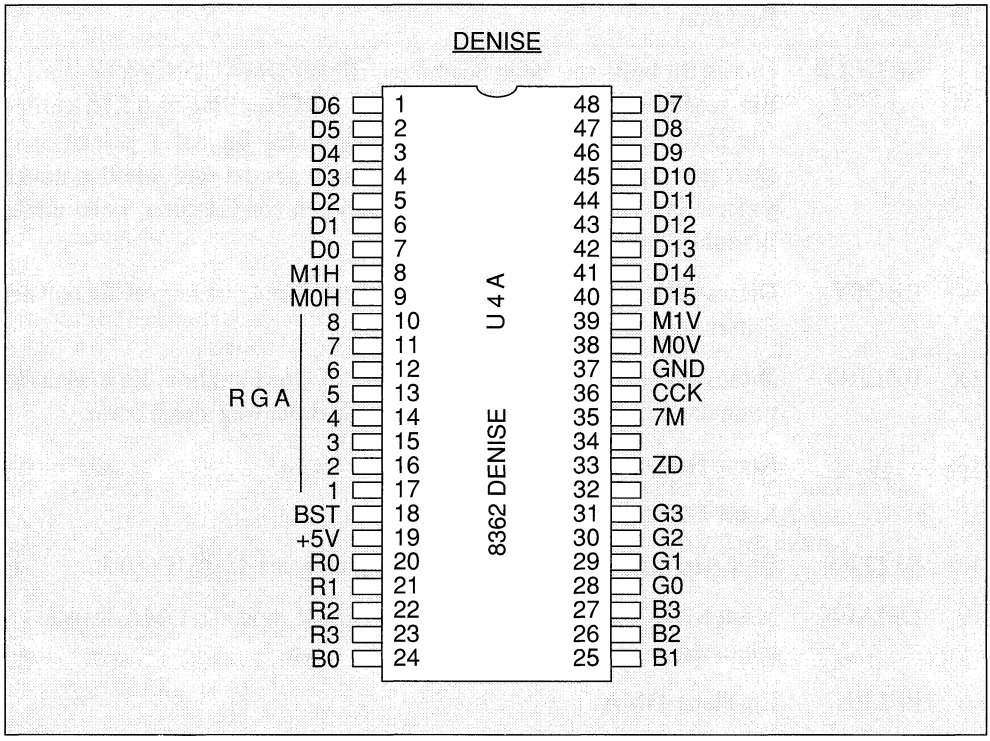
Die Beschreibung zu DMACONR/DMACON:

Bit	Name	Funktion
15	SET/CLR	Dieses Bit bestimmt beim Schreibzugriff auf DMACON, ob bestimmte Bits gesetzt oder gelöscht werden sollen. Ist dieses Bit gleich 1, so werden alle DMA-Kanäle, deren korrespondierendes Bit auf 1 gesetzt sind, aktiv gesetzt. Die Kanäle, deren Bits auf 0 gesetzt sind, bleiben unverändert. Das Gleiche gilt für das Inaktivieren von Kanälen, wenn dieses Bit auf 0 gesetzt wird.
14	BBUSY	Dieses Bit hat nur im Lesezugriff eine Funktion. Dort zeigt es an, daß der Blitter noch arbeitet.
13	BZERO	Dieses Bit hat ebenfalls nur im Lesezugriff eine Funktion. Es ist gleich 1, wenn das Ergebnis einer Blitteroperation durchweg gleich 0 war.
12		Keine Funktion.
11		Keine Funktion.
10	BLTPRI	Ist dieses Bit gesetzt, hat der Blitter Vorrang vor dem MC68000.
9	DMAEN	Master-DMA-Enable. Ist dieses Bit gelöscht, sind alle DMA-Kanäle geschlossen.
8	BPLEN	Bit-Plane-DMA.
7	COPEN	Copper-DMA.
6	BLTEN	Blitter-DMA.
5	SPREN	Sprite-DMA.
4	DSKEN	Disk-DMA.
3–0	AUDxEN	Audio-DMA für Kanal x (x = 3,2,1,0).

Kapitel 4.2: Denise

Denise ist für die gesamte Videodarstellung verantwortlich, einschließlich der Sprite-Kontrolle und der Playfield-Darstellung. Foto 11 im Farbteil zeigt Denise.

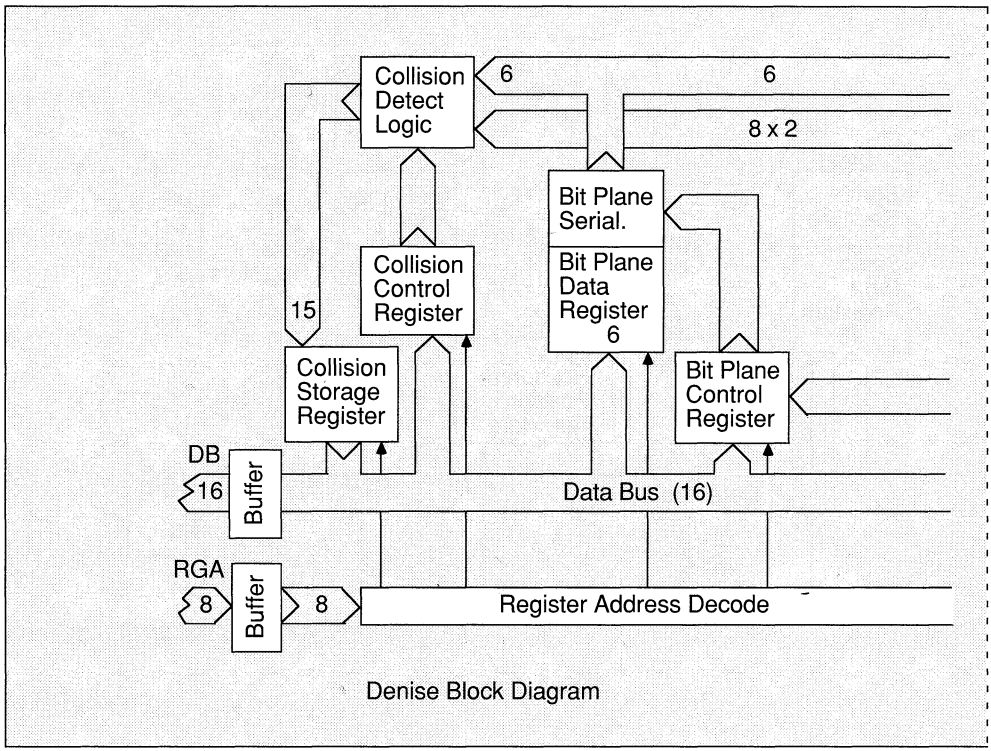
Hier die Pinbelegung und das Blockschaltbild von Denise:



Z 4.2-1: *Die Pinbelegung von Denise*

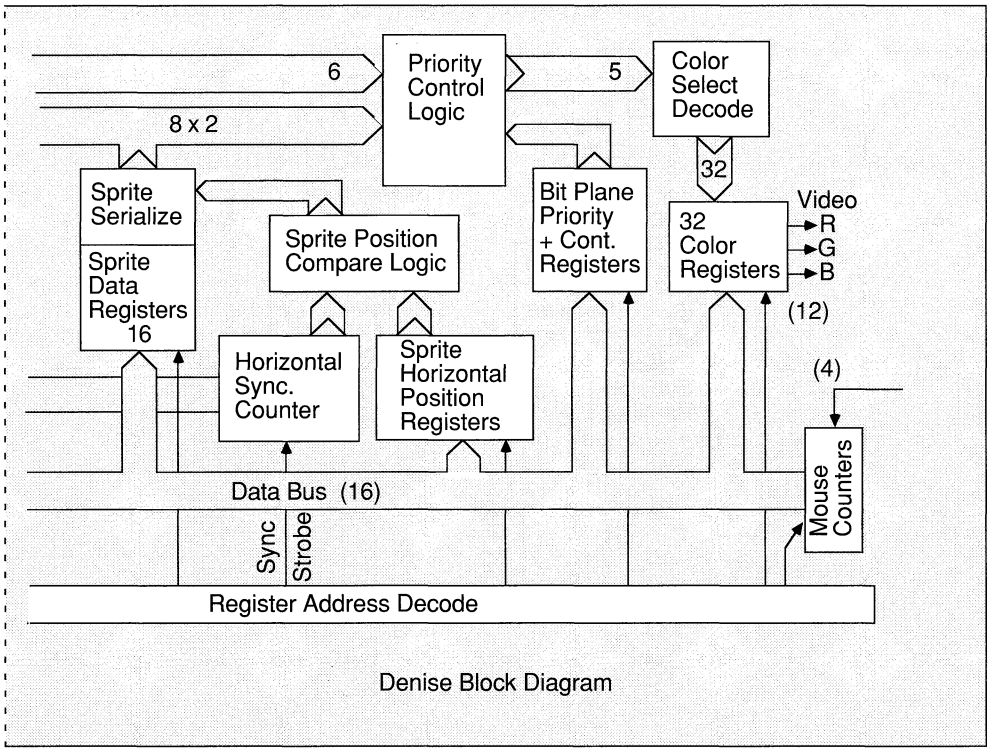
4.2.1: Die Pinbeschreibung zu Denise

Name	PIN	I/O	Beschreibung
D0–D6	7–1	I/O	Datenbusleitungen 0 bis 6.
M1H	8	I	Maus-Eingang von Port 1 horizontal.
M0H	9	I	Maus-Eingang von Port 0 horizontal.
RGA1–8	17–10	I	Diese Leitungen werden benutzt, um die internen Register zu adressieren.
BST	18	O	Color-Burst-Indikator.
VCC	19	I	+5Volt Versorgungsspannung.
R0–3	20–23	O	Rotes Videosignal digital Bit 0 bis 3.
B0–3	24–27	O	Blaues Videosignal digital Bit 0 bis 3.



Z 4.2-2: *Das Blockschaltbild von Denise (Teil 1)*

Name	PIN	I/O	Beschreibung
G0–3	28–31	O	GrünesVideosignal digital Bit 0 bis 3.
N/C	32		Nicht belegt.
ZD	33	O	Indikator für Hintergrundfarbe. Wird Hintergrundfarbe gezeigt, ist dieses Signal low.
N/C	34		Nicht belegt.
CLK	35	I	Systemtakt von 7 MHz für Denise.
CCK	36	I	Dies ist derTakt, der als Farbträgersignal dient.
VSS	37	I	Masseanschluß.
M0V	38	I	Mauseingang von Port 0 vertikal.
M1V	39	I	Mauseingang von Port 1 vertikal.
D7–D15	48–40	I/O	Datenleitungen 7 bis 15.



Z 4.2-2: *Das Blockschaltbild von Denise (Teil 2)*

4.2.2: Die Sprite-Hardware

Denise unterstützt 8 *Hardware-Sprites*, die als unabhängige Grafikteile dargestellt werden können. Die Sprites werden allerdings immer, also unabhängig von der Screen-Auflösung in einer LoRes-Auflösung dargestellt. Sie sind 16 LoRes-Punkte breit, beliebig hoch und können mit drei Farben dargestellt werden, als auch teilweise durchsichtig sein. Wenn ein Sprite komplett dargestellt wurde, kann es nochmals zur Darstellung eines weiteren Images verwendet werden.

Um ein Sprite darstellen zu können, muß zuerst eine Datenliste im *ChipMem* erstellt werden. Das Spriteimage selbst setzt sich aus zwei Planes zusammen, wodurch der Zugriff auf die drei verschiedenen Farben möglich ist. Ist ein Bit in der ersten Plane gleich 0 und das korrespondierende Bit in der zweiten Plane ebenfalls, so erscheint dieser Punkt transparent. Die Sprites sind immer in Zweiergruppen zusammengefaßt, so daß immer zwei Sprites auf die gleichen Farben in der *Farbtabelle* zugreifen. Hier eine Aufstellung der Kombinationen:

Sprites	Plane1/2	Farbregister
0 1	00	Transparent
0 1	01	17
0 1	10	18
0 1	11	19
2 3	00	Transparent
2 3	01	21
2 3	10	22
2 3	11	23
4 5	00	Transparent
4 5	01	25
4 5	10	26
4 5	11	27
6 7	00	Transparent
6 7	01	29
6 7	10	30
6 7	11	31

Die Datenliste hat folgende Form:

Word	Funktion

1 Legt die vertikale und horizontale Startposition des Sprites fest, wobei die Position nicht r lativ zum Screen festgelegt ist, sondern relativ zur physikalischen Screenposition. Das erst Byte legt die vertikale und das zweite Byte die horizontale Position fest.

2 Legt die vertikale Endposition des Sprites fest. Diese Position errechnet sich aus der vert kalen Startposition plus der Höhe des Sprites.

3 Spritedaten für die erste Zeile und erste Plane.

4 Spritedaten für die erste Zeile und zweite Plane.

5 Spritedaten für die zweite Zeile und erste Plane.

usw.

n 2 0-Words am Ende der Datenliste

Um ein Sprite, zu dem eine solche Datenliste erstellt wurde, darzustellen, müssen die Reg ster SPRxPTH/SPRxPTL des Sprites, das verwendet werden soll, auf den Anfang der Date liste gesetzt werden. Bevor dann die Sprite-DMA eingeschaltet wird, müssen aber die Sprit Datenzeiger für die nicht verwendeten Sprites auf eine leere Datenliste gesetzt werden, dam sie nicht auch dargestellt werden. Da die Zeiger aber in jedem Darstellungsdurchlauf durc die Hardware verändert werden, müssen sie laufend neu gesetzt werden. Dies geschieht i Normalfall durch die *Copperliste.*

```
/*****************************

Sprite-Demonstration
last update 16/02/88
von Frank Kremser und Jörg Koch
© Markt & Technik 1988

*****************************

Diese Demonstration bewegt ein Sprite 'natürlich' über den Bildschirm

*****************************/

#include <exec/types.h>          /* Include-Files laden */
#include <exec/tasks.h>
#include <exec/libraries.h>
#include <exec/memory.h>
#include <exec/devices.h>
#include <devices/keymap.h>
#include <graphics/copper.h>
#include <graphics/display.h>
#include <graphics/gfxbase.h>
#include <graphics/text.h>
#include <graphics/view.h>
#include <graphics/gels.h>
#include <graphics/regions.h>
#include <graphics/sprite.h>
#include <hardware/blit.h>
#include <intuition/intuition.h>
#include <intuition/intuitionbase.h>

struct  GfxBase          *GfxBase;     /* Lib Zeiger */
struct  IntuitionBase    *IntuitionBase;

struct  Screen *screen;                /* Screen-Structure-Zeiger */

USHORT Data1[] =                       /* Sprite-Image */
 {
  0, 0,
  0x0FC0, 0x0FC0,
  0x3FF0, 0x3030,
  0x7FF8, 0x4008,
  0x7FF8, 0x4008,
  0xF33C, 0x8CC4,
  0xFFFC, 0x8004,
  0xFFFC, 0x8004,
  0xFCFC, 0x8304,
  0xFFFC, 0x8004,
  0xFFFC, 0x9024,
  0x7FF8, 0x4848,
```

```
 51   0x7FF8, 0x4788,
 52   0x3FF0, 0x3030,
 53   0x0FC0, 0x0FC0,
 54
 55   0,0
 56  };
 57
 58 struct SimpleSprite sprite1 =          /* Sprite-Structure */
 59  {
 60   &Data1[0],
 61   14,                                  /* Hoehe */
 62   100,                                 /* X - Position */
 63   0,                                   /* Y - Position */
 64   2                                    /* Sprite Nummer */
 65  };
 66
 67 struct NewScreen ns =                  /* New-Screen-Structure */
 68  {
 69   0,                                   /* Linke Ecke */
 70   0,                                   /* Obere Ecke */
 71   320,                                 /* Breite */
 72   256,                                 /* Hoehe */
 73   2,                                   /* Tiefe */
 74   0,                                   /* DetailPen */
 75   1,                                   /* BlockPen */
 76   SPRITES,                             /* ViewModes */
 77   CUSTOMSCREEN,                        /* Type */
 78   NULL,
 79   NULL,
 80   NULL,
 81   NULL
 82  };
 83
 84
 85 main() /* HAUPTPROGRAMM */
 86 {
 87  int warte,step,x;
 88  USHORT schleife,schleife2,v;
 89
 90  /* Libraries öffnen */
 91  if ((IntuitionBase = (struct IntuitionBase *)
 92   OpenLibrary("intuition.library", 0)) == 0) exit();
 93
 94  if ((GfxBase = (struct GfxBase *)
 95   OpenLibrary("graphics.library", 0)) == 0) exit();
 96                          /* Screen öffnen */
 97  if ((screen = (struct Screen*) OpenScreen(&ns)) == NULL) exit();
 98
 99  SetRGB4(&screen->ViewPort,20,9,9,9);          /* Farben setzen */
100  SetRGB4(&screen->ViewPort,21,11,11,11);
```

```
101 SetRGB4(&screen->ViewPort,22,13,13,13);
102 SetRGB4(&screen->ViewPort,23,15,15,15);
103
104 schleife = GetSprite(&spritel,3);                    /* Sprite 'holen' */
105
106 v = 0; /* Variablen auf Start setzen */
107 x = 80;
108 step = 10; /* waagerechte Schrittweite */
109 /* Sprite über den Screen bewegen */
110 for(schleife2 = 0; schleife2 < 10; schleife2++)
111   {
112    for (schleife = 1; schleife < 22; schleife++)
113     {
114      x += step;
115      if(x > 300)
116       {
117        x = 290;
118        step = -10;
119       }
120      if(x < 0)
121       {
122        x = 10;
123        step = 10;
124       }
125      v += schleife; /* senkrechte Schrittweite wird ständig größer */
126      MoveSprite(&screen->ViewPort, &spritel, x, v);
127      for (warte = 0; warte < 5000; warte++);
128     }
129    for (schleife = 21; schleife > 0; schleife--)
130     {
131      x += step;
132      if(x > 300)
133       {
134        x = 290;
135        step = -10;
136       }
137      if(x < 0)
138       {
139        x = 10;
140        step = 10;
141       }
142      v -= schleife; /* senkrechte Schrittweite wird ständig kleiner */
143      MoveSprite(&screen->ViewPort, &spritel, x, v);
144      for (warte = 0; warte < 5000; warte++);
145     }
146    }
147
148   FreeSprite(3);                          /* Sprite löschen */
149
150   CloseScreen(screen);                    /* Screen und Libs */
```

```
151   CloseLibrary(GfxBase);              /* schließen */
152   CloseLibrary(IntuitionBase);
153  }
```

```
 1 ;*******************************
 2 ;
 3 ; 1.Sprite-Demonstration
 4 ; last update 10/03/88
 5 ;von Frank Kremser und Jörg Koch
 6 ; © Markt & Technik 1988
 7 ;
 8 ;*******************************
 9 ;
10 ;Diese Demonstration bewegt ein Sprite diagonal über den Screen.
11 ;Dazu werden die Befehle aus der Gfx-Library verwendet.
12 ;
13 ;*******************************
14
15      OpenLibrary  =-30-378
16      FreeSprite   =-30-384
17      GetSprite    =-30-378
18      ChangeSprite=-30-390
19      ExecBase     =4
20
21          move.l  ExecBase,a6       ;Graphics Library öffnen.
22          lea.l   GfxName,a1
23          jsr     OpenLibrary(a6)
24          move.l  d0,GfxBase
25
26          move.l  GfxBase,a6
27          move.l  #1,d0             ;Sprite Nr. 1 verfügbar machen
28          lea.l   SimpleSprite,a0   ;Setzt auch SpriteNr in der
29          jsr     GetSprite(a6)     ;SimpleSprite-Structure
30
31          move.w  #0,d5             ;Variable für x-pos initialisieren
32          move.w  #0,d6             ;Variable für y-pos initialisieren
33 loop:    move.l  GfxBase,a6
34          move.l  0,a0              ;aktueller ViewPort (keiner)
35          lea.l   SimpleSprite,a1   ;Zeiger auf SimpleSprite-Structure
36          lea.l   Image,a2          ;Zeiger auf Spritedaten
37          move.l  #5000,d0          ;Warteschleife wait:
38          sub.l   #1,d0
39          bne     wait
40          add.w   #5,d5             ;X-Position erhöhen
41          add.w   #4,d6             ;Y-Position erhöhen
42          move.w  d5,SimpleSprite+6 ;Neue X-Pos setzen
43          move.w  d6,SimpleSprite+8 ;Neue Y-Pos setzen
44          jsr     ChangeSprite(a6)  ;Sprite neu setzen
45          cmp.w   #300,d5           ;Wenn X-Position erreicht
```

```
46          bne     loop
47
48 wait2:   btst    #6,$bfe001      ;Warten, bis Maustaste gedrückt
49          bne     wait2
50
51          move.l  GfxBase,a6
52          move.l  #1,d0           ;Sprite Nr. 1 wird als
53          jsr     FreeSprite(a6)  ;frei gekennzeichnet
54          rts                     ;Rückkehr
55
56 even
57 GfxName: dc.b    'graphics.library',0
58 even
59 GfxBase: blk.l   1,0
60 even
61 SimpleSprite:                    ;SimpleSprite-Structure
62          dc.w    0,0             ;Zeiger auf Image (wird später gesetzt)
63          dc.w    16,0,0          ;Höhe, X- und Y-Position (wird später gesetzt)
64          dc.w    0               ;Sprite Nummer (wird später gesetzt)
65 even
66 Image:    dc.l    0               ;Spritedaten
67          dc.l    $07e00000,$181807e0,$2004ff8,$5e7a2184
68          dc.l    $7ffe0c30,$bffd5042,$bffd5042,$bffd4002
69          dc.l    $9ff96186,$80017ffe,$80017ffe,$46623ffc
70          dc.l    $47e23ffc,$23c41ff8,$181807e0,$07e00000
71          dc.l    0
```

```
 1 ;*******************************
 2 ;
 3 ; 2.Sprite-Demonstration
 4 ; last update 10/03/88
 5 ; von Frank Kremser und Jörg Koch
 6 ; © Markt & Technik 1988
 7 ;
 8 ;*******************************
 9 ;
10 ;Diese Demonstration zeigt zwei Sprites, die direkt über die
11 ;Hardwareregister angesprochen werden
12 ;Läuft nur über Seka einwandfrei!
13 ;
14 ;*******************************
15
16     ExecBase     =4
17     Level3Int    =$6c
18     SprBuffer    =$45000
19     INTREQR      =$dff01e          ; Interrupt request read
20     DMACON       =$dff096
21     INTENA       =$dff09a
22     Permit       =-138             Forbid=-132
```

```
23      OpenLibrary  =-408
24      CloseLibrary =-414
25
26            move.l  sp,initialSP
27            move.l  ExecBase,a6
28            lea     GfxName,a1          ;GfxLibrary öffnen
29            jsr     OpenLibrary(a6)
30            move.l  d0,GfxBase
31            move.l  d0,d6
32
33            lea     SprBuffer,a1        ;Sprites in den Spritepuffer kopieren
34            lea     Sprites,a0
35            move.l  #144,d0
36  loopl:    move.b  (a0)+,(a1)+
37            sub     #1,d0
38            bne     loopl
39
40            move.l  ExecBase,a6
41            jsr     Forbid(a6)          ;Multitasking abschalten
42
43            move.l  GfxBase,a0
44            add.l   #$32,a0             ;Zeiger auf LOFlist
45
46            move.w  #$4000,INTENA       ;Master Interrupt Enable OFF
47            move.l  Level3Int,OldVector ;Alten Interrupt sichern
48            move.l  #Handler,Level3Int  ;Neuen Interrupt setzen
49
50            move.w  #$c000,INTENA
51
52            move.w  #$0080,DMACON       ;Copper DMA stoppen
53            move.l  (a0),OldCopper      ;Zeiger auf alte Copperliste
54            move.l  #CopList,(a0)       ;Neue Liste setzen
55            move.w  #$80A0,DMACON       ;und Copper starten
56
57  wait:     btst    #6,$bfe001          ;warten, bis linke Maustaste
58            bne     wait                ;gedrückt
59
60            move.l  GfxBase,a0
61            add.l   #$32,a0             ;Zeiger auf LOFlist
62
63            move.w  #$0080,DMACON       ;Copper und Sound abschalten
64            move.l  OldCopper,(a0)      ;Alte Copperliste setzen
65            move.w  #$8180,DMACON       ;Copper einschalten
66
67            move.w  #$4000,INTENA       ;Interrupts abschalten
68            move.l  OldVector,Level3Int ;Alten Interrupt setzen
69            move.w  #$c000,INTENA       ;Interrupts einschalten
70
71            move.l  ExecBase,a6
72            jsr     Permit(a6)          ;Multitasking einschalten
```

```
 73          move.l  GfxBase,a1          ;GfxLibrary schließen
 74          jsr     CloseLibrary(a6)
 75          move.l  initialSP,sp
 76          clr.l   d0
 77          rts                         ;Rückkehr
 78
 79 Handler: movem.l d0-d2/a0-a1,-(a7)  ;Register retten
 80          move    SR,-(sp)            ;Statusregister retten
 81          move.w  INTREQR,d0          ;Interruptrequestregister lesen
 82          btst    #5,d0               ;und prüfen, ob VBlank-Interrupt
 83          bne.s   VBlank              ;wenn ja, dann weiter
 84          bra     EndHandler          ;sonst Interrupt beenden
 85
 86 VBlank:  lea     x_position,a1
 87          lea     y_position,a2
 88          lea     index,a3
 89          move.b  (a3),d2             ;Pos.-Index laden
 90          move.b  0(a1,d2.b),d0       ;neue Position laden
 91          move.b  0(a2,d2.b),d1
 92          add.b   #1,(a3)             ;Index erhöhen
 93          cmp.b   #42,(a3)            ;Wenn alle Positionen geladen,
 94          bne     cont
 95          move.l  #0,(a3)             ;dann neu beginnen
 96 cont:    move.b  d0,SprBuffer        ;neue x-position
 97          move.b  d1,SprBuffer+1      ;Neue y-position
 98          move.b  xpos,SprBuffer+72   ;Letzte x-pos. für 2. Sprite
 99          move.b  ypos,SprBuffer+73   ;Letzte y-pos.
100          move.b  d0,xpos             ;Neue Positionen sichern
101          add.b   #16,d1              ;Y-Pos + Höhe des Sprites
102          move.b  d1,SprBuffer+2      ;eintragen
103          move.b  ypos,d1             ;Das gleiche mit 2. Sprite
104          add.    #16,d1
105          move.b  d1,SprBuffer+74
106          move.b  SprBuffer+1,ypos
107
108 EndHandler:
109          move    (sp)+,SR            ;Statusregister zurückspeichern
110          movem.l (sp)+,d0-d2/a0-a1   ;Register zurück
111
112          dc.w    $4ef9               ;Interrupt beenden
113 OldVector:
114          dc.l    $0
115
116 GfxName: dc.b    'graphics.library',0
117
118 even
119 Sprites: dc.l    $a0a0b000;x-Pos, y-Pos, y-Pos + Höhe
120          dc.l    $07e00000,$181807e0,$20041ff8,$5e7a2184;SpriteDaten
121          dc.l    $7ffe0c30,$bffd5042,$bffd5042,$bffd4002
122          dc.l    $9ff96186,$80017ffe,$80017ffe,$46623ffc
```

```
123             dc.l    $47e23ffc,$23c41ff8,$181807e0,$07e00000
124             dc.l    0
125
126 Image:      dc.l    $b0b0c000;x-Pos,y-Pos,y-Pos+Höhe
127             dc.l    $07e00000,$181807e0,$2004lff8,$5e7a2184;SpriteDaten
128             dc.l    $7ffe0c30,$bffd5042,$bffd5042,$bffd4002
129             dc.l    $9ff96186,$80017ffe,$80017ffe,$46623ffc
130             dc.l    $47e23ffc,$23c41ff8,$181807e0,$07e00000
131             dc.l    0
132
133 CopList:    dc.l    $008e2c81           ;DIWSTRT
134             dc.l    $0090f4c1           ;DIWSTOP
135             dc.l    $00920038           ;DDFSTRT
136             dc.l    $009400d0           ;DDFSTOP
137             dc.l    $00e00005
138             dc.l    $00e20000
139             dc.l    $01080000           ;Modulo odd Planes
140             dc.l    $01001200           ;2 BitPlanes
141             dc.l    $01020000
142             dc.l    $01040024
142
143             dc.l    $01800000           ;Color0
144             dc.l    $01820000           ;Colorl
145             dc.l    $01a2000f           ;Colorl7
146             dc.l    $01a400af           ;Colorl8
147             dc.l    $01a600dd           ;Colorl9
148
149             dc.l    $01200004           ;Zeiger auf erstes Sprite
150             dc.l    $01225000
151
152             dc.l    $01240004           ;Zeiger auf zweites Sprite
153             dc.l    $01265048
154
155             dc.l    $fffffffe           ;Ende der Copperliste
156 x_position:
157             dc.b    90,95,100,105,110,115,120,125,130,135,140
158             dc.b    145,150,155,160,165,170,175,180,185,190
159             dc.b    185,180,175,170,165,160,155,150,145,140
160             dc.b    135,130,125,120,115,110,105,100,95,90
161
162 y_position:
163             dc.b    90,95,100,105,110,115,120,125,130,135,140
164             dc.b    145,150,155,160,165,170,175,180,185,190
165             dc.b    185,180,175,170,165,160,155,150,145,140
166             dc.b    135,130,125,120,115,110,105,100,95,90
167
168 GfxBase:    blk.l   1,0
169 initialSP:
170             blk.l   1,0
171 OldCopper:
```

```
172             blk.l   1,0
173 xpos:       dc.b    0
174 ypos:       dc.b    0
175 index:      dc.b    0


  1 ;*******************************
  2 ;
  3 ; 3.Sprite-Demonstration
  4 ; last update 10/03/88
  5 ; von Frank Kremser und Jörg Koch
  6 ; © Markt & Technik 1988
  7 ;
  8 ;*******************************
  9 ;
 10 ;Diese Demonstration zeigt nun die dritte Möglichkeit Sprites
 11 ;darzustellen. Ständig wird ein neuer Mauszeiger gesetzt, so daß
 12 ;der Eindruck entsteht, der Mauszeiger rotiert.
 13 ;
 14 ;*******************************
 15
 16       OpenLibrary  = -30-378
 17       CloseLibrary = -$19e
 18       ViewPortAdr  = -30-270
 19       SetPointer   = -30-240
 20       SetRGB4      = -30-258
 21       WaitTOF      = -30-240
 22       Delay        = -30-168
 23       ExecBase     =  4
 24
 25          jsr       Init
 26          jsr       SetSprCol
 27          jsr       Rotate
 28          rts
 29
 30 Init:     move.l  ExecBase,a6
 31           lea.l   GfxName,a1          ;GfxLibrary öffnen
 32           jsr     OpenLibrary(a6)
 33           move.l  d0,GfxBase
 34
 35           move.l  ExecBase,a6
 36           lea.l   IntName,a1          ;IntuitionLibrary öffnen
 37           jsr     OpenLibrary(a6)
 38           move.l  d0,IntBase
 39
 40           move.l  ExecBase,a6
 41           lea.l   DosName,a1          ;DosLibrary öffnen
 42           jsr     OpenLibrary(a6)
 43           move.l  d0,DosBase
 44
```

```
45            move.l  IntBase,a0          ;Aktuelles Window ermitteln
46            move.l  52(a0),AktWin
47
48            move.l  IntBase,a6
49            move.l  AktWin,a0
50            jsr     ViewPortAdr(a6)     ;ViewPort ermitteln
51            move.l  d0,AktVP
52            rts
53
54  SetSprCol:
55            move.l  AktVp,a0            ;ViewPort
56            move.l  GfxBase,a6
57            move.l  #17,d0              ;Color17
58            move.l  #0,d1               ;Rot
59            move.l  #0,d2               ;Grün
60            move.l  #0,d3               ;Blau
61            jsr     SetRGB4(a6)         ;Farbe setzen
62            move.l  AktVP,a0
63            move.l  #18,d0
64            move.l  #13,d1
65            move.l  #2,d2
66            move.l  #2,d3
67            jsr     SetRGB4(a6)
68            move.l  AktVP,a0
69            move.l  #19,d0
70            move.l  #15,d1
71            move.l  #12,d2
72            move.l  #12,d3
73            jsr     SetRGB4(a6)
74            rts
75
76  Rotate:   move.l  IntBase,a6
77            move.l  AktWin,a0           ;Aktuelles Window
78            lea.l   Image0,a1           ;Image setzen
79            move.l  #14,d0              ;X-Position setzen
80            move.l  #16,d1              ;Y-Position
81            move.l  #-7,d2              ;HotSpot x
82            move.l  #-7,d3              ;HotSpot y
83            jsr     SetPointer(a6)      ;Mauszeiger setzen
84            jsr     Warte               ;Warten
85            btst    #6,$bfe00           ;Ist Maustaste gedrückt?
86            bne     cont1               ;Wenn nein, dann weiter
87            bra     ende                ;Ansonsten beenden
88  cont1:    move.l  IntBase,a6
89            move.l  AktWin,a0
90            lea.l   Image1,a1
91            move.l  #14,d0
92            move.l  #16,d1
93            move.l  #-6,d2
94            move.l  #-8,d3
```

```
 95          jsr     SetPointer(a6)
 96          jsr     Warte
 97          btst    #6,$bfe001
 98          bne     cont2
 99          bra     ende
100 cont2:   move.l  IntBase,a6
101          move.l  AktWin,a0
102          lea.l   Image2,a1
103          move.l  #14,d0
104          move.l  #16,d1
105          move.l  #-5,d2
106          move.l  #-7,d3
107          jsr     SetPointer(a6)
108          jsr     Warte
109          btst    #6,$bfe001
110          bne     cont3
111          bra     ende
112 cont3:   move.l  IntBase,a6
113          move.l  AktWin,a0
114          lea.l   Image3,a1
115          move.l  #14,d0
116          move.l  #16,d1
117          move.l  #-4,d2
118          move.l  #-6,d3
119          jsr     SetPointer(a6)
120          jsr     Warte
121          btst    #6,$bfe001
122          bne     cont4
123          bra     ende
124 cont4:   move.l  IntBase,a6
125          move.l  AktWin,a0
126          lea.l   Image4,a1
127          move.l  #14,d0
128          move.l  #16,d1
129          move.l  #-5,d2
130          move.l  #-5,d3
131          jsr     SetPointer(a6)
132          jsr     Warte
133          btst    #6,$bfe001
134          bne     cont5
135          bra     ende
136 cont5:   move.l  IntBase,a6
137          move.l  AktWin,a0
138          lea.l   Image5,a1
139          move.l  #14,d0
140          move.l  #16,d1
141          move.l  #-,d2
142          move.l  #-4,d3
143          jsr     SetPointer(a6)
144          jsr     Warte
```

```
145             btst    #6,$bfe001
146             bne     cont6
147             bra     ende
148 cont6:      move.l  IntBase,a6
149             move.l  AktWin,a0
150             lea.l   Image6,a1
151             move.l  #14,d0
152             move.l  #16,d1
153             move.l  #-7,d2
154             move.l  #-5,d3
155             jsr     SetPointer(a6)
156             jsr     Warte
157             btst    #6,$bfe001
158             bne     cont7
159             bra     ende
160 cont7:      move.l  IntBase,a6
161             move.l  AktWin,a0
162             lea.l   Image7,a1
163             move.l  #14,d0
164             move.l  #16,d1
165             move.l  #-8,d2
166             move.l  #-6,d3
167             jsr     SetPointer(a6)
168             jsr     Warte
169             btst    #6,$bfe001
170             bne     Rotate
171             bra     ende
172
173 Warte:      move.l  DosBase,a66
174             move.l  #2,d1               ;2 Sekunden warten
175             jsr     Delay(a6)
176             move.l  GfxBase,a6          ;Bis zum nächsten Bildaufbau warten
177             jsr     WaitTOF(a6)
178             move.l  GfxBase,a6          ;Noch einmal
179             jsr     WaitTOF(a6)
180             rts
181
182 ende:       move.l  ExecBase,a6
183             move.l  GfxBase,a1          ;GfxLibrary schließen
184             jsr     CloseLibrary(a6)
185
186             move.l  IntBase,a1          ;IntuitionLibrary schließen
187             jsr     CloseLibrary(a6)
188
189             move.l  DosBase,a1          ;DosLibrary schließen
190             jsr     CloseLibrary(a6)
191             rts
192
193 even
194 GfxName:    dc.b    graphics.library,0
```

```
195 IntName:  dc.b    'intuition.library',0
196 DosName:  dc.b    'dos.library',0
197 even
198 GfxBase:  blk.l   1,0
199 IntBase:  blk.l   1,0
200 DosBase:  blk.l   1,0
201 AktWin:   blk.l   1,0
202 AktVP:    blk.l   1,0
203
204 even
205 Image0:   dc.l    0
206           dc.w    %0000000000000000,%0000000000000000
207           dc.w    %0111111000000000,%0000000000000000
208           dc.w    %0111111100000000,%0011111000000000
209           dc.w    %0100001100000000,%0011111000000000
210           dc.w    %0100011000000000,%0011111000000000
211           dc.w    %0100001100000000,%0011111000000000
212           dc.w    %0100100110000000,%0011011100000000
213           dc.w    %0011010011000000,%0000001110000000
214           dc.w    %0000001001100000,%0000000111000000
215           dc.w    %0000000100110000,%0000000011100000
216           dc.w    %0000000010100000,%0000000001000000
217           dc.w    %0000000001000000,%0000000000000000
218           dc.w    %0000000000000000,%0000000000000000
219           dc.w    %0000000000000000,%0000000000000000
220           dc.l    0
221
222 even
223 Image1:   dc.l    0
224           dc.w    %0000001100000000,%0000000000000000
225           dc.w    %0000010110000000,%0000001100000000
226           dc.w    %0000100011000000,%0000011110000000
227           dc.w    %0001000001100000,%0000111111000000
228           dc.w    %0001000111100000,%0000111111000000
229           dc.w    %0000110111000000,%0000001100000000
230           dc.w    %0000010110000000,%0000001100000000
231           dc.w    %0000010110000000,%0000001100000000
232           dc.w    %0000010110000000,%0000001100000000
233           dc.w    %0000010110000000,%0000001100000000
234           dc.w    %0000010110000000,%0000001100000000
235           dc.w    %0000010110000000,%0000001100000000
236           dc.w    %0000011110000000,%0000000000000000
237           dc.w    %0000000000000000,%0000000000000000
238           dc.l    0
239
240 even
241 Image2:   dc.l    0
242           dc.w    %0000000000000000,%0000000000000000
243           dc.w    %0000000111111000,%0000000000000000
244           dc.w    %0000001000011000,%0000000111110000
```

```
245             dc.w    %0000001000011000,%0000000111110000
246             dc.w    %0000000100011000,%0000000011110000
247             dc.w    %0000001001011000,%0000000111110000
248             dc.w    %0000010011111000,%0000001110110000
249             dc.w    %0000100110110000,%0000011100000000
250             dc.w    %0001001100000000,%0000111000000000
251             dc.w    %0010011000000000,%0001110000000000
252             dc.w    %0001110000000000,%0000100000000000
253             dc.w    %0000100000000000,%0000000000000000
254             dc.w    %0000000000000000,%0000000000000000
255             dc.w    %0000000000000000,%0000000000000000
256             dc.l    0
257
258 even
259 Image3:     dc.l    0
260             dc.w    %0000000000000000,%0000000000000000
261             dc.w    %0000000000000000,%0000000000000000
262             dc.w    %0000000000000000,%0000000000000000
263             dc.w    %0000000001100000,%0000000000000000
264             dc.w    %0000000010010000,%0000000001100000
265             dc.w    %0111111110001000,%0000000001110000
266             dc.w    %0100000000000100,%0011111111111000
267             dc.w    %0111111111001100,%0011111111111000
268             dc.w    %0111111111011000,%0000000001110000
269             dc.w    %0000000011110000,%0000000001100000
270             dc.w    %0000000001100000,%0000000000000000
271             dc.w    %0000000000000000,%0000000000000000
272             dc.w    %0000000000000000,%0000000000000000
273             dc.w    %0000000000000000,%0000000000000000
274             dc.l    0
275
276 even
277 Image4:     dc.l    0
278             dc.w    %0000000000000000,%0000000000000000
279             dc.w    %0000000000000000,%0000000000000000
280             dc.w    %0000100000000000,%0000000000000000
281             dc.w    %0001010000000000,%0000100000000000
282             dc.w    %0011001000000000,%0001110000000000
283             dc.w    %0001100100000000,%0000111000000000
284             dc.w    %0000110010110000,%0000011100000000
285             dc.w    %0000011001001000,%0000001110110000
286             dc.w    %0000001100001000,%0000000111110000
287             dc.w    %0000000110001000,%0000000011110000
288             dc.w    %0000001100001000,%0000000111110000
289             dc.w    %0000001111111000,%0000000111110000
290             dc.w    %0000000111111000,%0000000000000000
291             dc.w    %0000000000000000,%0000000000000000
292             dc.l    0
293
294 even
```

```
295 Image5:     dc.l    0
296             dc.w    %0000000000000000,%0000000000000000
297             dc.w    %0000011110000000,%0000000000000000
298             dc.w    %0000011010000000,%0000001100000000
299             dc.w    %0000011010000000,%0000001100000000
300             dc.w    %0000011010000000,%0000001100000000
301             dc.w    %0000011010000000,%0000001100000000
302             dc.w    %0000011010000000,%0000001100000000
303             dc.w    %0000011010000000,%0000001100000000
304             dc.w    %0000111011000000,%0000001100000000
305             dc.w    %0001111000100000,%0000111111000000
306             dc.w    %0001100000100000,%0000111111000000
307             dc.w    %0000110001000000,%0000011110000000
308             dc.w    %0000011010000000,%0000001100000000
309             dc.w    %0000001100000000,%0000000000000000
310             dc.l    0
311
312 even
313 Image6:     dc.l    0
314             dc.w    %0000000000000000,%0000000000000000
315             dc.w    %0000000000000000,%0000000000000000
316             dc.w    %0000000001000000,%0000000000000000
317             dc.w    %0000000011100000,%0000000001000000
318             dc.w    %0000000110010000,%0000000011100000
319             dc.w    %0000001100100000,%0000000111000000
320             dc.w    %0011011001000000,%0000001110000000
321             dc.w    %0111110010000000,%0011011100000000
322             dc.w    %0110100100000000,%0011111000000000
323             dc.w    %0110001000000000,%0011110000000000
324             dc.w    %0110000100000000,%0011111000000000
325             dc.w    %0110000100000000,%0011111000000000
326             dc.w    %0111111000000000,%0000000000000000
327             dc.w    %0000000000000000,%0000000000000000
328             dc.l    0
329
330 even
331 Image7:     dc.l    0
332             dc.w    %0000000000000000,%0000000000000000
333             dc.w    %0000000000000000,%0000000000000000
334             dc.w    %0000000000000000,%0000000000000000
335             dc.w    %0001100000000000,%0000000000000000
336             dc.w    %0011110000000000,%0001100000000000
337             dc.w    %0110111111111000,%0011110000000000
338             dc.w    %1100111111111000,%0111111111110000
339             dc.w    %1000000000001000,%0111111111110000
340             dc.w    %0100011111111000,%0011110000000000
341             dc.w    %0010010000000000,%0001100000000000
342             dc.w    %0001100000000000,%0000000000000000
343             dc.w    %0000000000000000,%0000000000000000
344             dc.w    %0000000000000000,%0000000000000000
```

```
345             dc.w    %0000000000000000,%0000000000000000
346             dc.l    0


  1 ;*******************************
  2 ;
  3 ; 4.Sprite-Demonstration
  4 ; last update 10/03/88
  5 ; von Frank Kremser und Jörg Koch
  6 ; ©Markt & Technik 1988
  7 ;
  8 ;*******************************
  9 ;
 10 ;Diese Demonstration gleicht der 3. Demonstration, läßt den Mauszeiger
 11 ;aber Kreise ziehen.
 12 ;
 13 ;*******************************
 14
 15       OpenLibrary= -30 -378
 16       CloseLibrary= -$19e
 17       ViewPortAdr= -30 -270
 18       SetPointer  = -30 -240
 19       SetRGB4     = -30 -258
 20       WaitTOF     = -30 -240
 21       Delay       = -30 -168
 22       ExecBase    = 4
 23
 24           jsr     Init
 25           jsr     SetSprCol
 26           jsr     Circle
 27           rts
 28
 29 Init:     move.l  ExecBase,a6
 30        l  ea.l    GfxName,a1          ;GfxLibrary öffnen
 31           jsr     OpenLibrary(a6)
 32           move.l  d0,GfxBase
 33
 34           move.l  ExecBase,a6
 35           lea.l   IntName,a1          ;IntuitionLibrary öffnen
 36           jsr     OpenLibrary(a6)
 37           move.l  d0,IntBase
 38
 39           move.l  ExecBase,a6
 40           lea.l   DosName,a1          ;DosLibrary öffnen
 41           jsr     OpenLibrary(a6)
 42           move.l  d0,DosBase
 43
 44           move.l  IntBase,a0          ;Aktuelles Window ermitteln
 45           move.l  52(a0),AktWin
 46
```

```
 47          move.l  IntBase,a6
48           move.l  AktWin,a0
49           jsr     ViewPortAdr(a6)    ;ViewPort ermitteln
50           move.l  d0,AktVP
51           rts
52
53  SetSprCol:
54           move.l  AktVp,a0           ;ViewPort
55           move.l  GfxBase,a6
56           move.l  #17,d0             ;Color17
57           move.l  #0,d1              ;Rot
58           move.l  #0,d2              ;Grün
59           move.l  #0,d3              ;Blau
60           jsr     SetRGB4(a6)        ;Farbe setzen
61           move.l  AktVP,a0
62           move.l  #18,d0
63           move.l  #13,d1
64           move.l  #2,d2
65           move.l  #2,d3
66           jsr     SetRGB4(a6)
67           move.l                     AktVP,a0
68           move.l  #19,d0
69           move.l  #15,d1
70           move.l  #12,d2
71           move.l  #12,d3
72           jsr     SetRGB4(a6)
73           rts
74
75  Circle:  move.   IntBase,a6
76           move.l  AktWin,a0          ;Aktuelles Window
77           lea.l   Image0,a1          ;Spriteimage
78           move.l  #14,d0             ;X-Position
79           move.l  #16,d1             ;Y-Position
80           move.l  #-14,d2            ;HotSpot x
81           move.l  #1,d3              ;HotSpot y
82           jsr     SetPointer(a6)     ;neuen Mauszeiger setzen
83           jsr     Warte              ;Warten
84           btst    #6,$bfe001         ;Ist linke Maustaste gedrückt?
85           bne     cont1              ;Wenn nein, dann weiter
86           bra     ende               ;Ansonsten beenden
87  cont1:   move.l  IntBase,a6
88           move.l  AktWin,a0
89           lea.l   Image1,a1
90           move.l  #14,d0
91           move.l  #16,d1
92           move.l  #-17,d2
93           move.l  #-6,d3
94           jsr     SetPointer(a6)
95           jsr     Warte
96           btst    #6,$bfe001
```

```
 97            bne     cont2
 98            bra     ende
 99 cont2:     move.l  IntBase,a6
100            move.l  AktWin,a0
101            lea.l   Image2,a1
102            move.l  #14,d0
103            move.l  #16,d1
104            move.l  #-14,d2
105            move.l  #-14,d3
106            jsr     SetPointer(a6)
107            jsr     Warte
108            btst    #6,$bfe001
109            bne     cont3
110            bra     ende
111 cont3:     move.l  IntBase,a6
112            move.l  AktWin,a0
113            lea.l   Image3,a1
114            move.l  #14,d0
115            move.l  #16,d1
116            move.l  #-7,d2
117            move.l  #-17,d3
118            jsr     SetPointer(a6)
119            jsr     Warte
120            btst    #6,$bfe001
121            bne     cont4
122            bra     ende
123 cont4:     move.l  IntBase,a6
124            move.l  AktWin,a0
125            lea.l   Image4,a1
126            move.l  #14,d0
127            move.l  #16,d1
128            move.l  #1,d2
129            move.l  #-14,d3
130            jsr     SetPointer(a6)
131            jsr     Warte
132            btst    #6,$bfe001
133            bne     cont5
134            bra     ende
135 cont5:     move.l  IntBase,a6
136            move.l  AktWin,a0
137            lea.l   Image5,a1
138            move.l  #14,d0
139            move.l  #16,d1
140            move.l  #4,d2
141            move.l  #-7,d3
142            jsr     SetPointer(a6)
143            jsr     Warte
144            btst    #6,$bfe001
145            bne     cont6
146            bra     ende
```

```
147 cont6:    move.l  IntBase,a6
148           move.l  AktWin,a0
149           lea.l   Image6,a1
150           move.l  #14,d0
151           move.l  #16,d1
152           move.l  #1,d2
153           move.l  #1,d3
154           jsr     SetPointer(a6)
155           jsr     Warte
156           btst    #6,$bfe001
157           bne     cont7
158           bra     ende
159 cont7:    move.l  IntBase,a6
160           move.l  AktWin,a0
161           lea.l   Image7,a1
162           move.l  #14,d0
163           move.l  #16,d1
164           move.l  #-6,d2
165           move.l  #4,d3
166           jsr     SetPointer(a6)
167           jsr     Warte
168           btst    #6,$bfe001
169           bne     Circle
170           bra     ende
171
172 Warte:    move.l  DosBase,a6
173           move.l  #2,d1               ;2 Sekunden warten
174           jsr     Delay(a6)
175           move.l  GfxBase,a6          ;Bis zum nächsten Bildaufbau warten
176           jsr     WaitTOF(a6)
177           move.l  GfxBase,a6          ;Noch einmal
178           jsr     WaitTOF(a6)
179           rts     ende:
180           move.l  ExecBase,a6
181           move.l  GfxBase,a1          ;GfxLibrary schließen
182           jsr     CloseLibrary(a6)
183           move.l  IntBase,a1          ;IntuitionLibrary schließen
184           jsr     CloseLibrary(a6)
185           move.l  DosBase,a1          ;DosLibrary schließen
186           jsr     CloseLibrary(a6)
187           rts
188
189 even
190 GfxName: dc.b    'graphics.library',0
191 IntName: dc.b    'intuition.library',0
192 DosName: dc.b    'dos.library',0
193 even
194 GfxBase: blk.l   1,0
195 IntBase: blk.l   1,0
196 DosBase: blk.l   1,0
```

```
197 AktWin:   blk.l   1,0
198 AktVP:    blk.l   1,0
199
200 even
201 Image0:   dc.l    0
202           dc.w    %0000000000000000,%0000000000000000
203           dc.w    %0111111000000000,%0000000000000000
204           dc.w    %0111111100000000,%0011111000000000
205           dc.w    %0100001100000000,%0011111000000000
206           dc.w    %0100011000000000,%0011111000000000
207           dc.w    %0100001100000000,%0011111000000000
208           dc.w    %0100100110000000,%0011011100000000
209           dc.w    %0011010011000000,%0000001110000000
210           dc.w    %0000001001100000,%0000000111000000
211           dc.w    %0000000100110000,%0000000011100000
212           dc.w    %0000000010100000,%0000000001000000
213           dc.w    %0000000001000000,%0000000000000000
214           dc.w    %0000000000000000,%0000000000000000
215           dc.w    %0000000000000000,%0000000000000000
216           dc.l0
217
218 even
219 Image1:   dc.l    0
220           dc.w    %0000001100000000,%0000000000000000
221           dc.w    %0000010110000000,%0000001100000000
222           dc.w    %0000100011000000,%0000011110000000
223           dc.w    %0001000001100000,%0000111111000000
224           dc.w    %0001000111100000,%0000111111000000
225           dc.w    %0000110111000000,%0000001100000000
226           dc.w    %0000010110000000,%0000001100000000
227           dc.w    %0000010110000000,%0000001100000000
228           dc.w    %0000010110000000,%0000001100000000
229           dc.w    %0000010110000000,%0000001100000000
230           dc.w    %0000010110000000,%0000001100000000
231           dc.w    %0000010110000000,%0000001100000000
232           dc.w    %0000011110000000,%0000000000000000
233           dc.w    %0000000000000000,%0000000000000000
234           dc.l    0
235
236 even
237 Image2:   dc.l    0
238           dc.w    %0000000000000000,%0000000000000000
239           dc.w    %0000000111111000,%0000000000000000
240           dc.w    %0000001000011000,%0000000111110000
241           dc.w    %0000001000011000,%0000000111110000
242           dc.w    %0000000100011000,%0000000011110000
243           dc.w    %0000001001011000,%0000000111110000
244           dc.w    %0000010011111000,%0000001110110000
245           dc.w    %0000100110110000,%0000011100000000
246           dc.w    %0001001100000000,%0000111000000000
```

```
247             dc.w    %0010011000000000,%0001110000000000
248             dc.w    %0001110000000000,%0000100000000000
249             dc.w    %0000100000000000,%0000000000000000
250             dc.w    %0000000000000000,%0000000000000000
251             dc.w    %0000000000000000,%0000000000000000
252             dc.l    0
253
254 even
255 Image3:     dc.l    0
256             dc.w    %0000000000000000,%0000000000000000
257             dc.w    %0000000000000000,%0000000000000000
258             dc.w    %0000000000000000,%0000000000000000
259             dc.w    %0000000001100000,%0000000000000000
260             dc.w    %0000000010010000,%0000000001100000
261             dc.w    %0111111110001000,%0000000001110000
262             dc.w    %0100000000000100,%0011111111111000
263             dc.w    %0111111111001100,%0011111111111000
264             dc.w    %0111111111011000,%0000000001110000
265             dc.w    %0000000011110000,%0000000001100000
266             dc.w    %0000000001100000,%0000000000000000
267             dc.w    %0000000000000000,%0000000000000000
268             dc.w    %0000000000000000,%0000000000000000
269             dc.w    %0000000000000000,%0000000000000000
270             dc.l    0
271
272 even
273 Image4:     dc.l    0
274             dc.w    %0000000000000000,%0000000000000000
275             dc.w    %0000000000000000,%0000000000000000
276             dc.w    %0000100000000000,%0000000000000000
277             dc.w    %0001010000000000,%0000100000000000
278             dc.w    %0011001000000000,%0001110000000000
279             dc.w    %0001100100000000,%0000111000000000
280             dc.w    %0000110010110000,%0000011100000000
281             dc.w    %0000011001001000,%0000001110110000
282             dc.w    %0000001100001000,%0000000111110000
283             dc.w    %0000000110001000,%0000000011110000
284             dc.w    %0000001100001000,%0000000111110000
285             dc.w    %0000001111111000,%0000000111110000
286             dc.w    %0000000111111000,%0000000000000000
287             dc.w    %0000000000000000,%0000000000000000
288             dc.l0
289
290 even
291 Image5:     dc.l    0
292             dc.w    %0000000000000000,%0000000000000000
293             dc.w    %0000011110000000,%0000000000000000
294             dc.w    %0000011010000000,%0000001100000000
295             dc.w    %0000011010000000,%0000001100000000
296             dc.w    %0000011010000000,%0000001100000000
```

```
297             dc.w    %0000011010000000,%0000001100000000
298             dc.w    %0000011010000000,%0000001100000000
299             dc.w    %0000011010000000,%0000001100000000
300             dc.w    %0000111011000000,%0000001100000000
301             dc.w    %0001111000100000,%0000111111000000
302             dc.w    %0001100000100000,%0000111111000000
303             dc.w    %0000110001000000,%0000011110000000
304             dc.w    %0000011010000000,%0000001100000000
305             dc.w    %0000001100000000,%0000000000000000
306             dc.l    0
307
308 even
309 Image6:     dc.l    0
310             dc.w    %0000000000000000,%0000000000000000
311             dc.w    %0000000000000000,%0000000000000000
312             dc.w    %0000000001000000,%0000000000000000
313             dc.w    %0000000011100000,%0000000001000000
314             dc.w    %0000000110010000,%0000000011100000
315             dc.w    %0000001100100000,%0000000111000000
316             dc.w    %0011011001000000,%0000001110000000
317             dc.w    %0111111001000000,%0011011100000000
318             dc.w    %0110100100000000,%0011111000000000
319             dc.w    %0110001000000000,%0011110000000000
320             dc.w    %0110000100000000,%0011111000000000
321             dc.w    %0110000100000000,%0011111000000000
322             dc.w    %0111111000000000,%0000000000000000
323             dc.w    %0000000000000000,%0000000000000000
324             dc.l    0
325
326 even
327 Image7:     dc.l    0
328             dc.w    %0000000000000000,%0000000000000000
329             dc.w    %0000000000000000,%0000000000000000
330             dc.w    %0000000000000000,%0000000000000000
331             dc.w    %0001100000000000,%0000000000000000
332             dc.w    %0011110000000000,%0001100000000000
333             dc.w    %0110111111111000,%0011100000000000
334             dc.w    %1100111111111000,%0111111111110000
335             dc.w    %1000000000001000,%0111111111110000
336             dc.w    %0100011111111000,%0011100000000000
337             dc.w    %0010010000000000,%0001100000000000
338             dc.w    %0001100000000000,%0000000000000000
339             dc.w    %0000000000000000,%0000000000000000
340             dc.w    %0000000000000000,%0000000000000000
341             dc.w    %0000000000000000,%0000000000000000
342             dc.l    0
```

Eine besondere Möglichkeit der Darstellung bietet noch der *Attached-Mode*. Es kann Sprite 1 zu 0, 3 zu 2, 5 zu 4 und 7 zu 6 geschaltet werden. Dazu muß das Bit 7 des zweiten Datenwortes der Sprites 1, 3, 5 oder 7 gesetzt werden. Der Attached-Mode bewirkt, daß aus den zwei angegebenen Sprites jeweils ein einzelnes Sprite wird, das aber nicht mehr aus drei Farben plus Transparent, sondern aus 15 Farben plus Transparent besteht. Dazu muß erstens das oben erwähnte Attached-Bit gesetzt werden, und jeweils eine Datenliste für die beiden Sprites, die zusammen dargestellt werden sollen, erstellt werden. Die Positionsdaten von beiden Sprites müssen übereinstimmen, aber bei dem zweiten Datenwort des zugewiesenen Sprites (1, 3, 5 oder 7) muß Bit 7 gesetzt werden. Die Imagedaten der Sprites 0, 2, 4 oder 6 werden als Planes 0 und 1, die der Sprites 1, 3, 5 oder 7 als Planes 2 und 3 interpretiert. Als Farben werden die Farbregister 17 bis 31 verwendet.

4.2.3: Die Playfield-Hardware

Fast die gesamte Videodarstellung des Amiga beruht auf den Playfields. Alle normalen Screens sind als Playfield aufzufassen. Um nun ein solches Playfield zu erstellen, müssen einige Vorarbeiten erledigt werden.

Als erstes muß natürlich genügend Speicherplatz für das Playfield bereitgestellt werden, das erstellt werden soll. Soll das Playfield 320 x 200 Punkte groß sein, so werden 8000 Byte für ein BitPlane benötigt. Als *BitPlanes* werden die logischen Speicherebenen eines Playfields bezeichnet, die die verwendeten Farben kennzeichnen. Ein Playfield mit 3 BitPlanes kann beispielsweise 2 hoch 3, also 8 Farben repräsentieren. Für ein solches Playfield müssen demnach 3 mal 8000 Byte, also 24000 Byte bereitgestellt werden. Maximal kann ein Playfield 6 BitPlanes (im HAM-Modus) besitzen. Im LoRes-Modus sind maximal 5 Planes möglich und im HiRes-Modus maximal 4.

Hat man diesen Speicherplatz reserviert, so müssen die Zeiger BPLxPTH/BPLxPTL für die Startadressen der einzelnen BitPlanes gesetzt werden. Als nächstes gibt man im BPLCON0-Register die ViewModi an, die verwendet werden sollen:

Registerbeschreibung BPLCON0:

Bit	Name	Funktion
15	HIRES	Setzt den HiRes-Modus.
14–12	BPU2–0	Setzt die Anzahl der verwendeten BitPlanes (010 = zwei BitPlanes).
11	HOMOD	Setzt den Hold-And-Modify-Mode.
10	DBLPF	Setzt den Dual-Playfield-Modus.
2	LACE	Setzt den Interlace-Mode.

Als nächstes müssen die Register BPL1MOD und BPL2MOD auf 0 gesetzt werden. Um die genaue Auflösung und Bildpositionierung zu bestimmen, müssen weiterhin die Register DIWSTRT, DIWSTOP, DDFSTRT und DDFSTOP gesetzt werden.

DIWSTRT legt die Display-Position fest, ab der das Playfield gezeigt werden soll. Das erste Byte von DIWSTRT legt die vertikale und das zweite Byte die horizontale Position fest. Im Normalfall ist DIWSTRT gleich $2C81. DIWSTOP hingegen legt die rechte untere Endposition des darzustellenden Playfields fest. Sowohl DIWSTRT, als auch DIWSTOP sind nicht vom Display-Modus abhängig, bleiben also auch im HiRes-Modus unverändert. Das erste Byte bei DIWSTOP enthält die vertikale Endposition. Das höchste Bit dieses Bytes wird zusätzlich invertiert und als 9. Bit erkannt, wodurch auch Werte, die größer als 256 sind, angegeben werden können. Von der physikalischen horizontalen Endposition wird $100 (=256) abgezogen und im zweiten Byte angegeben. Wenn DIWSTRT gleich $2C81 ist, ist der Normalwert für DIWSTOP gleich $F4C1.

Die Register DDFSTRT und DDFSTOP legen Datenzugriffsstart und -stop fest. Für DDFSTRT wird zuerst die horizontale Startposition, aus obigem Beispiel ist dies $81, durch zwei geteilt. Ist die Darstellung im LoRes-Modus, muß dann noch der Wert 8*5 und im HiRes-Modus der Wert 4*5 abgezogen werden. Das ergibt für DDFSTRT einen Wert von $38 im LoRes-Modus und $3C im HiRes-Modus. Der Wert für DDFSTOP errechnet sich dann wie folgt:

DDFSTOP = DDFSTRT + (8*(Screenbreite/16–1)) für LoRes

DDFSTOP = DDFSTRT + (4*(Screenbreite/16–1)) für HiRes

Mit Screenbreite ist die Breite des darzustellenden Playfields in Punkten gemeint.

Hat man alle oben erwähnten Schritte durchgeführt, so braucht man nur noch die BitPlane-DMA zu starten. Wie dies geschieht, ersehen Sie aus dem Kapitel zur DMA-Kontroll-Logik. Besonders angemerkt werden muß noch, daß es keinen Zweck hat, durch einfaches Setzen der Register ein Playfield zu erzeugen. Dies sollte durch eine Copperliste geschehen.

Das oben erwähnte Vorgehen erzeugt ein normales Playfield, das auch als Screen bekannt ist. Doch es bestehen noch eine Vielzahl von weiteren Möglichkeiten. Zum ersten ist hier der Dual-Playfield-Modus zu erwähnen. Ist dieser Modus durch Setzen von Bit 10 in BPLCON0 eingeschaltet, so werden die BitPlanes mit ungerader Nummer unabhängig von denen mit gerader Nummer behandelt. Das bedeutet, das die BitPlanes 1, 3 und 5, sowie die BitPlanes 2, 4 und 6 jeweils ein eigenes Playfield darstellen, die weitgehend unabhängig voneinander behandelt werden können. Jedes dieser beiden Playfields kann also bis zu 3 BitPlanes besitzen, also bis zu 8 Farben darstellen. Das Playfield mit den ungeraden Planes spricht die Farbregister 0 bis 7 an, während das andere Playfield die Register 16 bis 23 anspricht. Ist ein Bildpunkt im ersten Playfield

auf Register 0 gesetzt, so erscheint dieser Punkt als transparent, wodurch das zweite Playfield sichtbar wird.

Ein weiterer Punkt, der Playfields so interessant macht, ist der, daß sie größer sein können als der dargestellte Ausschnitt.

Soll ein Playfield höher sein als der dargestellte Ausschnitt, so ist dies ohne allzu großen Aufwand möglich. Dazu erstellt man lediglich genügend große BitPlanes, wobei natürlich die gewünschte Höhe berücksichtigt werden muß. Die Höhe des dargestellten Ausschnittes ist ja durch DIWSTRT und DIWSTOP festgelegt. Soll der Ausschnitt nun vertikal »bewegt« werden, so addiert, bzw. subtrahiert man einfach Anzahl der Worte pro Zeile zu den Startzeigern BPLxPTH/BPLxPTL der jeweiligen BitPlane-Zeiger.

Will man aber Playfields erstellen, die breiter sind als der dargestellte Ausschnitt, so wird es etwas schwieriger. Um dies überhaupt durchführen zu können, sind die Register BPL1MOD und BPL2MOD notwendig. Wurde eine Zeile komplett dargestellt, so wird zu dem momentanen Display-BitPlane-Zeiger der Wert aus BPLxMOD hinzuaddiert, um bei der nächsten Zeile an der neuen Speicherposition mit der Darstellung der Daten zu beginnen. In den Registern BPLxMOD muß die Anzahl der Words stehen, die übersprungen werden sollen. Wenn man also einen Ausschnitt von 320 Punkten Breite (=20 Words) hat, das Playfield aber 640 (=40 Words) breit ist, muß BPLxMOD auf den Wert 40–20 = 20 gesetzt werden. Es existieren zwei BPLxMOD-Register, damit im Dual-Playfield-Modus die zwei Playfields verschiedene Abmaße haben können. BPL1MOD bezieht sich auf alle ungeraden BitPlanes und BPL2MOD auf alle geraden.

Soll ein solches breites Playfield horizontal »bewegt« werden, braucht man nur noch die einzelnen BitPlane-Zeiger um die Anzahl der Words zu erhöhen, bzw. erniedrigen. Dies verschiebt das Playfield aber immer um 16 Punkte. Um nur um einen Punkt verschieben zu können, muß noch das Register BPLCON1 entsprechend gesetzt werden. Die Bits 0 bis 3 beeinflussen das Scrolling des ersten Playfields und die Bits 4 bis 7 das Scrolling von Playfield 2. Will man beispielsweise das erste Playfield um 16 Punkte nach links bewegen, so setzt man die Bits 0 bis 3 von BPLCON1 auf 0 und die BitPlane-Zeiger auf die Startposition des Playfields. Anschließend inkrementiert man BPLCON1 jeweils um 1, bis die Bits 0 bis 3 gleich %1111, bzw. $F sind. Dann setzt man sie wieder zurück und erhöht den BitPlane-Zeiger um ein Word. Soll ein Playfield höher und breiter sein als der gezeigte Ausschnitt, so muß man nur noch die beiden oben angeführten Methoden kombinieren.

```
/*******************************

1. Playfield-Demonstration
last update 16/02/88
von Frank Kremser und Jörg Koch
© Markt & Technik 1988

*******************************

Diese Demo zeigt ein einzelnes Playfield mit einer Größe von 960 x 600

*******************************/

#include <exec/types.h> /* Include-Files laden */
#include <exec/memory.h>
#include <graphics/gfxbase.h>
#include <graphics/copper.h>
#include <graphics/view.h>
#include <graphics/rastport.h>
#include <devices/gameport.h>
#include <devices/inputevent.h>

UWORD colors[] =
 {
  0,
  0x555,
  0xAAA,
  0xFFF
 };

struct View         v,*oldview;
struct ViewPort     vp;
struct ColorMap     *cm;
struct RasInfo      ri;
struct BitMap       *bm;
struct RastPort     rp;

struct GfxBase      *GfxBase; /* Library-Zeiger */

main() /* HAUPTPROGRAMM */
{
 int i, x, y;
                          /* Library öffnen */
 if (!(GfxBase = OpenLibrary ("graphics.library", 0)))
  exit();
            /* Speicher für BitMap-Structure bereitstellen */
 if (!(bm = AllocMem ((long) sizeof (*bm), MEMF_CHIP|MEMF_CLEAR)))
  exit();

 InitView (&v);       /* Neue View-Structure initialisieren */
```

```
51  InitVPort(&vp); /* Neue Viewport-Structure initialisieren */
52  InitBitMap(bm, (long)2, 960, 600); /* BitMap-Structure init. */
53  InitRastPort(&rp); /* Rastport-Structure initialisieren */
54
55  v.ViewPort = &vp; /* ViewPort eintragen */
56
57  ri.BitMap = bm; /* BitMap eintragen */
58  ri.RxOffset = ri.RyOffset = ri.Next = NULL;
59
60  vp.DWidth = 320; /* dargestellte Ausschnittgröße festlegen */
61  vp.DHeight = 256;
62  vp.RasInfo = &ri; /* RasInfo-Structure eintragen */
63  vp.ColorMap = GetColorMap(4); /* Color-Map eintragen und init. */
64
65  rp.BitMap = bm; /* BitMap eintragen */
66
67  for(i=0; i<2; i++) /* Speicher für Bitplanes bereitstellen */
68   if(!(bm->Planes[i] = AllocRaster(960, 600)))
69    exit();
70
71  MakeVPort(&v, &vp); /* ViewPort in View-Structure eintragen */
72  MrgCop(&v); /* View in Copperliste eintragen */
73  LoadRGB4(&vp, colors, 4); /* Farben setzen */
74  oldview = GfxBase->ActiView; /* Alten View sichern */
75  LoadView(&v); /* Neuen View darstellen */
76
77  i = 1; /*                Muster zeichnen */
78  SetDrMd(&rp, JAM1);
79  SetRast(&rp, 0);
80  SetAPen(&rp, 1);
81  for(x=0, y=0; x<960; x += 8, y += 5)
82   {
83    Move(&rp, (long)x, 0);
84    Draw(&rp, 959, (long)y);
85    Draw(&rp, 959-x, 599);
86    Draw(&rp, 0, 599-y);
87    Draw(&rp, (long)x, 0L);
88    if(!(++i&3)) i++;
89    SetAPen(&rp, (long)i);
90   }
91  SetAPen(&rp, 3);                /* Text zeichnen */
92  Move(&rp, 429, 301);
93  Text(&rp, "Ein Superplayfield", 18);
94  SetAPen(&rp, 1);
95  Move(&rp, 428, 300);
96  Text(&rp, "Ein Superplayfield", 18);
97
98  for(x=0;x<640;x+=4)       /* Playfield nach links bewegen */
99   {
100   ri.RxOffset = x;
```

```
101     WaitTOF();
102     ScrollVPort(&vp);
103    }
104   for(y=0;y<344;y+=2)        /* Playfield nach oben bewegen */
105    {
106
107     ri.RyOffset = y;
108     WaitTOF();
109     ScrollVPort(&vp);
110    }
111   for(x=640;x>0;x-=4)        /* Playfield nach rechts bewegen */
112    {
113     ri.RxOffset = x;
114     WaitTOF();
115     ScrollVPort(&vp);
116    }
117   for(y=344;y>0;y-=2)        /* Playfield nah unten bewegen */
118    {
119     ri.RyOffset = y;
120     WaitTOF();
121     ScrollVPort(&vp);
122    }
123   y=12;
124   for(x=0;x<640;x+=4)        /* Playfield diagonal bewegen */
125    {
126     y+=2;
127     ri.RxOffset = x;
128     ri.RyOffset = y;
129     WaitTOF();
130     ScrollVPort(&vp);
131    }
132
133   LoadView(oldview); /* Alten View setzen */
134   WaitTOF();              /* Warten, bis oberer Bildschirmrand erreicht */
135   FreeVPortCopLists(&vp); /* Alte Copperlisten löschen */
136   FreeCprList(v.LOFCprList);
137   FreeColorMap(vp.ColorMap); /* Colormap löschen */
138   for(i=0; i<2; i++) /* Bitplanes löschen */
139    if(bm->Planes[i])
140     FreeRaster(bm->Planes[i], 960, 600);
141   FreeMem(bm, (long)sizeof(*bm)); /* BitMap-Structure löschen */
142
143   CloseLibrary(GfxBase); /* Library schließen */
    }
```

```
/*****************************

2. Playfield-Demonstration
last update 16/02/88
von Frank Kremser und Jörg Koch
© Markt & Technik 1988

*****************************

Diese Playfield zeigt ein Gittermuster, das sich bewegt. Dabei werden
jeweils die Linien einer Richtung auf ein eigenes Playfield gezeichnet.
Die Bewegung kommt dadurch zustande, daß eines der Playfields bewegt
wird. Optisch besteht allerdings zwischen einer Bewegung nach links
und unten, bzw nach rechts und oben kein Unterschied.

*****************************/

#include <exec/types.h>        /* Include-Files laden */
#include <intuition/intuition.h>
#include <graphics/gfxbase.h>

struct View         v, *oldview;
struct ViewPort     vp;
struct ColorMap     *cm;

struct BitMap     b, b2;
struct RastPort   rp, rp2;
struct RasInfo    ri, ri2;

LONG  i;
SHORT k,n;

struct GfxBase *GfxBase; /* Library-Zeiger */

UWORD colors[] =
 {
  0x000,  0xf00,  0,  0,
  0,      0,      0,  0,
  0,      0x4f8
 };

UWORD *colorpalette;
short w;

main() /* HAUPTPROGRAMM */
{
 int x;
                              /* Library öffnen */
 GfxBase = (struct GfxBase *)OpenLibrary("graphics.library",0);
 if (GfxBase == NULL) exit();
```

```
51
52  oldview = GfxBase->ActiView; /* Alten View sichern */
53
54  InitBitMap(&b,1,400,400); /* BitMap-Structures initialisieren */
55  InitBitMap(&b2,1,320,320);
56
57  for(i=0; i<1; i++) /* Speicher für Bitplanes des 1. Playfields */
58   {                 /* bereitstellen */
59    b.Planes[i] = (PLANEPTR)AllocRaster(400,400);
60    if(b.Planes[i] == NULL) exit();
61    BltClear(b.Planes[i],RASSIZE(400,400),0);
62   }
63
64  for(i=0; i<1; i++) /* Genauso bei zweitem Playfield */
65   {
66    b2.Planes[i] = (PLANEPTR)AllocRaster(320,320);
67    if(b2.Planes[i] == NULL) exit();
68    BltClear(b2.Planes[i],RASSIZE(320,320),0);
69   }
70
71  ri.BitMap    = &b; /* BitMap-Structure eintragen */
72  ri.RxOffset  = 0;
73  ri.RyOffset  = 0;
74  ri.Next      = &ri2; /* Zeiger auf nächstes Playfield */
75
76  ri2.BitMap = &b2; /* BitMap-Structure eintragen */
77  ri2.RxOffset = 0;
78  ri2.RyOffset = 0;
79  ri2.Next = 0;
80
81  InitView(&v); /* Neuen View initialisieren */
82  v.ViewPort = &vp; /* und ViewPort eintragen */
83
84  InitVPort(&vp); /* ViewPort initialisieren */
85
86  cm = (struct ColorMap *)GetColorMap(10); /* ColorMap bereitstellen */
87  colorpalette = (UWORD *)cm->ColorTable; /* und eintragen */
88  for(i=0; i<10; i++)
89   *colorpalette++ = colors[i]; /* Farben setzen */
90
91  vp.ColorMap = cm; /* ColorMap eintragen */
92
93  vp.DWidth  = 320; /* Breite des Ausschnittes */
94  vp.DHeight = 256;
95  vp.RasInfo = &ri; /* RasInfo eintragen */
96  vp.Modes   = DUALPF; /* Dual-Playfield-Mode einschalten */
97
98  MakeVPort( &v, &vp ); /* ViewPort eintragen */
99  MrgCop(&v); /* View in Copperliste eintragen */
100
```

```
101  InitRastPort(&rp); /* RastPort initialisieren */
102  rp.BitMap = &b; /* BitMap-Structure eintragen */
103
104  InitRastPort(&rp2); /* zweiten RastPort initialisieren */
105  rp2.BitMap = &b2; /* zweite BitMap-Structure eintragen */
106
107  SetRast(&rp,0);
108  SetRast(&rp2,0);
109  LoadView(&v); /* Neuen View darstellen */
110
111  SetAPen(&rp2,1); /* Muster zeichnen */
112  for(x=0; x<320; x+=16)
113   {
114    Move(&rp2,0,319-x);
115    Draw(&rp2,x,319);
116    Move(&rp2,x,0);
117    Draw(&rp2,319,319-x);
118   }
119
120  SetAPen(&rp,1);
121  for(x=0; x<400; x+=20)
122   {
123    Move(&rp,0,x);
124    Draw(&rp,x,0);
125    Move(&rp,x,399);
126    Draw(&rp,399,x);
127   }
128
129  for(w = 0; w < 70; w++) /* 1. Playfield nach links bewegen */
130   {
131    ri.RxOffset++;
132    MakeVPort(&v, &vp);
133    MrgCop(&v);
134    LoadView(&v);
135    WaitTOF();
136   }
137
138  for(w = 0; w < 130; w++) /* 1. Playfield nach oben bewegen */
139   {
140    ri.RyOffset++;
141    MakeVPort(&v, &vp);
142    MrgCop(&v);
143    LoadView(&v);
144    WaitTOF();
145   }
146
147  for(w = 0; w < 70; w++) /* 1. Playfield nach rechts bewegen */
148   {
149    ri.RxOffset--;
150    MakeVPort(&v, &vp);
```

```
151     MrgCop(&v);
152     LoadView(&v);
153     WaitTOF();
154    }
155
156   for(w = 0; w < 130; w++) /* 1. Playfield nach unten bewegen */
157    {
158     ri.RyOffset--;
159     MakeVPort(&v, &vp);
160     MrgCop(&v);
161     LoadView(&v);
162     WaitTOF();
163    }
 64
165   LoadView(oldview); /* Alten View darstellen */
166
167   for(i=0; i<1; i++)       /* Bitplanes löschen */
168    {
169     FreeRaster(b.Planes[i],400,300);
170     FreeRaster(b2.Planes[i],320,200);
171    }
172   FreeColorMap(cm); /* ColorMap löschen */
173   FreeVPortCopLists(&vp); /* Copperliste löschen */
174
175   CloseLibrary(GfxBase); /* Library schließen */
176 }


  1  /*******************************
  2
  3  3. Playfield-Demonstration
  4  last update 16/02/88
  5  von Frank Kremser und Jörg Koch
  6  © Markt & Technik 1988
  7
  8  *******************************
  9
 10  Diese Demo zeigt ein einzelnes Playfield mit einer Größe von 960 x 600
 11  Der Unterschied zur 1. Playfielddemonstration besteht darin, das die
 12  volle Auflösung von 352 x 362 Punkten zur Darstellung verwendet wird.
 13
 14  *******************************/
 15
 16  #include <exec/types.h>          /* Include-Files laden */
 17  #include <exec/memory.h>
 18  #include <graphics/gfxbase.h>
 19  #include <graphics/copper.h>
 20  #include <graphics/view.h>
 21  #include <graphics/rastport.h>
 22  #include <devices/gameport.h>
```

```
23 #include <devices/inputevent.h>
24
25 UWORD colors[] =
26  {
27   0,
28   0x555,
29   0xAAA,
30   0xFFF
31  };
32
33 struct View       v,*oldview;
34 struct ViewPort   vp;
35 struct ColorMap   *cm;
36 struct RasInfo    ri;
37 struct BitMap     *bm;
38 struct RastPort   rp;
39
40 struct GfxBase    *GfxBase; /* Library-Zeiger */
41
42
43 main() /* HAUPTPROGRAMM */
44 {
45  int i, x, y;
46                              /* Library öffnen */
47  if (!(GfxBase = OpenLibrary ("graphics.library", 0)))
48   exit();
49            /* Speicher für BitMap-Structure bereitstellen */
50  if (!(bm = AllocMem ((long) sizeof (*bm), MEMF_CHIP|MEMF_CLEAR)))
51   exit();
52
53  InitView (&v);  /* Neue View-Structure initialisieren */
54  InitVPort (&vp);/* Neue Viewport-Structure initialisieren */
55  InitBitMap (bm, (long) 2, 960, 600); /* BitMap-Structure init. */
56  InitRastPort (&rp); /* Rastport-Structure initialisieren */
57
58  v.ViewPort = &vp; /* ViewPort eintragen */
59  ri.BitMap = bm; /* BitMap eintragen */
60  ri.RxOffset = ri.RyOffset = ri.Next = NULL;
61
62  vp.DWidth = 352; /* dargestellte Ausschnittgröße festlegen */
63  vp.DHeight = 262;
64  vp.RasInfo = &ri; /* RasInfo-Structure eintragen */
65  vp.ColorMap = GetColorMap(4); /* Color-Map eintragen und init. */
66
67  rp.BitMap = bm; /* BitMap eintragen */
68
69  for (i=0; i<2; i++) /* Speicher für Bitplanes bereitstellen */
70   if (!(bm -> Planes[i] = AllocRaster (960, 600)))
71    exit();
72
```

```
 73  MakeVPort (&v, &vp); /* ViewPort in View-Structure eintragen */
 74  MrgCop (&v); /* View in Copperliste eintragen */
 75  LoadRGB4 (&vp, colors, 4); /* Farben setzen */
 76  oldview = GfxBase->ActiView; /* Alten View sichern */
 77  LoadView (&v); /* Neuen View darstellen */
 78
 79  i = 1; /*                Muster zeichnen */
 80  SetDrMd (&rp, JAM1);
 81  SetRast (&rp, 0);
 82  SetAPen (&rp, 1);
 83  for (x=0, y=0; x<960; x += 8, y += 5)
 84   {
 85    Move (&rp, (long) x, 0);
 86    Draw (&rp, 959, (long) y);
 87    Draw (&rp, 959-x, 599);
 88    Draw (&rp, 0, 599-y);
 89    Draw (&rp, (long) x, 0L);
 90    if (!(++i & 3)) i++;
 91    SetAPen (&rp, (long) i);
 92   }
 93  SetAPen (&rp, 3);                /* Text zeichnen */
 94  Move (&rp, 429, 301);
 95  Text (&rp, "Ein Superplayfield", 18);
 96  SetAPen (&rp, 1);
 97  Move (&rp, 428, 300);
 98  Text (&rp, "Ein Superplayfield", 18);
 99
100  for(x=0;x<608;x+=4)          /* Playfield nach links bewegen */
101   {
102    ri.RxOffset = x;
103    WaitTOF ();
104    ScrollVPort (&vp);
105   }
106  for(y=0;y<238;y+=2)          /* Playfield nach oben bewegen */
107   {
108    ri.RyOffset = y;
109    WaitTOF ();
110    ScrollVPort (&vp);
111   }
112  for(x=608;x>0;x-=4)          /* Playfield nach rechts bewegen */
113   {
114    ri.RxOffset = x;
115    WaitTOF ();
116    ScrollVPort (&vp);
117   }
118  for(y=238;y>0;y-=2)          /* Playfield nah unten bewegen */
119   {
120    ri.RyOffset = y;
121    WaitTOF ();
122    ScrollVPort (&vp);
```

```
123   }
124  y=12;
125  for(x=0;x<608;x+=4)          /* Playfield diagonal bewegen */
126   {
127    y+=1;
128    ri.RxOffset = x;
129    ri.RyOffset = y;
130    WaitTOF();
131    ScrollVPort(&vp);
132   }
133  LoadView(oldview); /* Alten View setzen */
134  WaitTOF(); /* Warten, bis oberer Bildschirmrand erreicht */
135  FreeVPortCopLists(&vp); /* Alte Copperlisten löschen */
136  FreeCprList(v.LOFCprList);
137  FreeColorMap(vp.ColorMap); /* Colormap löschen */
138  for(i=0; i<2; i++) /* Bitplanes löschen */
139   if(bm->Planes[i])
140    FreeRaster(bm->Planes[i], 960, 600);
141  FreeMem(bm, (long)sizeof(*bm)); /* BitMap-Structure löschen */
142
143  CloseLibrary(GfxBase); /* Library schließen */
144 }
```

```
  1 ;*******************************
  2 ;
  3 ; 1. Playfield - Demonstration
  4 ; last update 10/03/88
  5 ; von Frank Kremser und Jörg Koch
  6 ; © Markt & Technik 1988
  7 ;
  8 ;*******************************
  9 ;
 10 ;Diese Demonstration erstellt vier Bitplanes mit der Größe 640 x 256.
 11 ;Der DisplayScreen ist aber lediglich 320 x 256 Punkte groß.
 12 ;Das Programm verschiebt nun den Bitplaneausschnitt ständig
 13 ;bitweise in horizontaler Richtung.
 14 ;
 15 ;*******************************
 16
 17       BitPlane1     = $50000
 18       BitPlane2     = $58000
 19       BitPlane3     = $60000
 20       BitPlane4     = $68000
 21       ExecBase      = 4
 22       Permit        = -138
 23       Forbid        = -132
 24       OpenLibrary   = -408
 25       CloseLibrary  = -414
 26       InitRastPort  = -198
```

```
27        InitBitMap   = -390
28
29            move.l  ExecBase,a6
30            jsr     Forbid(a6)          ;Multitasking aus
31
32            move.l  ExecBase,a6
33            lea.l   dosname,a1          ;DosLibrary öffnen
34            jsr     OpenLibrary(a6)
35            move.l  d0,dosbase
36
37            lea     gfxname,a1
38            jsr     OpenLibrary(a6)     ;GfxLibrary öffnen
39            move.l  d0,gfxbase
40            move.l  d0,a6
41            add.l   #$32,d0
42            move.l  d0,copptr           ;Zeiger auf alte Copperliste
43
44            lea     bitmap,a0           ;Bitmap-Structure initialisieren
45            moveq   #4,d0               ;Tiefe(4 Bitplanes)
46            move.l  #320,d1             ;Breite
47            move.l  #256,d2             ;Höhe
48            jsr     InitBitMap(a6)      ;Bitmap initialisieren
49            lea     rastport,a1
50            jsr     InitRastPort(a6)    ;Rastport initialisieren
51            move.l  #bitmap,R_BitMap
52            move.l  #bitplane1,d0       ;1. Bitplane initialisieren
53            move.l  d0,plane1
54            move.l  d0,a0
55            move.w  d0,lo1              ;Zeiger in Copperliste aktual.
56            swap d0                     ;Hi-Byte ebenfalls
57            move.w  d0,hi1              ;in Copperliste eintragen
58            move.l  #20480,d0           ;Bitplane löschen
59            move.b  #$00,d1
60  loop1:    move.b  d1,(a0)+
61            dbf     d0,loop1
62
63            move.l  #bitplane2,d0
64            move.l  d0,plane2
65            move.l  d0,a0
66            move.w  d0,lo2
67            swap    d0
68            move.w  d0,hi2
69            move.l  #20480,d0
70            move.b  #$00,d1
71  loop2:    move.b  d1,(a0)+
72            dbf     d0,loop2
73
74            move.l  #bitplane3,d0
75            move.l  d0,plane3
76            move.l  d0,a0
```

```
 77           move.w  d0,lo3
 78           swap    d0
 79           move.w  d0,hi3
 80           move.l  #20480,d0
 81           move.b  #$00,dl
 82 loop3:    move.b  dl,(a0)+
 83           dbf     d0,loop3
 84
 85           move.l  #bitplane4,d0
 86           move.l  d0,plane4
 87           move.l  d0,a0
 88           move.w  d0,lo4
 89           swap    d0
 90           move.w  d0,hi4
 91           move.l  #20480,d0
 92           move.b  #$00,dl
 93 loop4:    move.b  dl,(a0)+
 94           dbf     d0,loop4
 95
 96           move.l  copptr,a0          ;Adresse der Copperliste nach a0
 97           move.l  (a0),oldcop        ;Alte Liste sichern
 98           move.l  #copper,(a0)       ;Neue Copperliste setzen
 99
100           move.w  #10,flag           ;Variablen mit Startwerten
101           move.w  lol,adrl           ;initialisieren
102           move.w  lo2,adr2
103           move.w  lo3,adr3
104           move.w  lo4,adr4
105           move.l  #bitplanel,al
106           move.l  #bitplane2,a2
107           move.l  #bitplane3,a3
108           move.l  #bitplane4,a4
109           move    #256,d0            ;Grafik in die Bitplane zeichnen
110 loop7:    move    #20,dl
111 loop8:    move.l  #$ffff0000,(al)+
112           move.l  #$00ff00ff,(a2)+
113           move.l  #$ff000000,(a3)+
114           move.l  #$ffff00ff,(a4)+
115           sub     #1,dl
116           bne     loop8
117           sub     #1,d0
118           bne     loop7
119
120 loop9:    andi.b  #64,$bfe001        ;Wurde linke Maustaste gedrückt
121           beq     ende               ;Wenn ja, dann beenden
122
123           move.l  #2000,d0           ;Warteschleife
124 loop10:   sub     #1,d0
125           bne     loop10
126
```

```
127 ;Die folgende Routine scrollt das Display bitweise horizontal.
128 ;Soll vertikal gescrollt werden, benötigt man die Finescrolling-
129 ;routine nicht(die ersten 3 Zeilen), sondern man braucht zu
130 ;lol - lo4 lediglich die Breite der Bitplane in Bytes
131 ;hinzuaddieren. Natürlich muß die Bitplane auch entsprechend
132 ;höher sein, als jetzt.
133
134           sub.w   #$11,finescrl      ;Even/OddScrollvalue dekrementieren
135           cmp.w   #$0000,finescrl    ;Wenn noch keine 16 Bits gescrollt,
136           bne     loop9              ;dann weiter scrollen
137           add.w   #2,lol             ;ansonsten um 2 Byte weiterscrollen
138           add.w   #2,lo2
139           add.w   #2,lo3
140           add.w   #2,lo4
141           move.w  #$00ff,finescrl    ;und Even/Odd Scrollvalue zurücksetzen
142           sub.w   #1,flag            ;Wenn Ende des BitMaps erreicht,
143           bne     loop9
144           move.w  #20,flag           ;dann Werte auf Startposition
145           move.w  adrl,lol           ;zurücksetzen
146           move.w  adr2,lo2
147           move.w  adr3,lo3
148           move.w  adr4,lo4
149           bra     loop9
150
151 ende:     move.l  copptr,a0
152           move.l  oldcop,(a0)        ;Alte Copperliste wieder setzen
153           move.l  ExecBase,a6
154           move.l  gfxbase,a1         ;Gfxlibrary schließen
155           jsr     CloseLibrary(a6)
156           move.l  dosbase,a1         ;DosLibrary schließen
157           jsr     CloseLibrary(a6)
158           move.l  ExecBase,a6
159           jsr     Permit(a6)         ;Multitasking einschalten
160           rts                        ;Rückkehr
161
162 even
163 Copper:   dc.w    $0180,$0000        ;Color0
164           dc.w    $0182,$0fff        ;Color1
165           dc.w    $0184,$000f        ;Color2
166           dc.w    $0186,$0f00        ;Color3
167           dc.w    $0188,$000f        ;Color4
168           dc.w    $018a,$0f0f        ;Color5
169           dc.w    $018c,$00ff        ;Color6
170           dc.w    $018e,$f0ff        ;Color7
271           dc.w    $0190,$0620        ;Color8
172           dc.w    $0192,$0e50        ;Color9
173           dc.w    $0194,$09f1        ;Color10
174           dc.w    $0196,$0eb0        ;Color11
175           dc.w    $0198,$055f        ;Color12
176           dc.w    $019a,$092f        ;Color13
```

```
177             dc.w    $019c,$00f8         ;Color14
178             dc.w    $019e,$0ccc         ;Color15
179             dc.w    $01a0,$0000         ;Color16
180             dc.w    $01a2,$0d22         ;Color17
181             dc.w    $01a4,$0000         ;Color18
182             dc.w    $01a6,$0fca         ;Color19
183             dc.w    $01a8,$0444         ;Color20
184             dc.w    $01aa,$0555         ;Color21
185             dc.w    $01ac,$0666         ;Color22
186             dc.w    $01ae,$0777         ;Color23
187             dc.w    $01b0,$0888         ;Color24
188             dc.w    $01b2,$0999         ;Color25
189             dc.w    $01b4,$0aaa         ;Color26
190             dc.w    $01b6,$0bbb         ;Color27
191             dc.w    $01b8,$0ccc         ;Color28
192             dc.w    $01ba,$0ddd         ;Color29
193             dc.w    $01bc,$0eee         ;Color30
194             dc.w    $01be,$0fff         ;Color31
195             dc.w    $00e0               ;BitPlane Pointer
196 hil:       dc.w    $0002
197             dc.w    $00e2
198 lol:       dc.w    $1000
199             dc.w    $00e4
200 hi2:       dc.w    $0002
201             dc.w    $00e6
202 lo2:       dc.w    $1000
203             dc.w    $00e8
204 hi3:       dc.w    $0002
205             dc.w    $00ea
206 lo3:       dc.w    $1000
207             dc.w    $00ec
208 hi4:       dc.w    $0002
209             dc.w    $00ee
210 lo4:       dc.w    $1000
211             dc.w    $0100,%0100000000000000     ;BPLCON0: 4 Bitplanes
212             dc.w    $0102                       ;BPLCON1
213 finescrl:
214             dc.w    $00ff
215             dc.w    $0108,$0026         ;BPL1MOD: Anzahl der Bytes, die am
216             dc.w    $010a,$0026         ;BPL2MOD: Ende der Zeile hinzuaddiert
217             dc.w    $0092,$0030         ;DDFSTRT; werden sollen, damit der
218             dc.w    $0094,$00d0         ;DDFSTOP; neue Zeilenanfang stimmt
219             dc.w    $008e,$2481         ;DIWSTRT; so können Bitplanes breiter
220             dc.w    $0090,$24c1         ;DIWSTOP; sein, als das Display
221             dc.w    $ffff,$fffe         ;Ende der Copperliste
222 BitMap:
223 BytesPerRow:
224             blk.w   1,0
225 Bytes:      blk.w   1,0
226 Flags:      blk.b   1,0
```

```
227 Depth:    blk.b   1,0
228 Pad:      blk.w   1,0
229 Planel:   blk.l   1,0
230 Plane2:   blk.l   1,0
231 Plane3:   blk.l   1,0
232 Plane4:   blk.l   1,0
233 Planes:   blk.l   1,0
234 RastPort:
235           blk.l   1,0
236 R_BitMap:
237           blk.l   1,0
238           blk.b   2+4+4+4+8,0
239           blk.b   4,0
240 cp_x:     blk.w   1,0
241 cp_y:     blk.w   1,0
242           blk.b   8+22+[7*2]+[2*4]+8,0
243           blk.w   6,0
244 even
245 gfxname:  dc.b    'graphics.library',0
246 dosname:  dc.b    'dos.library',0
247
248 even
249 copptr:   blk.l   1,0
250 oldcop:   blk.l   1,0
251 gfxbase:  blk.l   1,0
252 dosbase:  blk.l   1,0
253 x:        blk.l   1,0
254 flag:     blk.l   1,0
255 adrl:     blk.l   1,0
256 adr2:     blk.l   1,0
257 adr3:     blk.l   1,0
258 adr4:     blk.l   1,0
```

```
  1 ;*******************************
  2 ;
  3 ; 2. Playfield - Demonstration
  4 ; last update 10/03/88
  5 ; von Frank Kremser und Jörg Koch
  6 ; © Markt & Technik 1988
  7 ;
  8 ;*******************************
  9 ;
 10 ;Diese Demonstration erstellt 6 Bitplanes der Größe 640 x 256.
 11 ;Dabei werden durch den eingeschalteten Dual-Playfield-Modus
 12 ;die Bitplanes 1, 3 und 5 für das vordere Playfield verwendet und
 13 ;die Planes 2, 4 und 6 für das hintere.
 14 ;Das vordere und das hintere Playfield werden gegenläufig verschoben.
 15 ;
 16 ;*******************************
```

```
17        BitPlane1     =$50000
18        BitPlane2     =$56000
19        BitPlane3     =$5c000
20        BitPlane4     =$62000
21        BitPlane5     =$68000
22        BitPlane6     =$6e000
23        ExecBase      =4
24        Permit        =-138
25        Forbid        =-132
26        OpenLibrary   =-408
27        CloseLibrary  =-414
28        InitRastPort  =-198
29        InitBitMap    =-390
30
31            move.l  ExecBase,a6
32            jsr     Forbid(a6)          ;Multitasking aus
33
34            move.l  ExecBase,a6
35            lea.l   dosname,a1          ;DosLibrary öffnen
36            jsr     OpenLibrary(a6)
37            move.l  d0,dosbase
38
39            lea     gfxname,a1
40            jsr     OpenLibrary(a6)     ;GfxLibrary öffnen
41            move.l  d0,gfxbase
42            move.l  d0,a6
43            add.l   #$32,d0
44            move.l  d0,copptr           ;Zeiger auf alte Copperliste
45
46            lea     bitmap,a0           ;Bitmap-Structure initialisieren
47            moveq   #6,d0               ;Tiefe(6 Bitplanes)
48            move.l  #320,d1             ;Breite
49            move.l  #256,d2             ;Höhe
50            jsr     InitBitMap(a6)      ;Bitmap initialisieren
51            lea     rastport,a1
52            jsr     InitRastPort(a6)    ;Rastport initialisieren
53            move.l  #bitmap,R_BitMap
54
55
56            move.l  #bitplane1,d0       ;1. Bitplane initialisieren
57            move.l  d0,plane1
58            move.l  d0,a0
59            move.w  d0,lo1              ;Zeiger in Copperliste aktualisieren
60            swap    d0                  ;Hi-Byte ebenfalls
61            move.w  d0,hi1              ;in Copperliste eintragen
62            move.l  #20480,d0           ;Bitplane löschen
63            move.b  #$00,d1
64  loop1:    move.b  d1,(a0)+
65            dbf     d0,loop1
66
```

```
 67           move.l  #bitplane2,d0
 68           move.l  d0,plane2
 69           move.l  d0,a0
 70           add.l   #40,d0              ;Bei 2. Playfield die rechte Hälfte
 71           move.w  d0,lo2              ;zuerst anzeigen
 72           swap    d0
 73           move.w  d0,hi2
 74           move.l  #20480,d0
 75           move.b  #$00,dl
 76 loop2:    move.b  dl,(a0)+
 77           dbf     d0,loop2
 78
 79           move.l  #bitplane3,d0
 80           move.l  d0,plane3
 81           move.l  d0,a0
 82           move.w  d0,lo3
 83           swap    d0
 84           move.w  d0,hi3
 85           move.l  #20480,d0
 86           move.b  #$00,dl
 87 loop3:    move.b  dl,(a0)+
 88           dbf     d0,loop3
 89
 90           move.l  #bitplane4,d0
 91           move.l  d0,plane4
 92           add.l   #40,d0              ;Bei 2. Playfield rechte Hälfte
 93           move.w  d0,lo4              ;zuerst anzeigen
 94           swap    d0
 95           move.w  d0,hi4
 96           move.l  #20480,d0
 97           move.b  #$00,dl
 98 loop4:    move.b  dl,(a0)+
 99           dbf     d0,loop4
100
101           move.l  #bitplane5,d0
102           move.l  d0,plane5
103           move.l  d0,a0
104           move.w  d0,lo5
105           swap    d0
106           move.w  d0,hi5
107           move.l  #20480,d0
108           move.b  #$00,dl
109 loop5:    move.b  dl,(a0)+
110           dbf     d0,loop5
111
112           move.l  #bitplane6,d0
113           move.l  d0,plane6
114           move.l  d0,a0
115           add.l   #40,d0              ;Bei 2. Playfield rechte Hälfte
116           move.w  d0,lo6              ;zuerst anzeigen
```

```
117             swap    d0
118             move.w  d0,hi6
119             move.l  #20480,d0
120             move.b  #$00,dl
121 loop6:      move.b                      dl,(a0)+
122             dbf     d0,loop6
123
124             move.l  copptr,a0           ;Adresse der Copperliste nach a0
125             move.l  (a0),oldcop         ;Alte Liste sichern
126             move.l  #copper,(a0)        ;Neue Copperliste setzen
127             move.w  #10,flag            ;Variablen mit Startwerten
128             move.w  lol,adrl            ;initialisieren
129             move.w  lo2,adr2
130             move.w  lo3,adr3
131             move.w  lo4,adr4
132             move.w  lo5,adr5
133             move.w  lo6,adr6
134             move.l  #bitplanel,al
135             move.l  #bitplane2,a2
136             move.l  #bitplane3,a3
137             move.l  #bitplane4,a4
138             move.l  #bitplane5,a5
139             move.l  #bitplane6,a6
140             move    #256,d0
141 loop7:      move    #20,dl
142 loop8:      move.l  #$0f0000f0,(al)+    ;Grafik für vorderes Playfield
143             move.l  #$ff0000ff,(a3)+
144             move.l  #$f000000f,(a5)+
145             move.l  #$ffff0000,(a2)+    ;Grafik für hinteres Playfield
146             move.l  #$ffff0000,(a4)+
147             move.l  #$0000ffff,(a6)+
148             sub     #1,dl
149             bne     loop8
150             sub     #1,d0
151             bne     loop7
152
153 loop9:      andi.b  #64,$bfe001         ;Wurde linke Maustaste gedrückt
154             beq     ende                ;Wenn ja, dann beenden
155
156             move.l  #4000,d0            ;Warteschleife
157 loopl0:     sub     #1,d0
158             bne     loopl0
159
160             sub.w   #1,finescrl         ;Scrollvalue für OddPlane erniedrigen
161             add.w   #$10,finescrl       ;Scrollvalue für EvenPlane erhöhen
162             cmp.w   #$00f0,finescrl     ;Wenn noch keine 16 Bits gescrollt,
163             bne     loop9               ;dann weiter scrollen
164             add.w   #2,lol              ;ansonsten um 2 Byte weiterscrollen
165             add.w   #2,lo3              ;Odd Planes
166             add.w   #2,lo5
```

```
167             sub.w   #2,lo2             ;Even Planes
168             sub.w   #2,lo4
169             sub.w   #2,lo6
170             move.w  #$000f,finescrl    ;und Odd Modulo zurücksetzen
171             sub.w   #1,flag            ;Wenn Ende des BitMaps erreicht,
172             bne     loop9
173             move.w  #20,flag           ;dann Werte auf Startposition
174             move.w  adr1,lo1           ;zurücksetzen
175             move.w  adr3,lo3
176             move.w  adr5,lo5
177             move.w  adr2,lo2
178             move.w  adr4,lo4
179             move.w  adr6,lo6
180             bra     loop9
181
182 ende:       move.l  copptr,a0
183             move.l  oldcop,(a0)        ;Alte Copperliste wieder setzen
184             move.l  ExecBase,a6
185             move.l  gfxbase,a1         ;Gfxlibrary schließen
186             jsr     CloseLibrary(a6)
187             move.l  dosbase,a1         ;DosLibrary schließen
188             jsr     CloseLibrary(a6)
189             move.l  ExecBase,a6
190             jsr     Permit(a6)         ;Multitasking einschalten
191             rts     ;Rückkehr
192
193 even
194 Copper:
195             dc.w    $0180,$0000        ;Color0
196             dc.w    $0182,$0fff        ;Color1
197             dc.w    $0184,$000f        ;Color2
198             dc.w    $0186,$0f00        ;Color3
199             dc.w    $0188,$000f        ;Color4
200             dc.w    $018a,$0f0f        ;Color5
201             dc.w    $018c,$00ff        ;Color6
202             dc.w    $018e,$f0ff        ;Color7
203             dc.w    $0190,$0620        ;Color8
204             dc.w    $0192,$0e50        ;Color9
205             dc.w    $0194,$09f1        ;Color10
206             dc.w    $0196,$0eb0        ;Color11
207             dc.w    $0198,$055f        ;Color12
208             dc.w    $019a,$092f        ;Color13
209             dc.w    $019c,$00f8        ;Color14
210             dc.w    $019e,$0ccc        ;Color15
211             dc.w    $01a0,$0000        ;Color16
212             dc.w    $01a2,$0d22        ;Color17
213             dc.w    $01a4,$0000        ;Color18
214             dc.w    $01a6,$0fca        ;Color19
215             dc.w    $01a8,$0444        ;Color20
216             dc.w    $01aa,$0555        ;Color21
```

```
217             dc.w    $01ac,$0666         ;Color22
218             dc.w    $01ae,$0777         ;Color23
219             dc.w    $01b0,$0888         ;Color24
220             dc.w    $01b2,$0999         ;Color25
221             dc.w    $01b4,$0aaa         ;Color26
222             dc.w    $01b6,$0bbb         ;Color27
223             dc.w    $01b8,$0ccc         ;Color28
224             dc.w    $01ba,$0ddd         ;Color29
225             dc.w    $01bc,$0eee         ;Color30
226             dc.w    $01be,$0fff         ;Color31
227             dc.w    $00e0               ;BitPlane Pointer
228 hil:        dc.w    $0002
229             dc.w    $00e2
230 lol:        dc.w    $1000
231             dc.w    $00e4
232 hi2:        dc.w    $0002
233             dc.w    $00e6
234 lo2:        dc.w    $1000
235             dc.w    $00e8
236 hi3:        dc.w    $0002
237             dc.w    $00ea
238 lo3:        dc.w    $1000
239             dc.w    $00ec
240 hi4:        dc.w    $0002
241             dc.w    $00ee
242 lo4:        dc.w    $1000
243             dc.w    $00f0
244 hi5:        dc.w    $0002
245             dc.w    $00f2
246 lo5:        dc.w    $1000
247             dc.w    $00f4
248 hi6:        dc.w    $0002
249             dc.w    $00f6
250 lo6:        dc.w    $2000
251             dc.w    $0100,%0110010000000000  ;BPLCON0: 6 BitPl. + DUALPF
252             dc.w    $0102                    ;BPLCON1 finescrl:
253             dc.w    $000f
254             dc.w    $0108,$0026         ;BPL1MOD
255             dc.w    $010a,$0026         ;BPL2MOD
256             dc.w    $0092,$0030         ;DDFSTRT
257             dc.w    $0094,$00d0         ;DDFSTOP
258             dc.w    $008e,$2481         ;DIWSTRT
259             dc.w    $0090,$24c1         ;DIWSTOP
260             dc.w    $ffff,$fffe         ;Ende der Copperliste
261
262 BitMap:
263 BytesPerRow:
264             blk.w   1,0
265 Bytes:      blk.w   1,0
266 Flags:      blk.b   1,0
```

```
267 Depth:    blk.b   1,0
268 Pad:      blk.w   1,0
269 Planel:   blk.l   1,0
270 Plane2:   blk.l   1,0
271 Plane3:   blk.l   1,0
272 Plane4:   blk.l   1,0
273 Plane5:   blk.l   1,0
274 Plane6:   blk.l   1,0
275 Planes:   blk.l   1,0
276 RastPort:
277           blk.l   1,0
278 R_BitMap:
279           blk.l   1,0
280           blk.b   2+4+4+4+8,0
281           blk.b   4,0
282 cp_x:     blk.w   1,0
283 cp_y:     blk.w   1,0
284           blk.b   8+22+[7*2]+[2*4]+8,0
285           blk.w   6,0
286 even
287 gfxname:  dc.b    'graphics.library',0dosname:
288           dc.b    'dos.library',0
289
290 even
291 copptr:   blk.l   1,0
292 oldcop:   blk.l   1,0
293 gfxbase:  blk.l   1,0
294 dosbase:  blk.l   1,0
295 x:        blk.l   1,0
296 flag:     blk.l   1,0
297 adrl:     blk.l   1,0
298 adr2:     blk.l   1,0
299 adr3:     blk.l   1,0
300 adr4:     blk.l   1,0
301 adr5:     blk.l   1,0
302 adr6:     blk.l   1,0
```

4.2.4: Die Video-Prioritätsregister

Der Bildschirm hat normalerweise nur zwei Dimensionen. Aber es existiert noch eine imaginäre dritte Dimension. In dieser Dimension ist festgelegt, in welcher Reihenfolge die *Playfields* und Sprites dargestellt werden.

Die Reihenfolge der Spritedarstellung kann nicht geändert werden. Das bedeutet, daß Sprite 0 immer vor Sprite 1, dieses wiederum vor *Sprite* 2 usw. dargestellt wird. Verändert werden kann allerdings die Videopriorität der Sprites zu den Playfields. Ist nur ein normaler Screen dargestellt, so ist dieser als Playfield 1 aufzufassen.

Zu den Sprites ist anzumerken, daß sie in Zweiergruppen aufgeteilt sind. Das bedeutet, Sprite 0 und 1, 2 und 3, 4 und 5, sowie 6 und 7 bilden jeweils eine Gruppe. Die Videopriorität kann also immer nur für zwei Sprites zusammen geändert werden. Die Videoprioritäten werden mit Register BPLCON2 festgelegt. Nachfolgend die Funktionen der Bits:

Bit	Name	Funktion
15–7		Nicht benutzt, sollte aber auf 0 gesetzt werden.
6	PF2PRI	Priorität von Playfield 2 zu Playfield 1. Ist dieses Bit gesetzt, wird Playfield 2 vor Playfield 1 dargestellt.
5–3	PF2P2–PF2P0	Playfield 2 Priorität zu den Sprites.
2–0	PF1P2–PF1P0	Playfield 1 Priorität zu den Sprites.

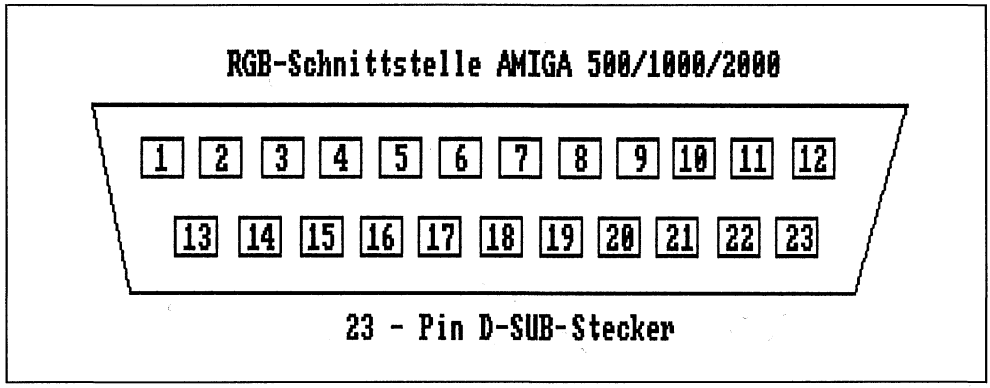
Die Bits PF2P2-PF2P0 und PF1P2-PF1P0 können nach folgender Tabelle gesetzt werden. PF steht dabei für Playfield 1 oder 2, SPxy steht für die Spritegruppe der Sprites x und y. Links steht dabei das Objekt mit der höchsten Videopriorität:

Wert	Darstellung				
000	PF	SP01	SP23	SP45	SP67
001	SP01	PF	SP23	SP45	SP67
010	SP01	SP23	PF	SP45	SP67
011	SP01	SP23	SP45	PF	SP67
100	SP01	SP23	SP45	SP67	PF

4.2.5: Das Video-Interface

Die Video-Signale, die Denise liefert, werden noch durch nachgeschaltete Baugruppen nachbearbeitet, bei den Rechnern Amiga 500 und B2000 beispielsweise durch einen *Video-Hybrid*-Baustein. Dadurch wird erreicht, daß ein *Composite-Video*-, ein *Digital-*

RGB- und ein *Analog-RGB*-Signal zur Verfügung stehen. Das Digital-RGB-Signal steht allerdings nur als 4-Bit-Signal zur Verfügung, so daß nur 15 verschiedene Farben dargestellt werden können. Die 4096 Farben, die Denise über die 12 Digital-Video-Ausgänge zur Verfügung stellt, können nur über einen Analog-Ausgang dargestellt werden, was im Normalfall auch geschieht. Hier die Pinbelegung des RGB-Video-Steckers:



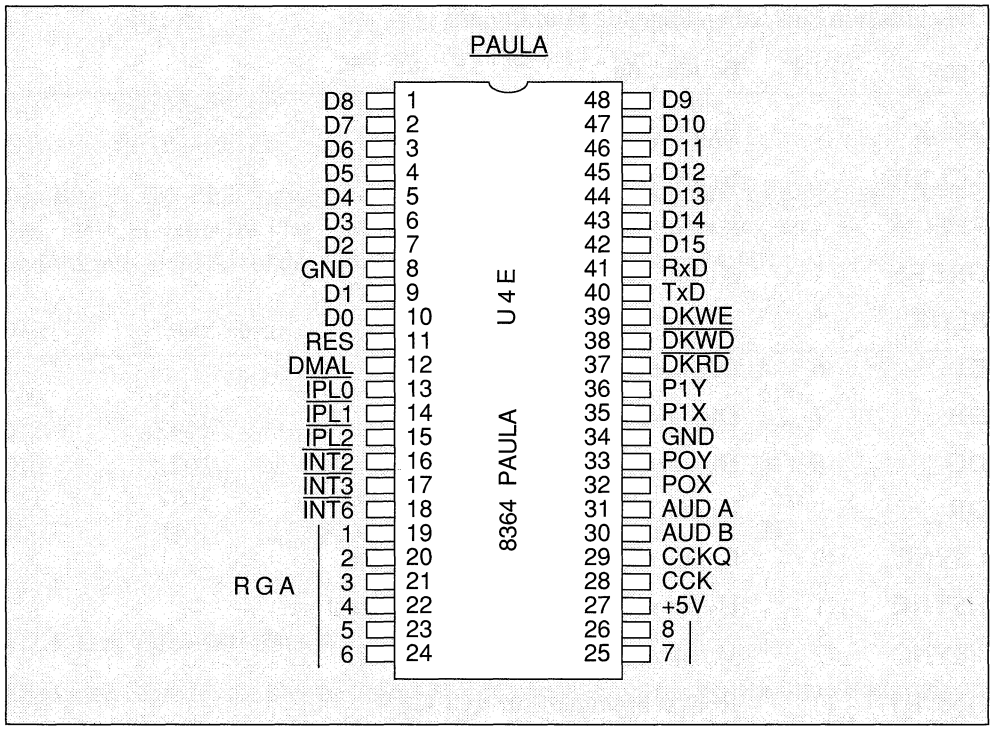
Die einzelnen Pins haben folgende Funktionen:

Name	PIN	Bedeutung
XCLK	1	Externer Takt
XCLKEN	2	Externer Takt »Enable«
RED	3	Analog Rot
GREEN	4	Analog Grün
BLUE	5	Analog Blau
DI	6	Digitale Intensität
DB	7	Digital Blau
DG	8	Digital Grün
DR	9	Digital Rot
CSYNC	10	Composite-Synchronisationssignal
HSYNC	11	Horizontales Synchronisationssignal
VSYNC	12	Vertikales Synchronisationssignal
GNDRTN	13	Rückkehreingang für XCLKEN
ZD	14	Null-Level-Kennzeichnung

Name	PIN	Bedeutung
C1	15	Taktausgang
GND	16–20	Masseanschluß
–12V	21	–12-Volt-Versorgung (50 mA)
+12V	22	+12-Volt-Versorgung (100 mA)
+5V	23	+5-Volt-Versorgung (100 mA)

4.3: Paula

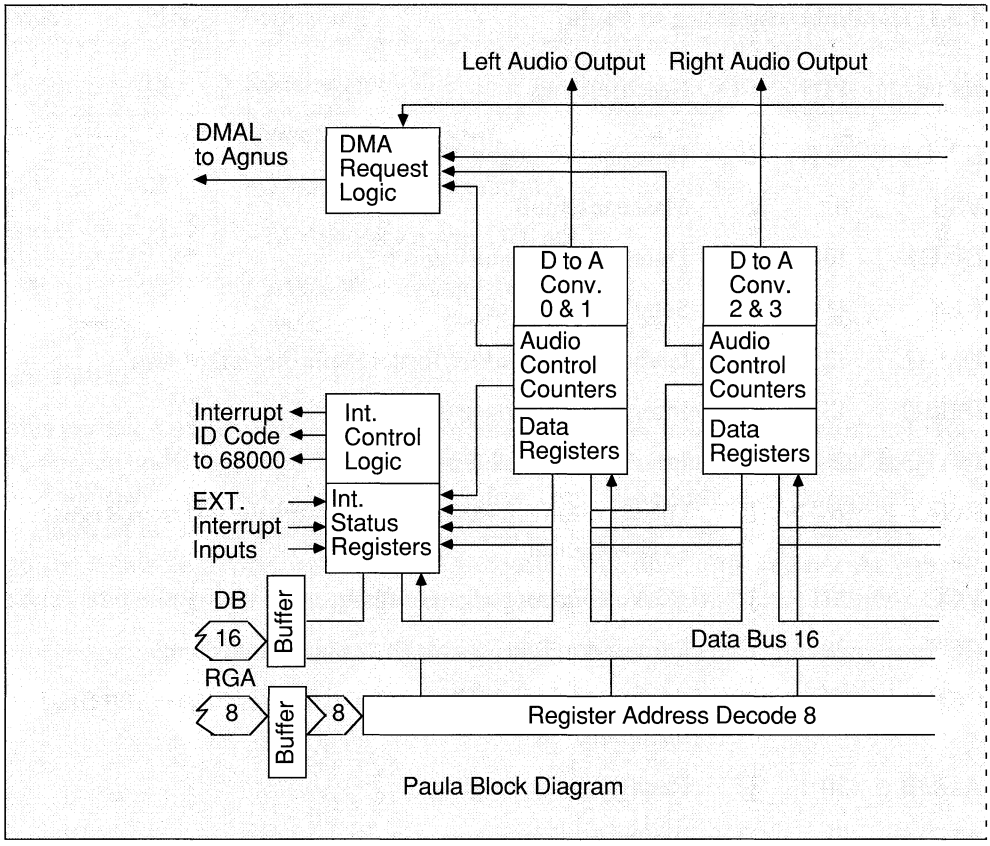
Paula enthält 4 *Audio-Kanäle,* die als Stereoausgänge geschaltet sind, über neun Oktaven reichen und komplexe Schwingungen beherrschen. Zudem besteht die Möglichkeit der *Amplituden-* und der *Frequenz-Modulation*. Der zweite Aufgabenschwerpunkt dieses Chips ist die Disk-Kontrolle. Paula enthält die I/O-Kontrollogik für die Diskdaten und die Kontroller-Ports, sowie einen Microdisk-Controller. Ein weiterer Aufgabenbereich ist die *Interrupt*-Kontrolle für das System (siehe auch Bild 12 im Farbteil).



Z 4.3-1: *Die Pin-Belegung von Paula*

4.3.1: Die Pinbeschreibung zu Paula

Name	PIN	I/O	Beschreibung
D2–D7	7–1	I/O	Datenbusleitungen 2 bis 7.
VSS	8	I	Masseanschluß.
D0,D1	10,9	I/O	Datenbusleitungen 0 und 1.
RES	11	I	Setzt Paula zurück.
DMAL	12	O	Ist dieses Signal aktiv, fordet Paula einen DMA an.
IPL0–2	13–15	O	Interrupt-Leitungen 0 bis 2.
INT2,3,6	16–18	I	Interrupt-Level 2, 3 und 6.
RGA1–8	26–19	I	Diese Leitungen werden benutzt, um die internen Register. zu adressieren.
VCC	27	I	+5-Volt-Versorgungsspannung.
CCK	28	I	Dies ist der Takt, der als Farbträgersignal dient.
CCKQ	29	I	Dies ist der gleiche Takt, wie CCK, allerdings um 90 Grad. nachhängend.
AUDB	30	O	Rechter Audioausgang.
AUDA	31	O	Linker Audioausgang.
POT0X	32	I/O	Anschluß PotX an Port 0.
POT0Y	33	I/O	Anschluß PotY an Port 0.
VSSANA	34	I	Masseanschluß zum Analog-Ausgang.
POT1X	35	I/O	Anschluß PotX an Port 1.
POT1Y	36	I/O	Anschluß PotY an Port 1.
DKRD	37	I	Disk-Read-Leitung.
DKWD	38	O	Disk-Write-Leitung.
DKWE	39	O	Disk-Write-Enable-Leitung.
TXD	40	O	Serielle Übertragungsleitung.
RXD	41	I	Serielle Empfangsleitung.
D9–D15	48–42	I/O	Datenbusleitungen 9 bis 15.

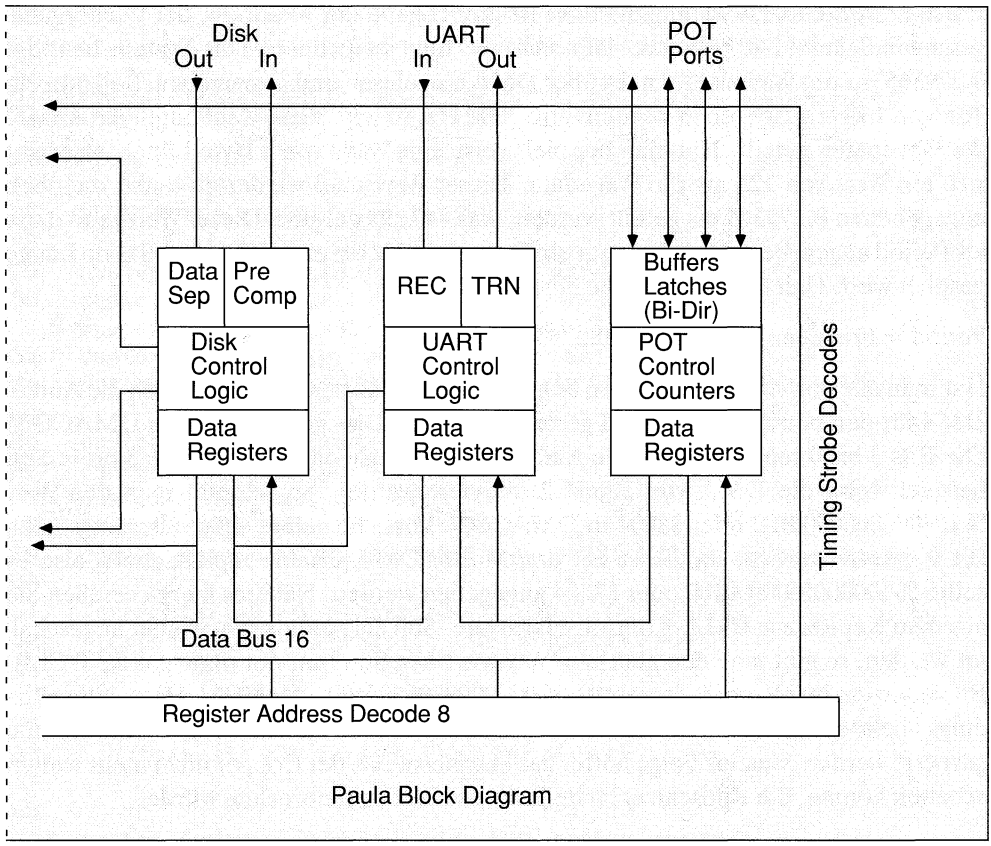


Z 4.3-2: *Das Blockschaltbild von Paula (Teil 1)*

4.3.2: Die Audio-Hardware

Paula verfügt über 4 Audio-Kanäle, die in zwei Gruppen als Stereo-Ausgänge geschaltet sind. Die Kanäle 0 und 3 bilden den linken *Stereo-*, die Kanäle 1 und 2 den rechten Stereo-Ausgang. Alle vier Kanäle sind völlig unabhängig in Lautstärke und Frequenz modulierbar. Jeder Kanal kann sogar über eigene Wavedaten verfügen. Diese Wavedaten stellen wohl den interessantesten Aspekt der Audiohardware des Amiga dar. Durch diese Wavedaten wird festgelegt, ob ein Kanal als Geräuschkanal, Musikkanal o.ä. benutzt wird. Jedes Musikinstrument besitzt beispielsweise eine individuelle *Wave*, die durch den Amiga, wenn sie digitalisiert wurde, imitiert werden kann. So kann jeder Audiokanal ein Instrument imitieren, ein komplettes Musikstück abspielen oder sogar gesprochene Worte wiedergeben.

Um überhaupt etwas über die Audiokanäle wiedergeben zu können, muß also eine solche Wave im Speicher, im ChipMem, vorhanden sein. Die Register AUDxLCH/



Z 4.3-2: *Das Blockschaltbild von Paula (Teil 2)*

AUDxLCL (das x steht für den betreffenden Kanal) müssen auf die Startadresse der Wave zeigen. Die Wave selbst muß Word-weise ausgerichtet und immer eine geradzahlige Anzahl von Bytes lang sein. Jedes Byte repräsentiert dabei einen Teil der Wave. Die Audiohardware arbeitet mit 8 Bit. Jedes Byte wird aber nicht als Wert von 0 bis 255 aufgefaßt, sondern als Wert von –128 bis +127. Wenn beispielsweise ein Ton mit einer Rechteckschwingung gespielt werden soll, so muß das erste Byte der Wave den Wert –127 (oder –128) und das zweite Byte den Wert +127 besitzen. Es geht natürlich auch umgekehrt.

Als nächster Punkt muß das Register AUDxLEN auf die Länge der Wave gesetzt werden, wobei die Länge in Words anzugeben ist. Wenn die Wave also 100 Byte lang ist, muß AUDxLEN auf 50 gesetzt werden. Anschließend wählt man die Lautstärke, die mit AUDxVOL gesetzt wird. Als Lautstärkewerte können Werte zwischen 0 und 64 angegeben werden, wobei 64 die lauteste Wiedergabe bewirkt.

Ein weiterer, äußerst wichtiger Punkt ist die Angabe der »Period«, der Wiedergabegeschwindigkeit. Die »Period« läßt sich wie folgt berechnen: Das System benötigt 0.279365 ms pro Wavedata, um es über DMA einzulesen und auszugeben. Soll nun ein Ton von 1 kHz ausgegeben werden, also 1000 Hz, so wird diese Zahl durch die Anzahl der Wavedaten geteilt. Hat man beispielsweise eine Wave von 8 Byte Länge, so ergibt sich ein Wert von 125 ms pro Wavedata. Dieser Wert muß wiederum durch die oben angegebenen 0.279365 ms geteilt werden, was 447.xxxx ergibt. Dieser Wert muß dann als Period angegeben werden, damit ein Ton von 1 kHz bei einer Wave von 8 Byte Länge gespielt wird. Hier noch einmal die Formel:

Period = Frequenz / Wavelänge / 0.279365

Hat man alle vorher beschriebenen Schritte durchgeführt, so muß nur noch die Audio-*DMA* für den betreffenden Kanal gestartet werden. Dies geschieht durch DMACON. Die Bits 3 bis 0 repräsentieren die Audio-DMA-Kanäle der Audiokanäle 3 bis 0. Soll beispielsweise die DMA von Kanal 2 aktiviert werden, so schreibt man den Wert %1000000000000100 oder $8004 in DMACON. Vorsichtshalber sollte allerdings noch Bit 9 gesetzt werden, da dieses Bit den Master-DMA-Enable repräsentiert, also es sollte %1000001000000100 oder $8204 angegeben werden. Näheres hierzu ersehen Sie aus dem Kapitel zur DMA-Kontroll-Hardware. Soll ein Audiokanal wieder abgeschaltet werden, so gibt man den gleichen Wert wie beim Start an, nur ohne das SET/CLR-Bit zu setzen, im obigen Beispiel also %0000000000000100 oder $0004. Hier sollte allerdings nicht die Master-DMA-Enable gelöscht werden, da dann alle DMA-Kanäle gesperrt werden, was zur Folge hätte, das beispielsweise der Copper nicht mehr weiterarbeiten könnte, die Bildschirmdarstellung also zusammenbrechen würde.

```
 1  /*****************************
 2
 3  1. Audio-Demonstration
 4  last update 16/02/88
 5  von Frank Kremser und Jörg Koch
 6  © Markt & Technik 1988
 7
 8  *****************************
 9
10  Diese Demonstration spielt über zwei Soundkanäle verschiedene Töne.
11  Dabei wird nicht eine Device verwendet, sondern die Kanäle werden direkt
12  angesprochen.
13
14  *****************************/
15
16  #include <exec/types.h>       /* Include-Files laden */
17  #include <hardware/custom.h>
18
19  extern struct Custom custom; /* Externe Structure laden */
```

```
20                                  /* Über sie können die Hardwareregister */
21                                  /* direkt angesprochen werden */
22
23  BYTE data[] = {0,90,127,90,0,-90,-127,-90}; /* Sinus als Wave */
24
25  main()
26  {
27   ULONG warte,schleife,schleife2;
28
29   custom.aud[0].ac_len = 4;   /* Länge der Wave in Words */
30   custom.aud[0].ac_per = 30;  /* Tonhöhe - Period */
31   custom.aud[0].ac_vol = 64;  /* Lautstärke */
32   custom.aud[0].ac_ptr = &data[0]; /* Zeiger auf Wave */
33   custom.dmacon = 0x8000 + 0x200 + 0x1; /* DMA für Kanal 0 starten */
34
35   custom.aud[1].ac_len = 4;
36   custom.aud[1].ac_per = 5;
37   custom.aud[1].ac_vol = 64;
38   custom.aud[1].ac_ptr = &data[0];
39   custom.dmacon = 0x8000 + 0x200 + 0x2; /* DMA für Kanal 1 starten */
40
41   for(schleife2 = 0; schleife2 < 5; schleife2++)
42    for(schleife = 0; schleife < 65; schleife++)
43     {
44      custom.aud[0].ac_per = schleife; /* Period für Kanal 0 und */
45      custom.aud[1].ac_vol = schleife; /* Volume für Kanal 1 ändern */
46      for(warte = 0; warte < 3000; warte++);
47     }
48
49   custom.dmacon = 0x0000 + 0x1; /* DMA abschalten */
50   custom.dmacon = 0x0000 + 0x2;
51  }
```

```
 1  /*****************************
 2
 3  2. Audio-Demonstration
 4  last update 16/02/88
 5  von Frank Kremser und Jörg Koch
 6  © Markt & Technik 1988
 7
 8  *****************************
 9
10  Diese Demonstration liefert in etwa das gleiche Ergebnis wie die 1. Demo,
11  aber die Audio-Hardware wird über die Device angesprochen.
12
13  *****************************/
14
15  #include <exec/types.h>         /* Include-Files laden */
16  #include <exec/memory.h>
```

```
#include <hardware/custom.h>
#include <hardware/dmabits.h>
#include <libraries/dos.h>
#include <devices/audio.h>

extern struct MsgPort *CreatePort();
struct IOAudio soundl,sound2;

UBYTE chan0=0x01,chanl=0x02,chan2=0x04,chan3=0x08,data[128];

main()
{
 UBYTE i,schleife,schleife2;
                          /* Ports erstellen */
 if((soundl.ioa_Request.io_Message.mn_ReplyPort =
                          CreatePort("rp",0))==NULL)
  exit();
 if((sound2.ioa_Request.io_Message.mn_ReplyPort =
                          CreatePort("rp",0))==NULL)
  exit();

 soundl.ioa_Request.io_Message.mn_Node.ln_Pri = 10;
 soundl.ioa_Data = &chan0;                /* Kanal 0 öffnen */
 soundl.ioa_Length = (ULONG)sizeof(chan0);
 if((OpenDevice(AUDIONAME,0,&soundl,0))!=NULL)
  exit();

 sound2.ioa_Request.io_Message.mn_Node.ln_Pri = 10;
 sound2.ioa_Data = &chanl;                /* Kanal 1 öffnen */
 sound2.ioa_Length = (ULONG)sizeof(chanl);
 if((OpenDevice(AUDIONAME,0,&sound2,0))!=NULL)
  exit();

 for(i=0; i<128; i++) /* Sägezahn als Wave */
  data[i] = i;
                    /* Request-Structure erstellen */
 soundl.ioa_Request.io_Command = CMD_WRITE;
 soundl.ioa_Request.io_Flags = ADIOF_PERVOL | IOF_QUICK;
 soundl.ioa_Cycles = 5; /* Dauer des Tones */
 soundl.ioa_Length = sizeof(data); /* Länge der Wave */
 soundl.ioa_Period = 508; /* Tonhöhe */
 soundl.ioa_Volume = 64; /* Lautstärke */
 soundl.ioa_Data = data; /* Zeiger auf Wave */

 sound2.ioa_Request.io_Command = CMD_WRITE;
 sound2.ioa_Request.io_Flags = ADIOF_PERVOL | IOF_QUICK;
 sound2.ioa_Cycles = 5;
 sound2.ioa_Length = sizeof(data); s
 sound2.ioa_Period = 20;
 sound2.ioa_Volume = 64;
```

```
67   sound2.ioa_Data = data;
68
69   for(schleife2=0; schleife2 < 5; schleife2++)
70    for(schleife=0; schleife < 64; schleife++)
71     {
72      soundl.ioa_Period = schleife*10+128; /* Period von Kanal 0 und */
73      sound2.ioa_Volume = schleife;       /* Volume von Kanal 1 ändern */
74      BeginIO(&soundl); /* Ton über Kanal 0 und */
75      BeginIO(&sound2); /* über Kanal 1 spielen */
76      WaitIO(&sound2); /* Warten, bis Ton auf Kanal 1 */
77      WaitIO(&soundl); /* und auf Kanal 0 fertig gespielt */
78     }
79
80   CloseDevice(&soundl); /* Kanal 0 und */
81   CloseDevice(&sound2); /* Kanal 1 schließen */
82  }


 1  /*******************************
 2
 3  3. Audio-Demonstration
 4  last update 16/02/88
 5  von Frank Kremser und Jörg Koch
 6  © Markt & Technik 1988
 7
 8  *******************************
 9
10  Diese Demonstration verdeutlicht die Stereomöglichkeiten der Audio-
11  Hardware. Auf Kanal 0 und 1 wird der gleiche Ton gespielt. Zu Beginn
12  steht Kanal 0 auf voller Lautstärke und Kanal 1 auf 0. Nach und nach wird
13  die Lautstäke von Kanal 0 herabgesetzt und die von Kanal 1 herauf,
14  so daß der Eindruck entsteht, der Ton 'wandere' von links nach rechts.
15
16  *******************************/
17
18  #include <exec/types.h>       /* Include-Files laden */
19  #include <exec/memory.h>
20  #include <hardware/custom.h>
21  #include <hardware/dmabits.h>
22  #include <libraries/dos.h>
23  #include <devices/audio.h>
24
25  extern struct MsgPort *CreatePort();
26  struct IOAudio soundl,sound2;
27
28  UBYTE chan0=0x01,chan1=0x02,chan2=0x04,chan3=0x08,data[128];
29
30
31  main()
32  {
```

```
33  UBYTE i,schleife;
34                              /* Ports einrichten */
35  if((soundl.ioa_Request.io_Message.mn_ReplyPort =
36                                  CreatePort("rp",0))==NULL)
37   exit();
38  if((sound2.ioa_Request.io_Message.mn_ReplyPort =
39                                  CreatePort("rp",0))==NULL)
40   exit();
41
42  soundl.ioa_Request.io_Message.mn_Node.ln_Pri = 10;
43  soundl.ioa_Data = &chan0;                          /* Kanal 0 öffnen */
44  soundl.ioa_Length = (ULONG)sizeof(chan0);
45  if((OpenDevice(AUDIONAME,0,&soundl,0))!=NULL)
46   exit();
47
48  sound2.ioa_Request.io_Message.mn_Node.ln_Pri = 10;
49  sound2.ioa_Data = &chanl;                          /* Kanal 1 öffnen */
50  sound2.ioa_Length = (ULONG)sizeof(chanl);
51  if((OpenDevice(AUDIONAME,0,&sound2,0))!=NULL)
52   exit();
53  for(i=0; i<128; i++)        /* Sägezahn als Wave */
54   data[i] = i;
55                      /* Request-Structure für Kanal 0 */
56  soundl.ioa_Request.io_Command = CMD_WRITE;
57  soundl.ioa_Request.io_Flags = ADIOF_PERVOL | IOF_QUICK;
58  soundl.ioa_Cycles = 50;      /* Dauer des Tones */
59  soundl.ioa_Length = sizeof(data); /* Länge der Wave */
60  soundl.ioa_Period = 200; /* Tonhöhe */
61  soundl.ioa_Volume = 64; /* Lautstärke */
62  soundl.ioa_Data = data; /* Zeiger auf Wave */
63
64  sound2.ioa_Request.io_Command = CMD_WRITE;
65  sound2.ioa_Request.io_Flags = ADIOF_PERVOL | IOF_QUICK;
66  sound2.ioa_Cycles = 50;
67  sound2.ioa_Length = sizeof(data);
68  sound2.ioa_Period = 200;
69  sound2.ioa_Volume = 0;
70  sound2.ioa_Data = data;
71
72  for(schleife=0; schleife < 65; schleife++)
73   {
74    soundl.ioa_Volume = 64-schleife;   /* Lautstärke herab-, */
75    sound2.ioa_Volume = schleife;      /* bzw. heraufregeln */
76    BeginIO(&soundl);                  /* und Ton auf beiden Kanälen */
77    BeginIO(&sound2);                  /* spielen */
78    WaitIO(&sound2);                   /* Warten, bis Ton gespielt */
79   }
80  CloseDevice(&soundl); /* Device schließen */
81  CloseDevice(&sound2);
82 }
```

```
;*******************************
;
; 1. Sound - Demonstration
; last update 10/03/88
; von Frank Kremser und Jörg Koch
; © Markt & Technik 1988
;
;*******************************
;
;Diese Demonstration spielt einen Ton über Kanal 0.
;
;*******************************

          lea.l   wave(pc),a0          ;Zeiger auf Wave
          move.l  a0,$dff0a0           ;In Register von Kanal 0 speichern
          move.w  #4,d0                ;Länge der Wave in Words
          move.w  d0,$dff0a4           ;setzen
          move.w  #240,$dff0a6         ;Tonhöhe (Period) von Kanal 0
          move.w  #64,$dff0a8          ;Lautstärke von Kanal 0
          move.w  #%1000000000000001,$dff096    ;Audio-DMA starten

wait:     andi.b  #64,$bfe001          ;linke Maustaste gedrückt?
          bne     wait                 ;nein, dann weiter warten

          move.w  #%0000000000000001,$dff096   ;Audio-DMA stoppen
          rts                                  ;Rückkehr

even
wave:     DC.B    0,90,127,90,0,-90,-127,-90   ;Wavedaten
```

```
;*******************************
;
; 2. Sound - Demonstration
; last update 10/03/88
; von Frank Kremser und Jörg Koch
; © Markt & Technik 1988
;
;*******************************
;
;Diese Demonstration zwei Töne. Der Ton auf Kanal 0 (links) wird in
;der Tonhöhe und der auf Kanal 1 (rechts) in der Lautstärke ver-
;ändert.
;
;*******************************

          lea.l   wave(pc),a0          ;Zeiger auf Wavedaten
          move.l  a0,$dff0a0           ;in Register für Kanal 0 und
          move.l  a0,$dff0b0           ;Kanal 1 schreiben
          move.w  #4,d0                ;Wavelänge in Register für
```

```
1           move.w  d0,$dff0a4          ;Kanal 0 und
2           move.w  d0,$dff0b4          ;Kanal 1 schreiben
3           move.w  #0,$dff0a6          ;Tonhöhe Kanal 0
4           move.w  #200,$dff0b6        ;Kanal 1 setzen
5           move.w  #64,$dff0a8         ;Lautstäke Kanal 0
6           move.w  #0,$dff0b8          ;Kanal 1 setzen
7           move.w  #%1000000000000011,$dff096   ;Audio-DMA ein
8
9           move.w  #0,d0               ;Variable für Lautstärke
10          move.w  #128,d2             ;für Tonhöhe setzen
11 loopl:   move.w  #25000,d1           ;Variable für Warteschleife
12 waitl:   sub     #1,d1               ;vermindern und wenn = 0,
13          bne     waitl               ;dann weiter
14          move.w  d2,$dff0a6          ;neue Tonhöhe setzen
15          move.w  d0,$dff0b8          ;neue Lautstärke setzen
16          add     #4,d2               ;Tonhöhe um 4 erhöhen (= tiefer)
17          add     #1,d0               ;Lautstärke um 1 erhöhen
18          cmpi    #64,d0              ;und wenn nicht gleich 64,
19          bne     loopl               ;dann weiter
20
21 wait:    andi.b  #64,$bfe001         ;Warten, bis Maustaste gedrückt
22          bne     wait
23
24          move.w  #%0000000000000011,$dff096   ;Audio-DMA stoppen
25          rts                                  ;Rückkehr
26
27 even
28 wave:    DC.B    0,90,127,90,0,-90,-127,-90   ;Wavedaten
```

```
1  ;*******************************
2  ;
3  ; 3. Sound - Demonstration
4  ; last update 10/03/88
5  ; von Frank Kremser und Jörg Koch
6  ; © Markt & Technik 1988
7  ;
8  ;*******************************
9  ;
10 ;Diese Demonstration spielt einen Ton, der von links nach rechts
11 ;'wandert'. Der Ton wird auf allen vier Kanälen gespielt, wobei zu
12 ;Beginn die Lautstärke der linken Kanäle auf höchster Stufe steht
13 ;und die der rechten Kanäle auf 0. Durch herunterstellen(herauf-)
14 ;der Lautstäke der linken(rechten) Seite, wird der 'Wandereffekt'
15 ;erzeugt.
16 ;
17 ;*******************************
18
19          lea.l   wave(pc),a0         ;a0 = Zeiger auf Wave
20          move.l  a0,$dff0a0          ;Wavezeiger Kanal 0
```

```
21              move.l  a0,$dff0b0          ;       Kanal 1
22              move.l  a0,$dff0c0          ;       Kanal 2
23              move.l  a0,$dff0d0          ;       Kanal 3 setzen
24              move.w  #4,d0               ;Wavelänge = 4 Words
25              move.w  d0,$dff0a4          ;Wavelänge Kanal 0
26              move.w  d0,$dff0b4          ;       Kanal 1
27              move.w  d0,$dff0c4          ;       Kanal 2
28              move.w  d0,$dff0d4          ;       Kanal 3 setzen
29              move.w  #240,$dff0a6        ;Tonhöhe Kanal 0
30              move.w  #240,$dff0b6        ;       Kanal 1
31              move.w  #240,$dff0c6        ;       Kanal 2
32              move.w  #240,$dff0d6        ;       Kanal 3 setzen
33              move.w  #64,$dff0a8         ;Lautstärke Kanal 0
34              move.w  #0,$dff0b8          ;       Kanal 1
35              move.w  #0,$dff0c8          ;       Kanal 2
36              move.w  #64,$dff0d8         ;       Kanal 3 setzen
37              move.w  #%1000000000001111,$dff096   ;Audio-DMA einschalten
38
39              move.w  #0,d0               ;Anfangslautstärke von linker
40              move.w  #64,d2              ;und rechter Seite
41  loopl:      move.w  #25000,d1           ;Wartezeit setzen und solange
42  waitl:      sub     #1,d1               ;herunterzählen, bis
43              bne     waitl               ;Wartezeit = 0
44              move.w  d2,$dff0a8          ;Lautstäke Kanal 0
45              move.w  d0,$dff0b8          ;       Kanal 1
46              move.w  d0,$dff0c8          ;       Kanal 2
47              move.w  d2,$dff0d8          ;       Kanal 3 setzen
48              sub     #1,d2               ;d2 um 1 erniedrigen
49              add     #1,d0               ;d0 um 1 erhöhen
50              cmpi    #64,d0              ;Falls rechte Kanäle noch
51              bne     loopl               ;nicht volle Lautst., weiter
52
53              move.w  #%0000000000001111,$dff096  ;Audio-DMA einschalten
54              rts                                 ;Rückkehr
55
56  even
57  wave:       DC.B    0,90,127,90,0,-90,-127,-90  ;Wavedaten


 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
```

```
12  ******************************
13
14            lea.l   wave(pc),a0        ;a0 = Zeiger auf die Wavedaten
15            move.l  a0,$dff0a0         ;Zeiger für Kanal 0
16            move.l  a0,$dff0b0         ;       Kanal 1
17            move.l  a0,$dff0c0         ;       Kanal 2
18            move.l  a0,$dff0d0         ;       Kanal 3 setzen
19            move.w  #4,d0              ;Wavelänge = 4
20            move.w  d0,$dff0a4         ;Wavelänge Kanal 0
21            move.w  d0,$dff0b4         ;       Kanal 1
22            move.w  d0,$dff0c4         ;       Kanal 2
23            move.w  d0,$dff0d4         ;       Kanal 3 setzen
24            move.w  #0,$dff0a6         ;Tonhöhe Kanal 0
25            move.w  #0,$dff0b6         ;       Kanal 1
26            move.w  #0,$dff0c6         ;       Kanal 2
27            move.w  #0,$dff0d6         ;       Kanal 3 setzen
28            move.w  #64,$dff0a8        ;Lautstärke Kanal 0
29            move.w  #64,$dff0b8        ;       Kanal 1
30            move.w  #64,$dff0c8        ;       Kanal 2
31            move.w  #64,$dff0d8        ;       Kanal 3 setzen
32            move.w  #%1000000000001111,$dff096   ;Audio-DMA ein
33
34  jump:     move.w  #128,d0            ;Variable Tonhöhe initialisieren
35  loopl:    move.w  #200,d1            ;Variable für Warteschleife
36  waitl:    sub     #1,d1              ;solange erniedrigen bis
37            bne     waitl              ;sie gleich 0 ist
38            move.w  d0,$dff0a6         ;neue Tonhöhe Kanal 0
39            move.w  d0,$dff0b6         ;       Kanal 1
40            move.w  d0,$dff0c6         ;       Kanal 2
41            move.w  d0,$dff0d6         ;       Kanal 3 setzen
42            add     #1,d0              ;und um 1 erhöhen
43            cmpi    #400,d0            ;falls Obergrenze nicht erreicht,
44            bne     loopl              ;dann weitermachen
45
46            andi.b  #64,$bfe001        ;Falls Maustaste gedrückt, dann
47            beq     ende               ;Programm beenden
48
49            move.w  #400,d0            ;Anfangstonhöhe initialisieren
50  loop2:    move.w  #200,d1            ;Warteschleife
51  wait2:    sub     #1,d1
52            bne     wait2
53            move.w  d0,$dff0a6         ;Neue Tonhöhe Kanal 0
54            move.w  d0,$dff0b6         ;       Kanal 1
55            move.w  d0,$dff0c6         ;       Kanal 2
56            move.w  d0,$dff0d6         ;       Kanal 3 setzen
57            sub     #1,d0              ;und um 1 vermindern
58            cmpi    #128,d0            ;Falls untere Grenze noch nicht
59            bne     loop2              ;erreicht, dann weiter
60            bra     jump               ;Erneut Tonhöhe erhöhen
61
```

```
62  ende:     move.w  #%0000000000001111,$dff096   ;Audio-DMA stoppen
63            rts                                  ;Rückkehr
64
65  even
66  wave:     DC.B    0,90,127,90,0,-90,-127,-90   ;Wavedaten


 1  ;*********************************
 2  ;
 3  ; 5. Sound - Demonstration
 4  ; last update 10/03/88
 5  ; Frank Kremser und Jörg Koch
 6  ; Markt & Technik 1988
 7  ;
 8  ;*********************************
 9  ;
10  ;Diese Demonstration spielt ein Musikstück, das nicht digitalisiert
11  ;ist.
12  ;Mit 'j on' kann es gestartet und mit 'j off' gestoppt werden. (SEKA)
13  ;Wir weisen ausdrücklich darauf hin, daß dieses Programm in seiner
14  ;Urfassung nicht von uns entwickelt worden ist!
15  ;
16  ;*********************************
17
18        irqvec        =$00000068
19        dmacon        =$dff096
20        timerlo       =$00
21        timerhi       =$30
22        talo          =$bfe401
23        tahi          =$bfe501
24        icr           =$bfed01
25        cra           =$bfee01
26        aud0lch       =$a0
27        aud0lcl       =$a2
28        aud0len       =$a4
29        aud0per       =$a6
30        aud0vol       =$a8
31        s_wavedco     =0
32        s_envelope    =32
33        s_sus         =48
34        s_end         =49
35        s_wavelfo     =50
36        s_spdlfo      =66+16
37        s_slctlfo     =68+16
38        s_typelfo     =68+16
39        s_phase       =69+16
40        s_arp         =70+16
41        s_spdport     =78+16
42        s_spdbend     =80+16
43        s_egfreq      =82+16
```

```
44        v_sactive     =0
45        v_envpoint    =4
46        v_lfowpoint   =8
47        v_phmark      =12
48        v_notetim     =14
49        v_lfotim      =16
50        v_arpoint     =18
51        v_actnote     =22
52        v_wntnote     =24
53        v_hardw       =26
54        v_trkstp      =30
55        v_crnot       =34
56        v_trkbck      =38
57        v_actfrq      =42
58        v_trnspse     =46
59        v_savnote     =50
60        v_add         =52
61        v_pauspnt     =54
62        v_actloud     =56 on:
63
64            move.l  #$1,d0              ;Schaltet den Sound ein
65            jmp     progstart
66  off:      move.l  #$0,d0              ;Schaltet den Sound aus
67            jmp     progstart
68
69  saveirqvec:
70            dc.l    0
71
72  soundl:   dc.b    -128,-120,-112,-104,-96,-88,-80,-72,-64,-56,-48
73            dc.b    -40,-32,-24,-16,-8,0,8,16,24,32,40,48,56,64,72
74            dc.b    80,88,96,104,112,120
75            dc.b    255,255,14,0,0,0,0,0,0,0,0,0,0,0,0 ;(Rate/Level)
76            dc.b    0,2
77
78            dc.b    0,2,4,6,8,6,4,2,0,-2,-4,-6,-8,-6,-4,-2
79            dc.b    0,2,4,6,8,6,4,2,0,-2,-4,-6,-8,-6,-4,-2
80
81            dc.b    0                   ;LFO Geschwindigkeit
82            dc.b    0
83            dc.b    1                   ;LFO ein
84
85            dc.b    0                   ;Phasentiefe = 0
86
87            dc.b    0,0,0,0,0,0,0,0
88
89            dc.w    0                   ;Speed Portamento
90
91            dc.w    0                   ;Bend-Rate
92
93            dc.w    0                   ;EG Frequenz aus
```

```
 94
 95 sound2:   dc.b    0,-16,-32,-48,-64,-80,-96,-112,-128,-112,-96,-80
 96           dc.b    -64,-48,-32,-16,0,16,32,48,64,80,96,112,127,112
 97           dc.b    96,80,64,48,32,16,150,150,0,0,0,0,0,0,0,0,0,0
 98           dc.b    0,0,0;              Envelope (Rate/Level)
 99           dc.b    0,1
100
101           dc.b    0,2,4,6,8,6,4,2,0,-2,-4,-6,-8,-6,-4,-2
102           dc.b    0,2,4,6,8,6,4,2,0,-2,-4,-6,-8,-6,-4,-2
103
104           dc.b    0                   ;LFO Geschwindigkeit
105           dc.b    0
106           dc.b    0                   ;LFO ein
107
108           dc.b    0                   ;Phasentiefe = 0
109
110           dc.b    0,0,0,0,0,0,0,0     ;Arpeggio
111
112           dc.w    0
113
114           dc.w    0                   ;Bend-Rate
115
116           dc.w    0                   ;EG Frequenz aus
117
118
119 sound3:   dc.b    128,128,128,128,128,128,128,128,128,128,128
120           dc.b    128,128,128,128,128,127,127,127,127,127,127
121           dc.b    127,127,127,127,127,127,127,127,127,127
122
123           dc.b    255,255,30,0,0,0,0,0,0,0,0,0,0,0,0  ;(Rate/Level)
124           dc.b    0,2
125
126           dc.b    0,2,4,6,8,6,4,2,0,-2,-4,-6,-8,-6,-4,-2
127           dc.b    0,2,4,6,8,6,4,2,0,-2,-4,-6,-8,-6,-4,-2
128
129           dc.b    0                   ;LFO Geschwindigkeit
130           dc.b    0
131           dc.b    0                   ;LFO ein
132
133           dc.b    0                   ;Phasentiefe = 0
134
135           dc.b    0,0,0,0,0,0,0,0
136
137           dc.w    0
138
139           dc.w    -30                 ;Bend-Rate
140
141           dc.w    0                   ;EG Frequenz aus
142
143
```

```
144 sound4:  dc.b   128,128,128,128,128,128,128,128,128,128,128,128
145          dc.b   128,128,128,128,127,127,127,127,127,127,127,127
146          dc.b   127,127,127,127,127,127,127,127
147
148          dc.b   40,175,10,100,6,0,0,0,0,0,0,0,0,0,0  ;(Rate/Level)
149          dc.b   2,3
150
151          dc.b   0,2,4,6,8,6,4,2,0,-2,-4,-6,-8,-6,-4,-2
152          dc.b   0,2,4,6,8,6,4,2,0,-2,-4,-6,-8,-6,-4,-2
153
154          dc.b   0                    ;LFO Geschwindigkeit
155          dc.b   0
156          dc.b   0                    ;LFO ein
157
158          dc.b   0                    ;Phasentiefe = 0
159
160          dc.b   0,0,0,0,0,0,0,0
161
162          dc.w   0
163
164          dc.w   0                    ;Bend-Rate
165
166          dc.w   0                    ;EG Frequenz aus
167
168
169 even
170 freqtab:  dc.w   6848,6464,6096,5760,5424,5120,4832,4560,4304
171          dc.w   4064,3840,3616,3424,3232,3048,2880,2712,2560
172          dc.w   2416,2280,2152,2032,1920,1808,1712,1616,1524
173          dc.w   1440,1356,1280,1208,1140,1076,1016,0960,0904
174          dc.w   0856,0808,0762,0720,0678,0640,0604,0570,0538
175          dc.w   0508,0480,0452,0428,0404,0381,0360,0339,0320
176          dc.w   0302,0285,0269,0254,0240,0226,0214,0202,0190
177          dc.w   0180,0170,0160,0151,0143,0135,0127
178
179 voice1:   dc.l   sound1              ;aktiver Sound
180          dc.l   0                    ;Zeiger auf Envelope
181          dc.l   0                    ;Zeiger auf LFO-Wave
182          dc.w   0                    ;Phaser
183          dc.w   0                    ;Noten-Timer
184          dc.w   0                    ;Timer von LFO-Speed
185          dc.l   0                    ;Zeiger auf Arpeggio
186          dc.w   0                    ;aktuelle Frequenz
187          dc.w   0                    ;Gewünschte Frequenz
188          dc.l   $dff0a0              ;Zeiger auf Hardwareregister
189          dc.l   0                    ;Derzeitige Spur
190          dc.l   0                    ;Derzeitige Note
191          dc.l   track1               ;Spur zurücknehmen
192          dc.l   0                    ;aktuelle Frequenz
193          dc.l   0                    ;Transpose
```

```
194             dc.w    0                       ;gesicherte Note
195             dc.w    0                       ;zu addierender Wert
196             dc.w    0                       ;Pause
197             dc.w    0                       ;aktuelle Lautstärke
198
199 voice2:     dc.l    soundl
200             dc.l    0
201             dc.l    0
202             dc.w    0
203             dc.w    0
204             dc.w    0
205             dc.l    0
206             dc.w    0
207             dc.w    0
208             dc.l    $dff0b0
209             dc.l    0
210             dc.l    0
211             dc.l    track2
212             dc.l    0
213             dc.l    0
214             dc.w    0
215             dc.w    0
216             dc.w    0
217             dc.w    0
218
219 voice3:     dc.l    soundl
220             dc.l    0
221             dc.l    0
222             dc.w    0
223             dc.w    0
224             dc.w    0
225             dc.l    0
226             dc.w    0
227             dc.w    0
228             dc.l    $dff0c0
229             dc.l    0
230             dc.l    0
231             dc.l    track3
232             dc.l    0
233             dc.l    0
234             dc.w    0
235             dc.w    0
236             dc.w    0
237             dc.w    0
238
239 voice4:     dc.l    soundl
240             dc.l    0
241             dc.l    0
242             dc.w    0
243             dc.w    0
```

```
244          dc.w    0
245          dc.l    0
246          dc.w    0
247          dc.w    0
248          dc.l    $dff0d0
249          dc.l    0
250          dc.l    0
251          dc.l    track4
252          dc.l    0
253          dc.l    0
254          dc.w    0
255          dc.w    0
256          dc.w    0
257          dc.w    0
258
259 trackl:   dc.l    scorel,l,scorel,-3,scorel,-8,score2,-1
260          dc.l    0,0
261 track2:   dc.l    score3,l
262          dc.l    0,0
263 track3:   dc.l    score4,0
264          dc.l    0,0
265 track4:   dc.l    score8,l,score5,l,score5,l,score7,l
266          dc.l    score7,l,score6,l,score6,l
267          dc.l    0,0
268
269 progstart:
270          movem.l d0-d7/a0-a6,-(a7) ;Register sichern
271          tst.l   d0                ;Testen, on: oder off:
272          beq     soff              ;Wenn off:, dann ausschalten
273          jsr     switchon          ;Ansonsten einschalten
274          bra     cont
275 soff:    jsr     switchoff
276 cont:    movem.l (a7)+,d0-d7/a0-a6 ;Register zurückgeben
277          rts                       ;Rückkehr
278
279 switchoff:
280          cmpi.l  #newirq,irqvec    ;Interrupt prüfen
281          bne     notoff            ;Wenn kein neuer, dann zurück
282          move.b  #$0l,icr
283          move.l  saveirqvec,irqvec ;Ansonsten Irq zurücksetzen
284          move.w  #$000f,dmacon     ;DMA abschalten
285          jsr     setback
286 notoff:  rts
287
288 switchon:
289          cmpi.l  #newirq,irqvec    ;Prüfen, ob neuer Interrupt
290          beq     noton             ;Wenn nein, dann zurück
291          move.b  #$8l,icr          ;Ansonsten Register setzen
292          move.b  #$2f,cra
293          move.b  #$8l,cra
```

```
294             move.b  #timerlo,talo
295             move.b  #timerhi,tahi
296             move.l  irqvec,saveirqvec ;Interruptvektor sichern
297             move.l  irqvec,statement+2
298             jsr     setback           ;Audioregister zurücksetzen
299             jsr     setup             ;Audioregister setzen
300             move.w  #$800f,dmacon     ;DMA einschalten
301             move.l  #newirq,irqvec    ;neuen Interruptvektor setzen
302 noton:      rts
303
304 setback:    lea     $dff000,a0        ;Kanal 0
305             jsr     reset             ;zurücksetzen
306             lea     $dff010,a0        ;Kanal 1
307             jsr     reset             ;zurücksetzen
308             lea     $dff020,a0        ;Kanal 2
309             jsr     reset             ;zurücksetzen
310             lea     $dff030,a0        ;Kanal 3
311             jsr     reset             ;zurücksetzen
312             rts
313 reset:      clr.l   aud0lch(a0)       ;Wavedatapointer löschen
314             clr.w   aud0len(a0)       ;Wavedatalenght löschen
315             clr.w   aud0per(a0)       ;Period löschen
316             clr.w   aud0vol(a0)       ;Volume löschen
317             rts
318 setup:      move.l  #track1,v_trkstp+voice1 ;Datas für Musik aufbereiten
319             move.l  #track2,v trkstp+voice2
320             move.l  #track3,v trkstp+voice3
321             move.l  #track4,v trkstp+voice4
322             clr.l   v_lfowpoint+voice1
323             clr.l   v_lfowpoint+voice2
324             clr.l   v_lfowpoint+voice3
325             clr.l   v_lfowpoint+voice4
326             clr.l   v_notetim+voice1
327             clr.l   v_notetim+voice2
328             clr.l   v_notetim+voice3
329             clr.l   v_notetim+voice4
330             move.l  track1,v_crnot+voice1
331             move.l  track2,v_crnot+voice2
332             move.l  track3,v_crnot+voice3
333             move.l  track4,v_crnot+voice4
334             move.l  track1+4,v_trnspse+voice1
335             move.l  track2+4,v_trnspse+voice2
336             move.l  track3+4,v_trnspse+voice3
337             move.l  track4+4,v_trnspse+voice4
338             clr.l   v_lfotim+voice1
339             clr.l   v_lfotim+voice2
340             clr.l   v_lfotim+voice3
341             clr.l   v_lfotim+voice4
342
343             move.l  #s_wavedco+soundl,aud0lch+$dff000 ;Wave Kanal 0
```

```
244          move.l  #s_wavedco+soundl,aud0lch+$dff010;  Kanal 1
245          move.l  #s_wavedco+soundl,aud0lch+$dff020;  Kanal 2
246          move.l  #s_wavedco+soundl,aud0lch+$dff030  ;Kanal 3 setzen
247          move.w  #$10,aud0len+$dff000              ;Wavelenght Kan. 0
248          move.w  #$10,aud0len+$dff010              ;  Kanal 1
249          move.w  #$10,aud0len+$dff020              ;  Kanal 2
250          move.w  #$10,aud0len+$dff030              ;  Kanal 3 setzen
251          rts
252
253 ;Neue Interruptroutine
254 newirq:   movem.l d0-d7/a0-a6,-(a7) ;Register retten
255          cmp.b   #timerhi-1,$bfe501
256          bne     nottim
257          jsr     playsound
258          eori.b  #$ff,sound4+16
259 nottim:  movem.l (a7)+,d0-d7/a0-a6 ;Register zurückgeben
260 statement:
261          jmp     $ffffffff          ;Interrupt beenden
262                                     ;Wird später noch gesetzt
263
264 playsound:
265          lea     voice1,a0
266          jsr     playvoice
267          lea     voice2,a0
268          jsr     playvoice
269          lea     voice3,a0
270          jsr     playvoice
271          lea     voice4,a0
272          jsr     playvoice
273          rts
274
275 playvoice:
276          subq.w  #1,v_notetim(a0)
277          bpl     notyet1
278          clr.w   v_add(a0)
279          clr.w   v_pauspnt(a0)
280 musll:   move.l  v_crnot(a0),a1
281          move.l  v_trnspse(a0),d3
282          clr.l   d4
283          move.w  (a1),d4
284          cmp.w   #128,d4
285          bne     testarp
286          clr.l   v_lfowpoint(a0)
287          move.l  2(a1),v_sactive(a0)
288          addi.l  #6,v_crnot(a0)
289          move.l  2(a1),d5
290          move.l  v_hardw(a0),a3
291          move.l  d5,(a3)
292          bra     musll
293
```

```
294 testarp:  cmp.w   #129,d4
295           bne     chpause
296           move.l  v_sactive(a0),a3
297           move.l  2(a1),s_arp(a3)
298           move.l  6(a1),s_arp+4(a3)
299           addi.l  #10,v_crnot(a0)
300           bra     musll
301 chpause:   cmp.w   #130,d4                                    ;pause?
302           bne     chweiter
303           move.w  #1,v_pauspnt(a0)
304           bra     gutl
305 chweiter:
306           add.w   d4,d3
307           cmpi.l  #$00,d4
308           bne     gutl
309           addi.l  #8,v_trkstp(a0)
310           move.l  v_trkstp(a0),a2
311           move.l  4(a2),v_trnspse(a0)
312           move.l  (a2),v_crnot(a0)
313           bne     musll
314           move.l  v_trkbck(a0),v_trkstp(a0)
315           move.l  v_trkstp(a0),a2
316           move.l  4(a2),v_trnspse(a0)
317           move.l  (a2),v_crnot(a0)
318           bra     musll
319
320 gutl:     tst.w   v_pauspnt(a0)
321           bne     notset
322           clr.l   v_envpoint(a0)
323           clr.w   v_actloud(a0)
324           move.w  d3,v_savnote(a0)
325 notset:   move.w  2(a1),v_notetim(a0)
326           subq.w  #1,v_notetim(a0)
327           adda.l  #4,a1
328           move.l  a1,v_crnot(a0)
329 notyetl:  move.l  v_hardw(a0),a2
330           move.l  v_sactive(a0),a3
331           move.w  v_wntnote(a0),d0
332           subq.w  #1,d0
333           mulu    #2,d0
334           lea     freqtab,a4
335           move.w  (a4,d0.w),d1
336           move.w  s_spdport(a3),d0
337           beq     noport
338           cmp.w   v_actfrq(a0),d1
339           blo     portdown
340           addi.w  d0,v_actfrq(a0)
341           cmp.w   v_actfrq(a0),d1
342           bhi     nochklei
343           move.w  d1,v_actfrq(a0)
```

```
344 nochklei:
345           bra     portaend
346 portdown:
347           subi.w  d0,v_actfrq(a0)
348           cmp.w   v_actfrq(a0),d1
349           blo     nochgroe
350           move.w  d1,v_actfrq(a0)
351 nochgroe:
352           bra     portaend
353 noport:   add.w   v_add(a0),d1
354           move.w  d1,v_actfrq(a0)
355 portaend:
356           move.l  v_sactive(a0),a3
357           adda.l  #s_arp,a3
358           move.l  v_arpoint(a0),d1
359           clr.l   d2
360           move.b  (a3,d1.l),d2
361           bpl     positiv
362           neg.b   d2
363           clr.l   d3
364           move.w  v_savnote(a0),d3
365           sub.w   d2,d3
366           move.w  d3,d2
367           bra     negativ
368 positiv:  add.w   v_savnote(a0),d2
369 negativ:  move.w  d2,v_wntnote(a0)
370           addq.l  #1,v_arpoint(a0)
371           cmp.l   #8,v_arpoint(a0)
372           bne     notnull
373           clr.l   v_arpoint(a0)
374 notnull:  move.l  v_sactive(a0),a3
375           tst.w   v_phmark(a0)
376           beq     ffff
377           clr.w   v_phmark(a0)
378           clr.l   d2
379           move.b  s_phase(a3),d2
380           move.w  v_actfrq(a0),d1
381           add.w   d2,d1
382           move.w  d1,v_actfrq(a0)
383           bra     wasffff
384 ffff:     move.w  #$ffff,v_phmark(a0)
385           clr.l   d2
386           move.b  s_phase(a3),d2
387           move.w  v_actfrq(a0),d1
388           sub.w   d2,d1
389           move.w  d1,v actfrq(a0)
390 wasffff:  clr.l   d2
391           move.w  s_spdbend(a3),d2
392           sub.w   d2,v_add(a0)
393           clr.l   d0
```

```
394             move.w  v_actfrq(a0),d0
395             move.w  s_egfreq(a3),dl
396             beq     noteg
397             bmi     dazu
398             sub.w   v_actloud(a0),d0
399             bra     noteg
400 dazu:       add.w   v_actloud(a0),d0
401 noteg:      move.l  v_lfowpoint(a0),dl
402             subq.b  #l,v_lfotim(a0)
403             bpl     notyet2
404             addq.l  #l,dl
405             cmp.l   #$20,dl
406             bne     nichtnull
407             clr.l   dl
408 nichtnull:
409             move.l  dl,v_lfowpoint(a0)
410             move.b  s_spdlfo(a3),v_lfotim(a0)
411 notyet2:    lea     s_wavelfo(a3),a4
412             clr.l   d2
413             move.b  (a4,dl.l),d2
414             ext.w   d2
415             tst.b   s_slctlfo(a3)
416             beq     nonelfo
417             add.w   d2,d0
418 nonelfo:    move.w  d0,$06(a2)
419             clr.l   d2
420             move.l  v_sactive(a0),a2
421             clr.l   d0
422             clr.l   dl
423             move.b  s_sus(a2),d0
424             move.b  s_end(a2),dl
425             cmp.l   v_envpoint(a0),dl
426             beq     envelopend
427             cmp.l   #$00,d0
428             beq     notsustep
429             cmp.l   v_envpoint(a0),d0
430             bne     notsustep
431             cmpi.w  #$00,v_pauspnt(a0)
432             beq     envelopend
433 notsustep:
434             move.l  v_envpoint(a0),d2
435             mulu    #2,d2
436             lea     s_envelope(a2),a3
437             clr.l   d3
438             clr.l   d4
439             move.b  (a3,d2.w),d3
440             move.b  l(a3,d2.w),d4
441             cmp.w   v_actloud(a0),d4
442             bhi     loudup
443             sub.w   d3,v_actloud(a0)
```

```
144           cmp.w   v_actloud(a0),d4
145           ble     nichtunt
146           move.w  d4,v actloud(a0)
147           addq.l  #$1,v_envpoint(a0)
148 nichtunt:
149           bra     envelopend
150 loudup:
151           add.w   d3,v_actloud(a0)
152           cmp.w   v_actloud(a0),d4
153           bhi     envelopend
154           move.w  d4,v actloud(a0)
155           addq.l  #$1,v_envpoint(a0)
156 envelopend:
157           clr.l   dl
158           move.w  v_actloud(a0),dl
159           divu    #4,dl
160           move.l  v_hardw(a0),al
161           move.w  dl,$08(al)
162           rts
163
164 scorel:   dc.w    128
165           dc.l    soundl
166           dc.w    129
167           dc.l    0,0
168           dc.w    25,12,25,24,25,6,37,6,25,12,25,36
169           dc.w    0,0
170
171 score2:   dc.w    128
172           dc.l    soundl
173           dc.w    129
174           dc.l    0,0
175           dc.w    25,12,25,24,25,6,23,6,25,12,25,36
176           dc.w    0,0
177
178 score3:   dc.w    128
179           dc.l    sound2
180           dc.w    129
181           dc.l    $00030700,$00030700
182           dc.w    61,96
183           dc.w    129
184           dc.l    $00030800,$00030800
185           dc.w    61,96
186           dc.w    129
187           dc.l    $fe0307fe,$fe0307fe
188           dc.w    61,96
189           dc.w    129
190           dc.l    $fe0205fe,$fe0205fe
191           dc.w    61,96
192           dc.w    0,0
193
```

```
494 score4:  dc.w   128
495          dc.l   sound3
496          dc.w   129
497          dc.l   0,0
498          dc.w   130,24,45,12,130,36,45,12,45,6,45,6
499          dc.w   130,24,45,12,130,36,45,8,44,8,43,8
500          dc.w   0,0
501
502 score5:  dc.w   128
503          dc.l   sound4
504          dc.w   129
505          dc.l   0,0
506          dc.w   49,48,52,12,51,12,49,12,47,12,45,48
507          dc.w   52,12,51,12,49,12,47,12,52,48
508          dc.w   52,12,51,12,49,12,47,12
509          dc.w   54,24,51,24,49,24,47,24
510          dc.w   0,0
511
512 score6:  dc.w   128
513          dc.l   sound4
514          dc.w   129
515          dc.l   0,0
516          dc.w   49,48,56,12,56,12,56,12,54,24,56,108
517          dc.w   52,12,52,24,52,24,52,12
518          dc.w   54,24,51,24,49,24,47,24
519          dc.w   0,0
520
521 score7:  dc.w   128
522          dc.l   sound4
523          dc.w   129
524          dc.l   0,0
525          dc.w   49,72,56,24,54,12,52,12,49,96
526          dc.w   52,12,52,24,52,24,52,12
527          dc.w   54,24,51,24,49,24,47,24
528          dc.w   0,0
529
530 score8:  dc.w   130,384
531          dc.w   0,0
```

```
;******************************
;
; 6. Sound - Demonstration
; last update 10/03/88
; von Frank Kremser und Jörg Koch
; © Markt & Technik 1988
;
;******************************
;
;Diese Demonstration lädt ein digitalisiertes Musikstück namens
;'Sound' von Diskette(aktuelles Laufwerk) und spielt es dann
;solange, bis die Maustaste gedrückt wird.
;
;******************************

      MaxSndLen  =131070          ;Maximale Länge des Sounds
      SndBuffer  =$50000          ;Startposition des Puffers

      ExecBase      =$4
      OpenLibrary   =-408
      CloseLibrary  =-414
      Open          =-30
      Close         =-36
      Read          =-42
      OPEN_OLD      =1005

          move.l  ExecBase,a6
          lea     DosName,a1        ;DosLibrary öffnen
          jsr     OpenLibrary(a6)

          move.l  d0,a6
          move.l  #File,d1
          move.l  #OPEN_OLD,d2
          jsr     Open(a6)          ;Datei 'Sound' öffnen
          move.l  d0,d7
          move.l  d0,d1
          move.l  #SndBuffer,d2
          move.l  #MaxSndLen,d3
          jsr     Read(a6)          ;Datei lesen(In d0 befindet sich
          lsr.l   #1,d0             ;anschließend die Anzahl der echt
          move.w  d0,SndLen         ;gelesenen Bytes (/2=Words))
          move.l  d7,d1
          jsr     Close(a6)         ;Datei schließen
          move.l  a6,a1
          move.l  ExecBase,a6       ;DosLibrary schließen
          jsr     CloseLibrary(a6)
play:     move.l  #$50000,$dff0a0   ;Wavedata Kanal 0
          move.l  #$50000,$dff0b0   ;      Kanal 1
          move.l  #$50000,$dff0c0   ;      Kanal 2
          move.l  #$50000,$dff0d0   ;      Kanal 3 setzen
```

```
51              move.w  SndLen,$dff0a4      ;Wavelänge Kanal 0
52              move.w  SndLen,$dff0b4      ;      Kanal 1
53              move.w  SndLen,$dff0c4      ;      Kanal 2
54              move.w  SndLen,$dff0d4      ;      Kanal 3 setzen
55              move.w  #340,$dff0a6        ;Period Kanal 0
56              move.w  #340,$dff0b6        ;      Kanal 1
57              move.w  #340,$dff0c6        ;      Kanal 2
58              move.w  #340,$dff0d6        ;      Kanal 3 setzen
59              move.w  #64,$dff0a8         ;Lautstärke Kanal 0
60              move.w  #64,$dff0c8         ;      Kanal 1
61              move.w  #64,$dff0b8         ;      Kanal 2
62              move.w  #64,$dff0d8         ;      Kanal 3 setzen
63              move.w  #%1000000000001111,$dff096        ;Audio DMA ein
64
65  wait:       btst    #6,$bfe001          ;Warten bis linke Maustaste gedrückt
66              bne     wait
67
68              move.w  #64,d0              ;volle Lautstärke
69  loop2:      move.w  #5000,d1            ;Warteschleife
70  loop1:      sub     #1,d1
71              bne     loop1
72              move.w  d0,$dff0a8          ;Lautstärke Kanal 0
73              move.w  d0,$dff0b8          ;      Kanal 1
74              move.w  d0,$dff0c8          ;      Kanal 2
75              move.w  d0,$dff0d8          ;      Kanal 3 setzen
76              sub     #1,d0               ;Lautstärke runterregeln
77              bne     loop2               ;Wenn nicht = 0, dann weiter
78
79              move.w  #%0000000000001111,$dff096        ;Audio DMA aus
80              rts                                       ;Rückkehr
81
82  File:       dc.b    'Sound',0
83  DosName:    dc.b    'dos.library',0
84  even
85  SndLen:     dc.w    0
```

Die oben beschriebene Methode, Daten über die Audiohardware auszugeben, ist die am häufigsten verwendete Methode. Eine weitere stellt die Amplituden- (Lautstärke-) und/oder die Frequenzmodulation dar. Über das ADKCON-Register kann angegeben werden, daß die Daten eines Audiokanals zur Amplituden- /Frequenzmodulation eines anderen Kanals verwendet werden. Ist ein Kanal so geschaltet, daß seine Audiodaten zur Modulation verwendet werden, so kann über ihn kein Sound mehr ausgegeben werden, da die gelesenen Daten ja nicht mehr als Sounddaten zur Verfügung stehen. Ein Kanal kann immer nur so geschaltet werden, daß die gelesenen Daten für diesen Kanal in das Lautstärke-, bzw. Period-Register des nächsthöheren Registers geschrieben werden. Wurde Kanal 2 zur *Periodmodulation* von Kanal 3 geschaltet, so werden die gelesenen Audiodaten in das Period-Register von Kanal 3 geschrieben, das Gleiche gilt für

lie Lautstärkemodulation. Wurde aber Kanal 2 gleichzeitig zur Period- und zur *Lautstärkemodulation* von Kanal 3 geschaltet, so werden die Daten immer abwechselnd als Lautstärkedaten und als Perioddaten aufgefaßt, wobei zuerst die Lautstärke gesetzt wird. Hier die Möglichkeiten zur Modulation durch das ADKCON-Register:

Bit	Name	Funktion
15	SET/CLR	Ist dieses Bit gesetzt, werden alle Modulationsmodi, deren korrespondierendes Bit gesetzt ist, »enabled«. Ist dieses Bit gelöscht, werden alle Modi, deren korrespondierendes Bit gesetzt ist, »disabled«.
7	ATPER3	Schaltet die Audio-Ausgabe über Kanal 3 ab.
6	ATPER2	Verwendet die Daten von Kanal 2, um das Period-Register von Kanal 3 zu modulieren.
5	ATPER1	Verwendet die Daten von Kanal 1, um das Period-Register von Kanal 2 zu modulieren.
4	ATPER0	Verwendet die Daten von Kanal 0, um das Period-Register von Kanal 1 zu modulieren.
3	ATVOL3	Schaltet die Audio-Ausgabe über Kanal 3 ab.
2	ATVOL2	Verwendet die Daten von Kanal 2, um das Volume-Register von Kanal 3 zu modulieren.
1	ATVOL1	Verwendet die Daten von Kanal 1, um das Volume-Register von Kanal 2 zu modulieren.
0	ATVOL0	Verwendet die Daten von Kanal 0, um das Volume-Register von Kanal 1 zu modulieren.

4.3.3: Die Interrupt-Kontroll-Logik

Paula kontrolliert die Interrupts des Amiga-Systems. Alle Interrupts, die von Peripheriebausteinen erzeugt werden, werden von Paula in einen der 6 Interrupt-Levels des MC68000 übertragen. Der MC68000 besitzt zusätzlich noch einen Interrupt des Levels 7, NonMaskAble-Interrupt genannt, der nicht von Paula unterstützt wird. Er wird auch von keinem Bauteil des derzeitigen Systems erzeugt. Falls eine externe Erweiterung diesen Interrupt-Level benötigt, kann er mittels der MC68000-Interruptleitungen IPL0 bis 2, die an den Erweiterungports anliegen, direkt erzeugt werden. Auch die Interruptleitungen INT2 und INT6 liegen dort an, wodurch externe Baugruppen auch Interrupts der Level 2 und 6 über Paula erzeugen können.

Wie schon erwähnt, erzeugt Paula Interrupts der Level 1 bis 6. Um das System vor bestimmten Interrupts zu »schützen«, können diese über einige Chip-Register kontrolliert, bzw. abgeschaltet werden.

Diese Register sind INTENA, INTENAR, INTREQ und INTREQR. INTENAR und INTREQR sind Leseregister, aus denen gelesen werden kann, welche Interrupts möglich sind, bzw. welche Interruptanfragen zur Zeit bestehen.

Registerbeschreibung zu INTENA: Mit INTENA können einzelne Interrupts zugelassen, bzw. abgeschaltet werden (Aus INTENAR kann der Zustand der Interrupts ausgelesen werden, die Bit-Funktionen sind mit INTENA identisch, Bit 15 hat dann keine Funktion):

Bit	Name	Lev	Funktion
15	SET/CLR		Ist dieses Bit gesetzt, werden alle Interrupts, deren korrespondierendes Bit gesetzt ist, »enabled«. Ist dieses Bit gelöscht, werden alle Interrupts, deren korrespondierendes Bit gesetzt ist, »disabled«.
14	INTEN		Setzt alle Interrupts »enable«/»disable«.
13	EXTER	6	Setzt Interrupt Level 6, also externe Interruptleitung.
12	DSKSYN	5	Dieser Interrupt zeigt an, daß beim Lesen von Diskette die Syncbytes gefunden wurden, die im DSKSYNC-Register festgelegt wurden.
11	RBF	5	Durch diesen Interrupt wird signalisiert, daß der Empfangspuffer für die serielle Schnittstelle voll ist.
10–7	AUD3–0	4	Dieser Interrupt wird immer dann erzeugt, wenn die Audio-Hardware einen Speicherblock für Kanal 3 bis 0 fertig abgespielt hat und neu beginnt.
6	BLIT	3	Dieser Interrupt signalisiert, daß der Blitter eine Operation abgeschlossen hat und für neue Aufgaben bereitsteht.
5	VERTB	3	SetztVERTB-Interrupt. Dieser Interrupt wird immer dann erzeugt, wenn der Rasterstrahl wieder neu mit dem Bildaufbau beginnt.
4	COPER	3	Dieser Interrupt wird benötigt, damit der Copper den MC68000 unterbrechen kann, um Daten in die Chip-Register zu schreiben.
3	PORTS	2	Setzt Interrupt Level 2, also externe Interruptleitung.

Bit	Name	Lev	Funktion
2	SOFT	1	Dieser Interrupt signalisiert einen Software-Interrupt.
1	DSKBLK	1	Dieser Interrupt zeigt an, daß ein Diskblock komplett eingelesen wurde.
0	TBE	1	Durch diesen Interrupt wird angezeigt, daß der Sende-Datenpuffer für die serielle Schnittstelle leer ist.

Registerbeschreibung zu INTREQ und INTREQR: Mittels INTREQ können Interrupts erzeugt werden und aus INTREQR kann ausgelesen werden, welche Interrupts zur Zeit erzeugt sind. Da die Bits aus INTREQ mit denen aus INTENA identisch sind, gehen wir hier nicht mehr genauer darauf ein. Die Funktionen der Bits unterscheiden sich nur dahingehend, daß jeder Interrupt, dessen korrespondierendes Bit gesetzt ist, auch erzeugt wird.

```
 1  ;*******************************
 2  ;
 3  ; Interrupt-Demonstration
 4  ; last update 10/03/88
 5  ; von Frank Kremser und Jörg Koch
 6  ; © Markt & Technik 1988
 7  ;
 8  ;*******************************
 9  ;
10  ;Diese Demonstration ist zwar lauffähig, aber nicht sehr effektvoll.
11  ;Es wird ein Interruptvektor gesetzt, der eine Unterroutine per
12  ;Interrupt aufruft (etwa alle 60stel Sekunde), während das Haupt-
13  ;programmm fortfährt.
14  ;
15  ;*******************************
16
17            move.w  #$4000,$dff09a     ;Interrupts abschalten
18            move.l  $6c,OldVector      ;Alten Interruptvektor sichern
19            move.l  #IRQRout,$6c       ;Neuen Vektor setzen
20            move.w  #$c000,$dff09a     ;Interrupts einschalten
21
22  ;*******************************
23  ;An dieser Stelle kann Ihr Programm stehen, das wie ein normales
24  ;Programm ausgeführt wird.
25  ;*******************************
26
27  wait:     btst    #6,$bfe001         ;Warten, bis Maustaste
28            bne     wait               ;gedrückt
29
30            move.w  #$4000,$dff09a     ;Interrupts abschalten
```

```
31          move.l  OldVector,$6c       ;Alten Interruptvektor setzen
32          move.w  #$c000,$dff09a      ;Interrupts zulassen
33          rts                         ;Rückkehr
34
35  IRQRout: movem.l d0-d2/a0-a1,-(a7)  ;Register sichern
36          move    SR,-(sp)            ;Statusregister sichern
37
38  ;*******************************
39  ;An dieser Stelle kann nun Ihr Programm stehen, das bei jedem
40  ;Interrupt ausgeführt werden soll.
41  ;*******************************
42
43  EndIRQ:  move    (sp)+,SR           ;Statusregister zurück
44          movem.l (sp)+,d0-d2/a0-a1   ;Register zurückspeichern
45          dc.w    $4ef9               ;Interrupt beenden
46
47  OldVector:
48          dc.l    $0000
```

4.3.4: Der Game-Port

In diesem Kapitel wird erläutert, wie ein *Joystick*, der in Port 0 oder 1 eingesteckt ist, abgefragt werden kann. Auf die Abfrage der Maus wird in einem späteren Kapitel eingegangen.

Das Abfragen der Joystick-Stellung ist äußerst einfach. Dazu muß nur das Register JOY0DAT für Port 0 oder JOY1DAT für Port 1 ausgelesen werden. Ist Bit 3 dieses Registerwertes gesetzt, so ist die linke Taste gedrückt, bei gesetztem Bit 4 die rechte. Ist Bit 1 gesetzt, so ist der Joystick nach rechts gedrückt, bei gesetztem Bit 9 nach links. Um zu überprüfen, ob der Joystick nach oben oder unten gedrückt ist, muß etwas mehr Aufwand getrieben werden, denn dazu müssen zwei Bits logisch verknüpft werden und zwar mit einer XOR-Verknüpfung. Ist Bit 1 XOR Bit 0 wahr, so ist der Joystick nach hinten gedrückt und bei 9 XOR 8 nach vorne.

```
 1  /*******************************
 2  2. Joyport-Demonstration
 3  last update 16/02/88
 4  von Frank Kremser und Jörg Koch
 5  © Markt & Technik 1988
 6  *******************************
 7
 8  Diese Demonstration fragt den Joystick in Port 1 ab.
 9  Mit diesem kann ein Sprite über den Bildschirm bewegt werden.
10  Betätigung des Feuerknopfes beendet das Programm.
11  *******************************/
12  #include <exec/types.h>                    /* Include-Files laden */
13  #include <exec/tasks.h>
```

```
13 #include <exec/libraries.h>
14 #include <exec/memory.h>
15 #include <exec/devices.h>
16 #include <devices/keymap.h>
17 #include <devices/gameport.h>
18 #include <devices/inputevent.h>
19 #include <graphics/copper.h>
20 #include <graphics/display.h>
21 #include <graphics/gfxbase.h>
22 #include <graphics/text.h>
23 #include <graphics/view.h>
24 #include <graphics/gels.h>
25 #include <graphics/regions.h>
26 #include <graphics/sprite.h>
27 #include <hardware/blit.h>
28 #include <intuition/intuition.h>
29 #include <intuition/intuitionbase.h>
30
31 struct GamePortTrigger gpt =
32  {
33   GPTF_UPKEYS|GPTF_DOWNKEYS,
34   0,
35   1,
36   1
37  };
38
39 struct InputEvent        joyreport;
40 struct IOStdReq          *gameio;       /* Device-Request-Block */
41 struct MsgPort           *gameport;     /* Messageport */
42
43 struct GfxBase           *GfxBase;      /* Library-Pointer */
44 struct IntuitionBase     *IntuitionBase;
45
46 struct Screen            *screen;       /* Screen-Structure-Zeiger */
47
48 USHORT Data1[] =                        /* Sprite-Image */
49  {
50   0, 0,
51
52   0x0FC0, 0x0FC0,
53   0x3FF0, 0x3030,
54   0x7FF8, 0x4008,
55   0x7FF8, 0x4008,
56   0xF33C, 0x8CC4,
57   0xFFFC, 0x8004,
58   0xFFFC, 0x8004,
59   0xFCFC, 0x8304,
60   0xFFFC, 0x8004,
61   0xFFFC, 0x9024,
62   0x7FF8, 0x4848,
```

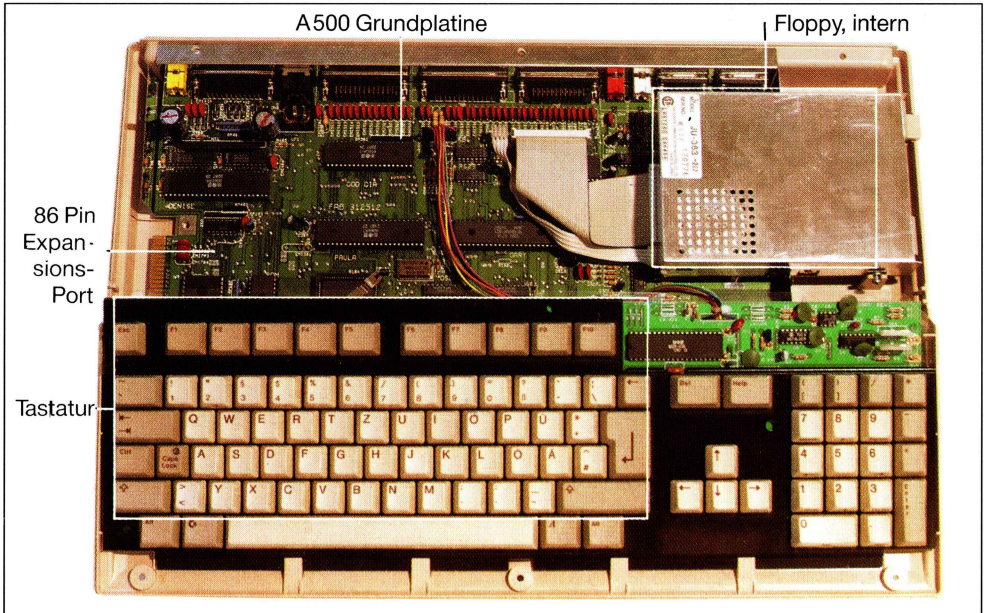


Bild 1: Amiga 500, der 64er-Nachfolger.

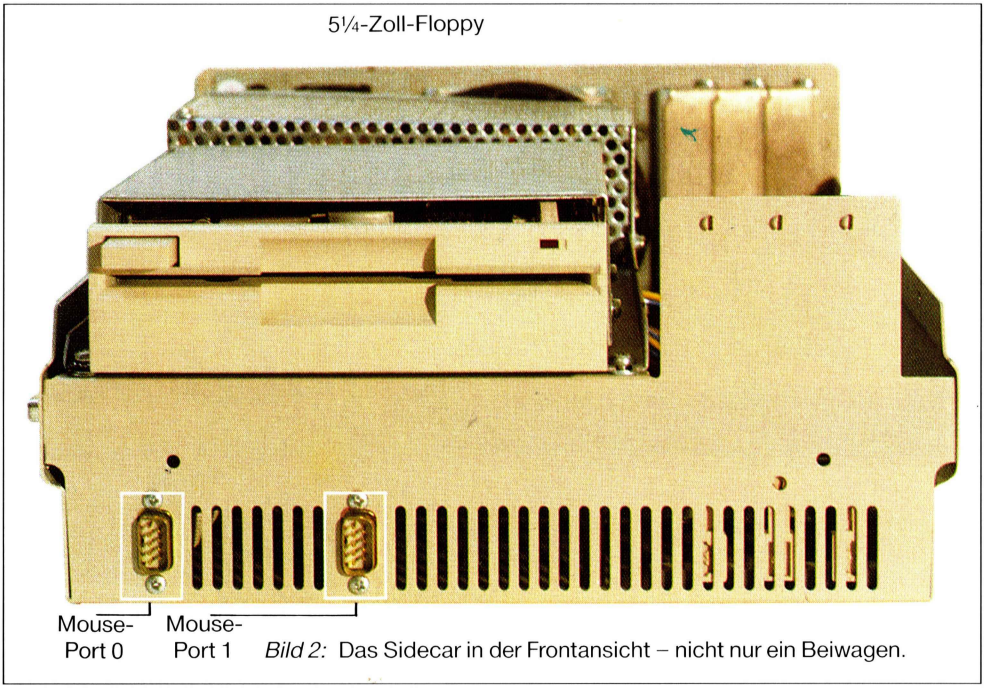


Bild 2: Das Sidecar in der Frontansicht – nicht nur ein Beiwagen.

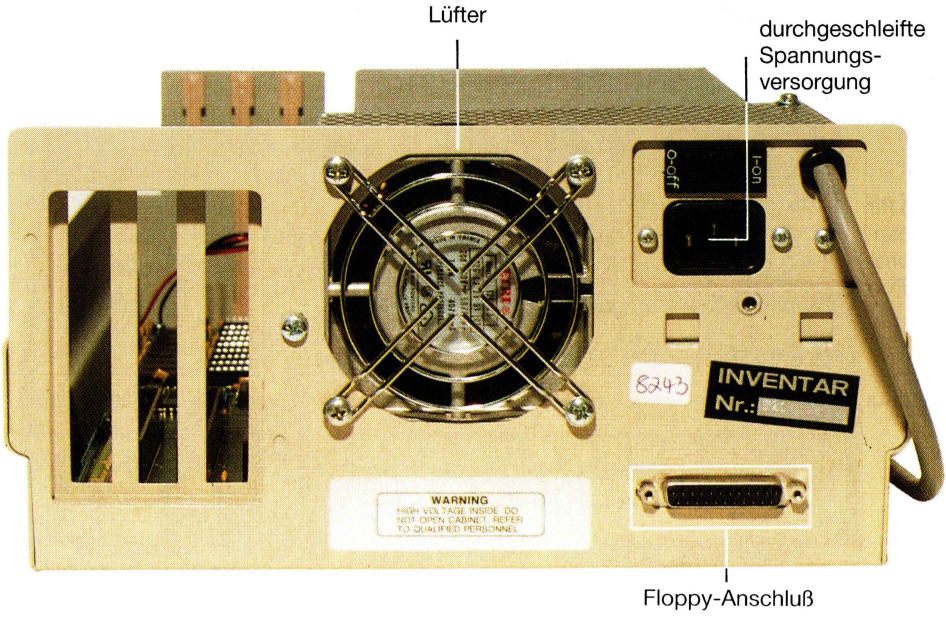


Bild 3: Das Sidecar, von hinten betrachtet.

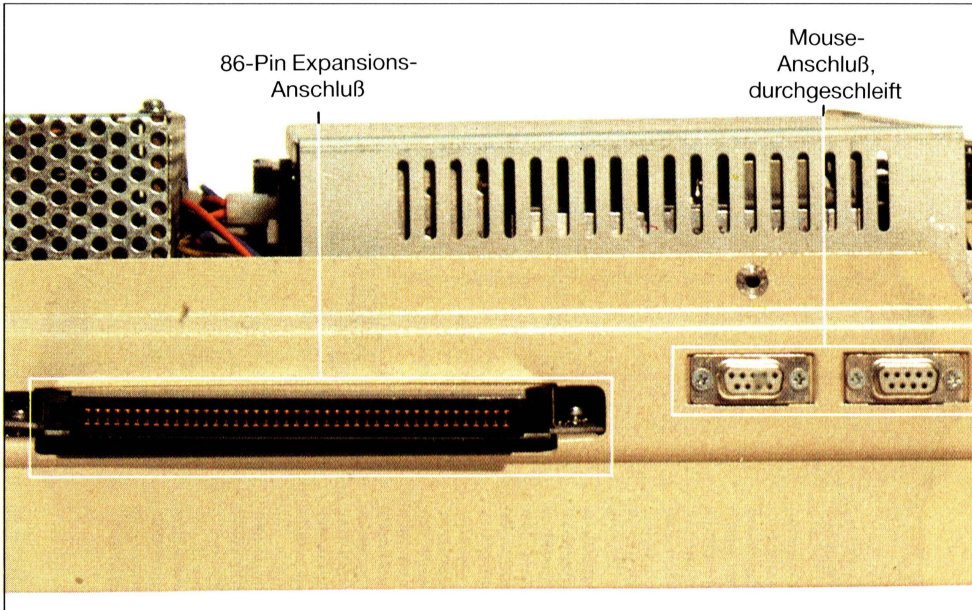


Bild 4: Der Amiga-Expansions-Port-Stecker des Sidecars.

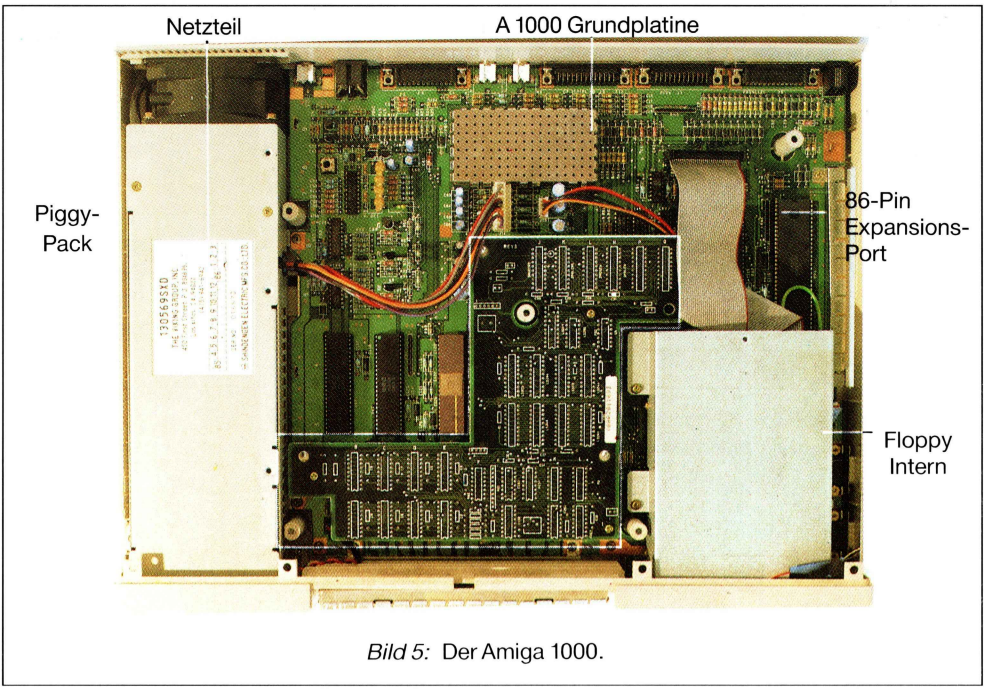


Bild 5: Der Amiga 1000.

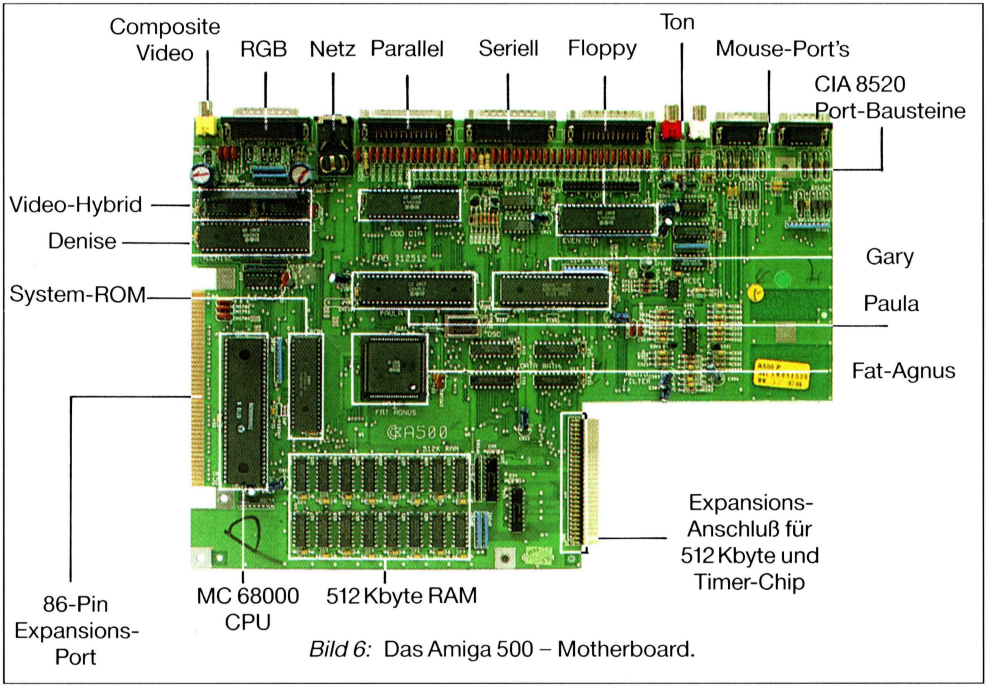


Bild 6: Das Amiga 500 – Motherboard.

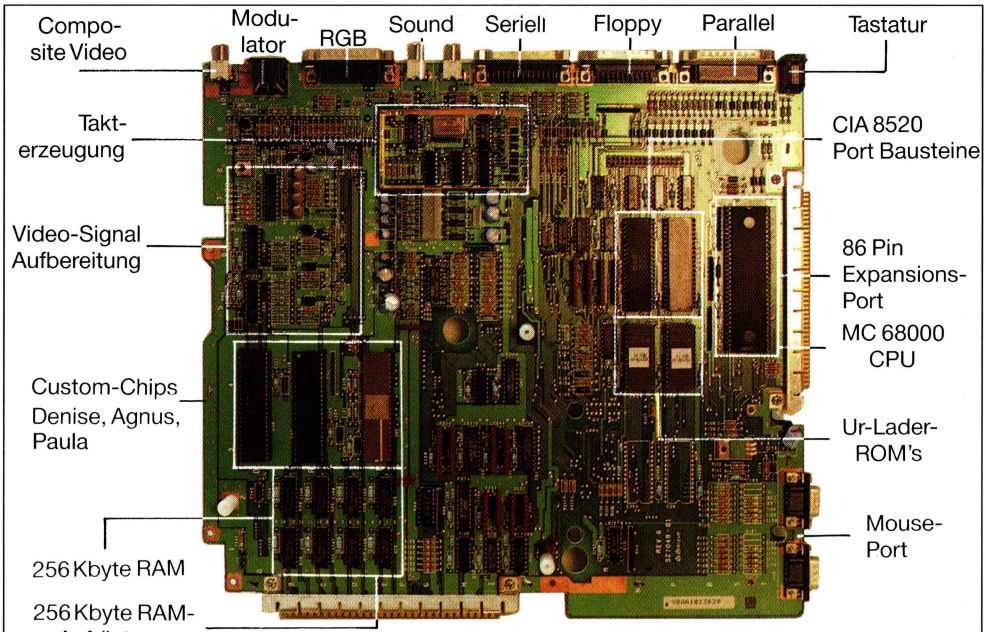


Bild 7: Das Motherboard in voller Größe.

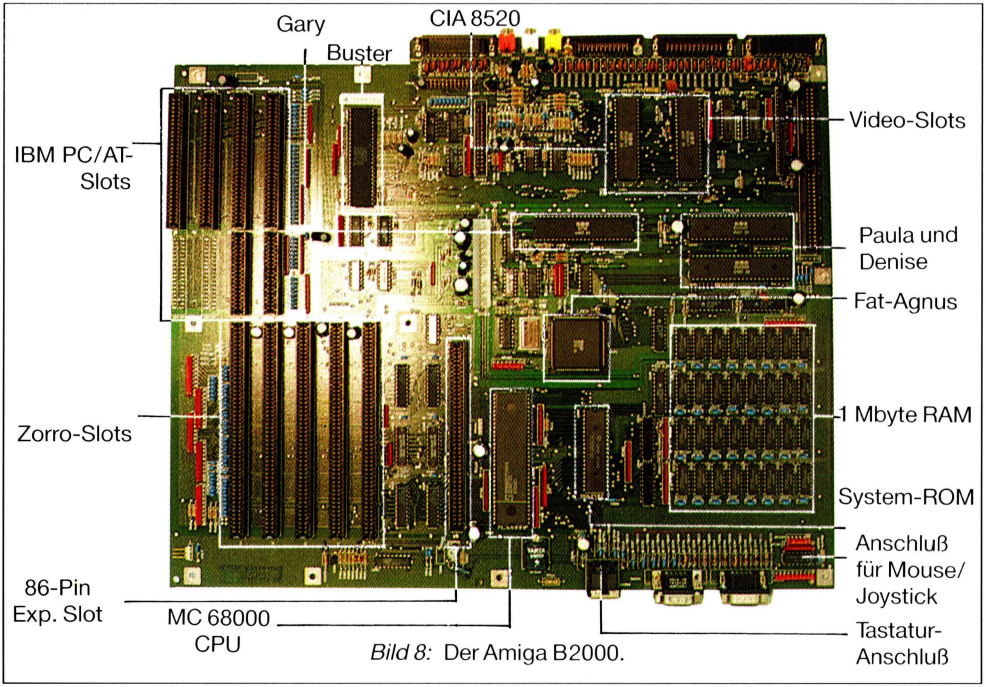


Bild 8: Der Amiga B2000.

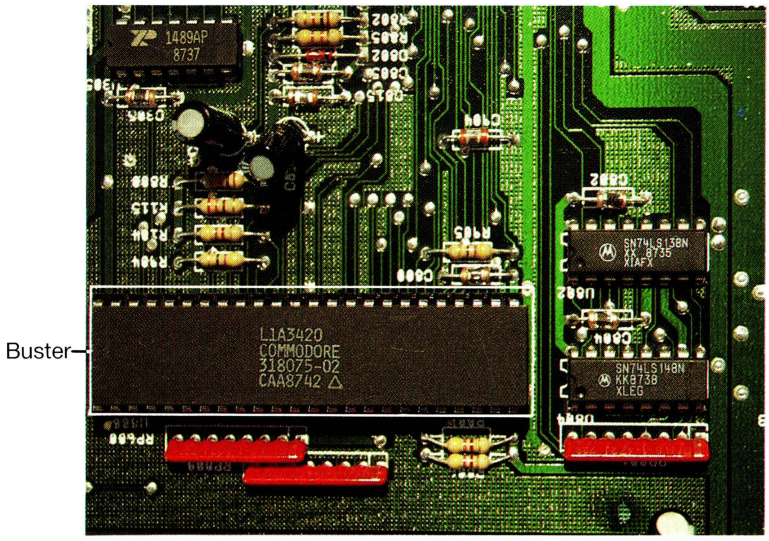


Bild 9: Buster.

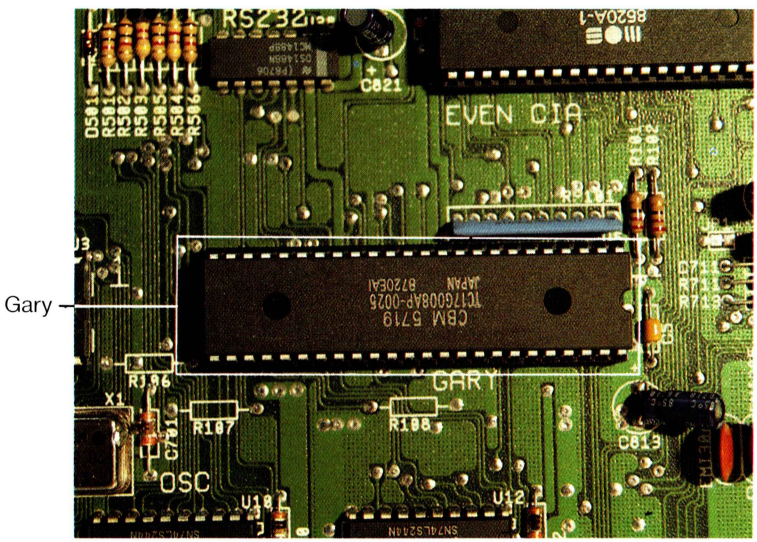


Bild 10: Der Custom-Chip Gary.

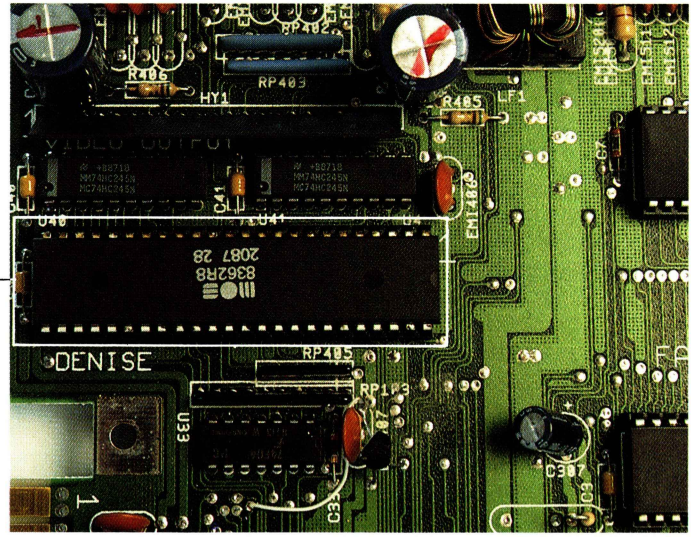


Bild 11: Der Denise-Chip.

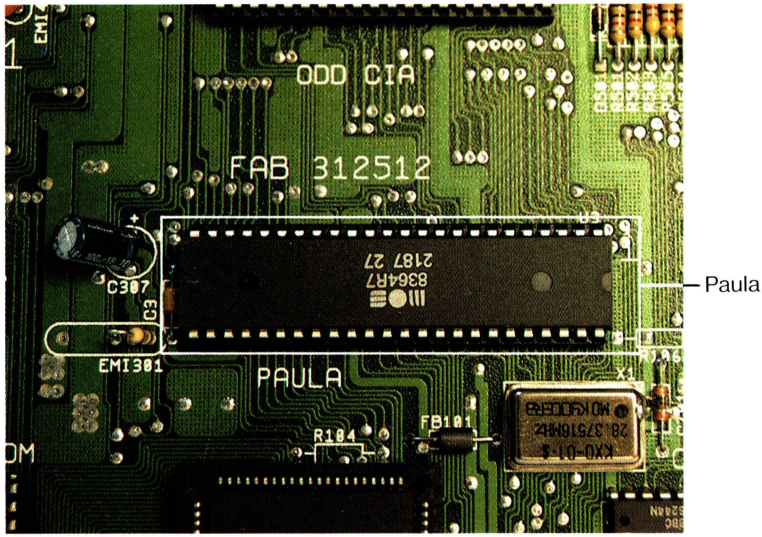


Bild 12: Der Paula-Custom-Chip.

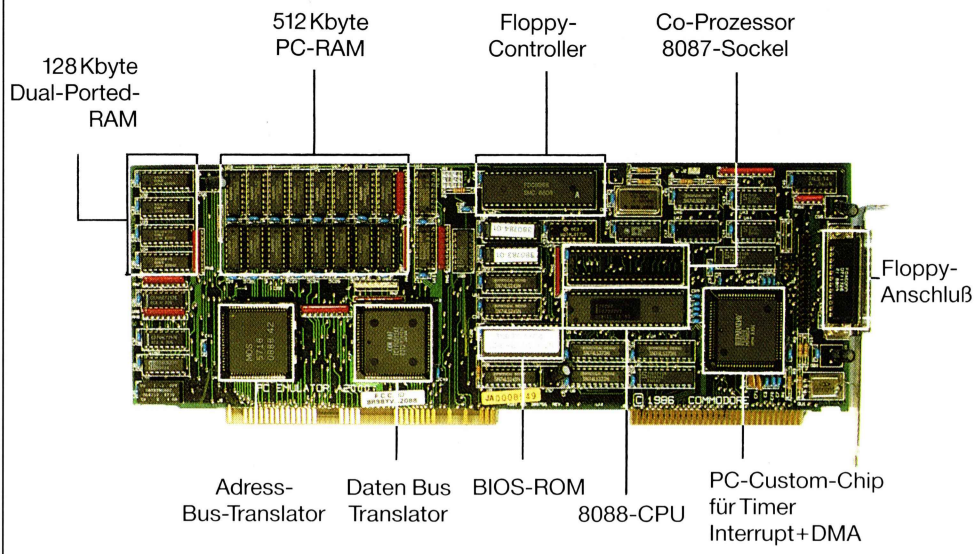


Bild 13: Ein vollwertiger PC auf kleinstem Raum.

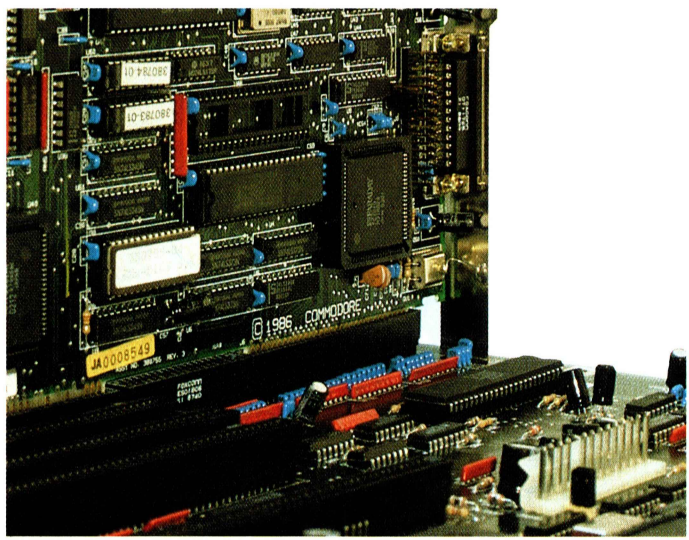


Bild 14: Das 2000er Slot-Konzept.

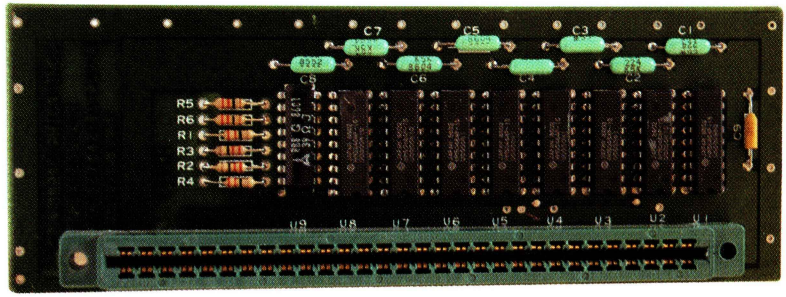


Bild 15: Die 256 KByte-Erweiterung des A 1000

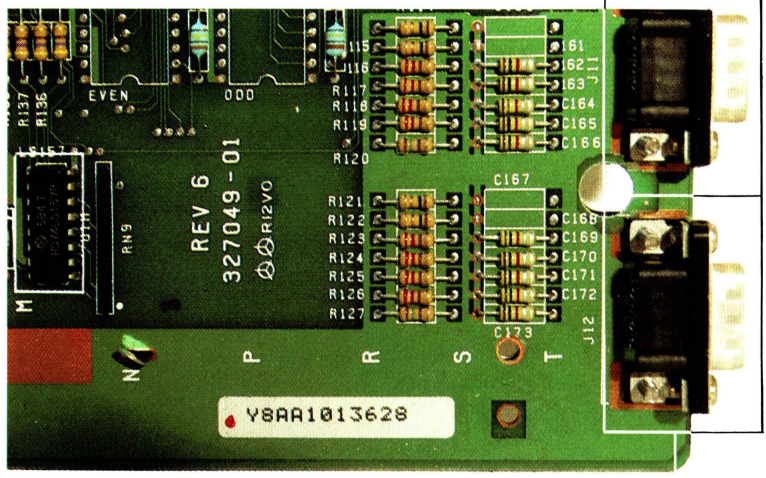


Bild 16: Vier 2 zu 1-Multiplexer setzen die ankommenden Signale von Port 0 und 1 zu Denise um.

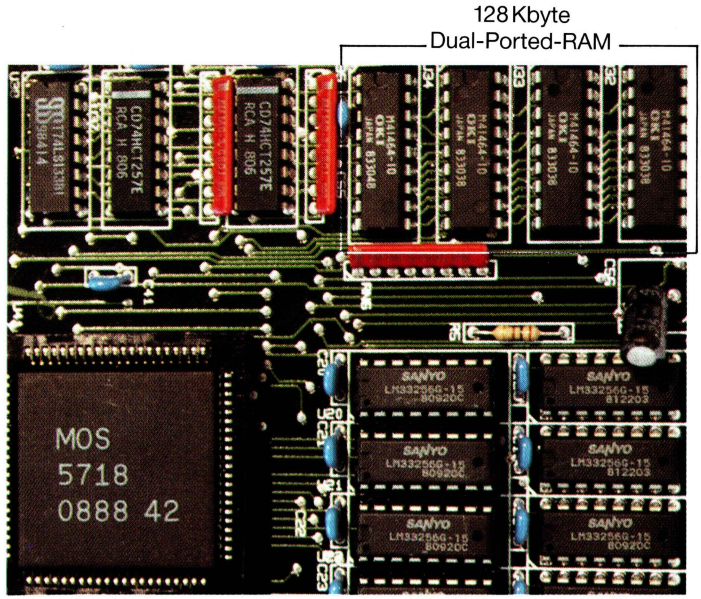


Bild 17: Das Dual-Ported-RAM der PC-Karte.

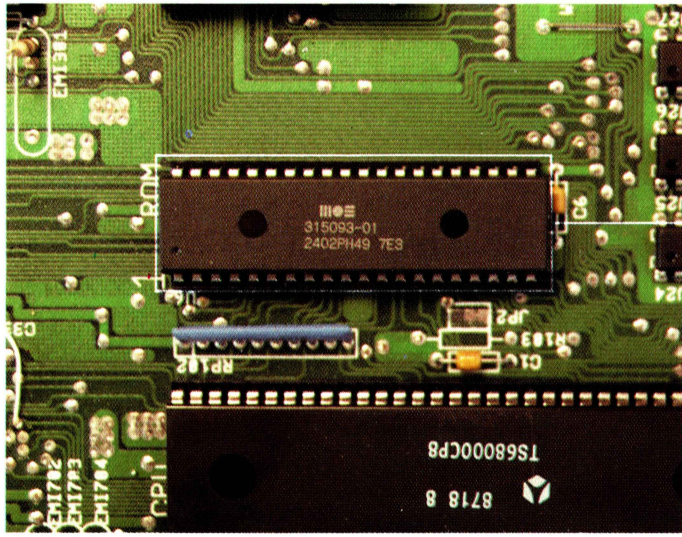


Bild 18: In diesem kleinen ROM findet beim Amiga 500 und 2000 sowohl das Kickstart als auch der Ur-Lader Platz.

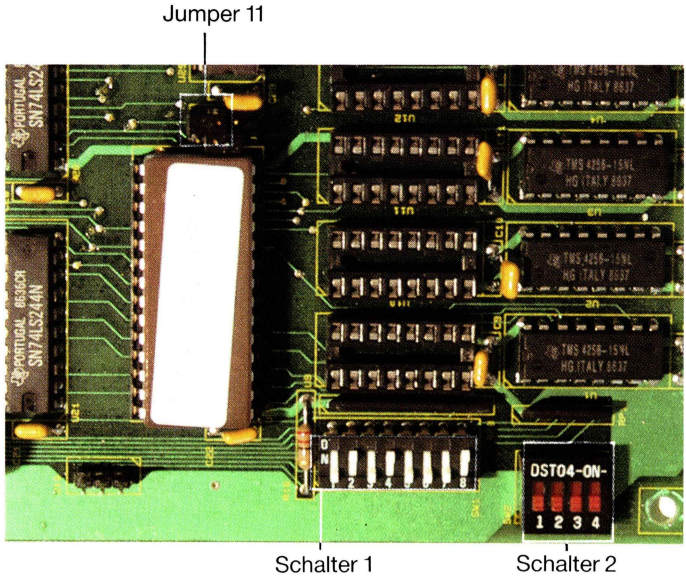


Bild 19: Die Jumper im Überblick.

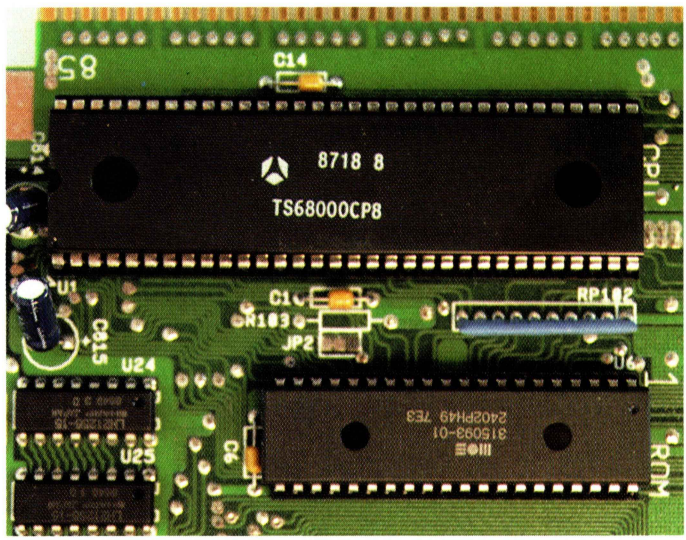


Bild 20: Der 68000er-Chip.

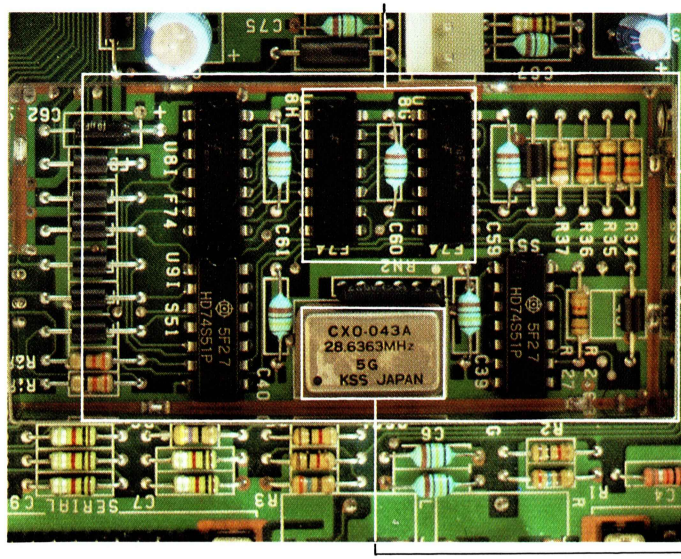


Bild 21: Die Takterzeugungs-Schaltung im Amiga 1000.

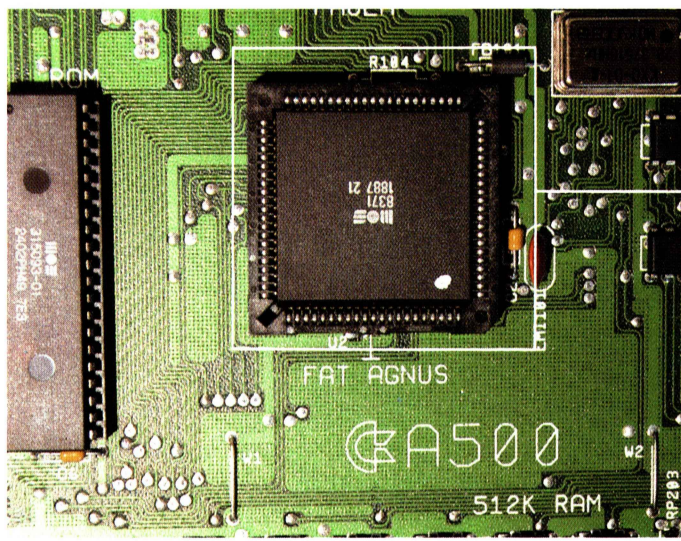


Bild 22: Der Fat-Agnus-Chip.

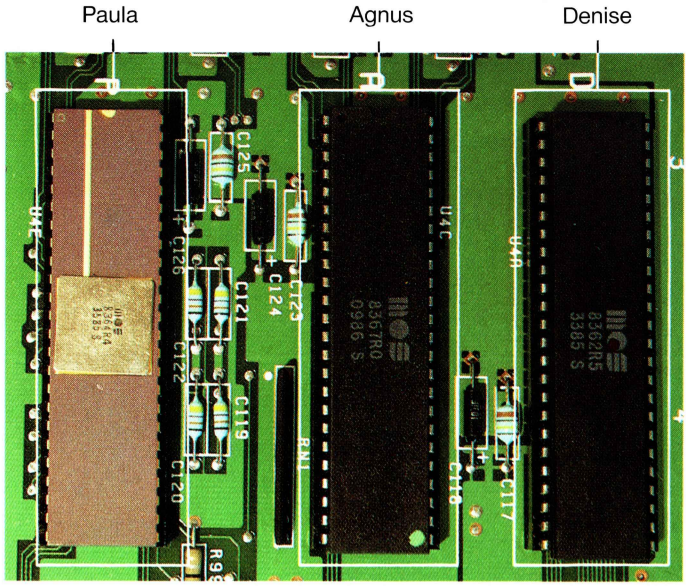


Bild 23: Agnus, Denise und Paula im Amiga 1000.

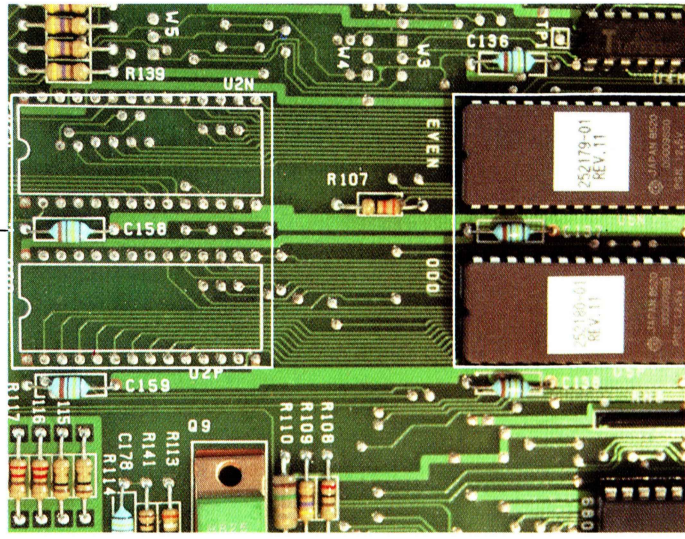


Bild 24: Die ROM's im Amiga 1000.

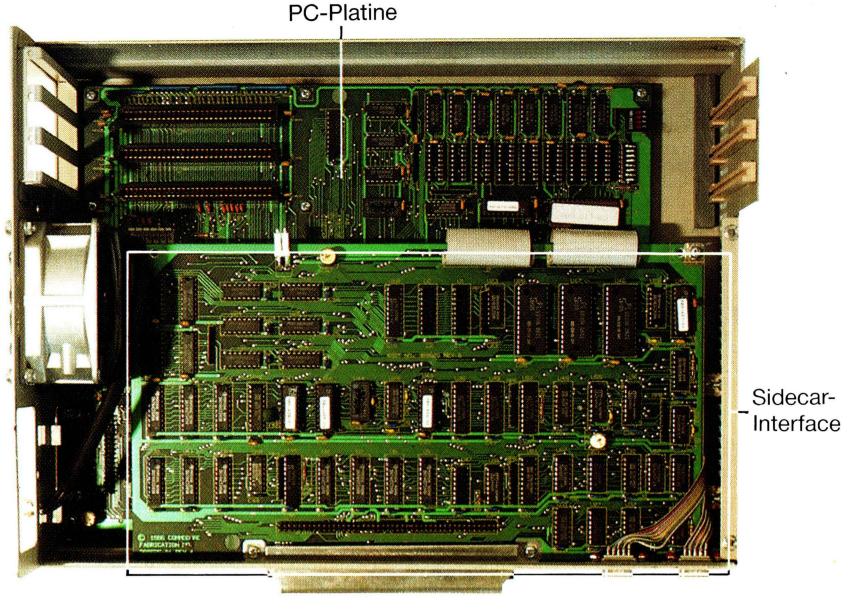


Bild 25: Das Sidecar ohne Abdeckung.

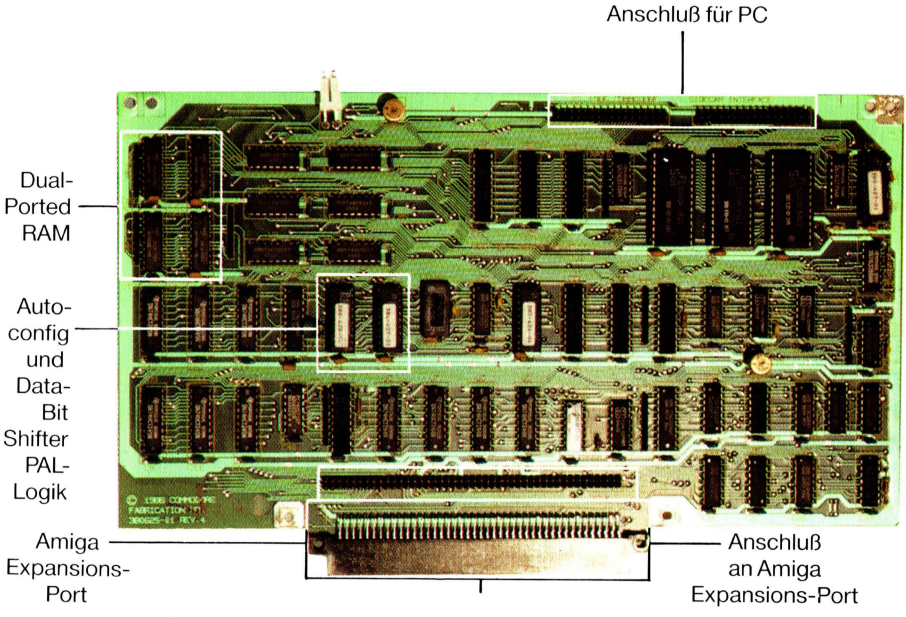


Bild 26: Das Interface zum Amiga.

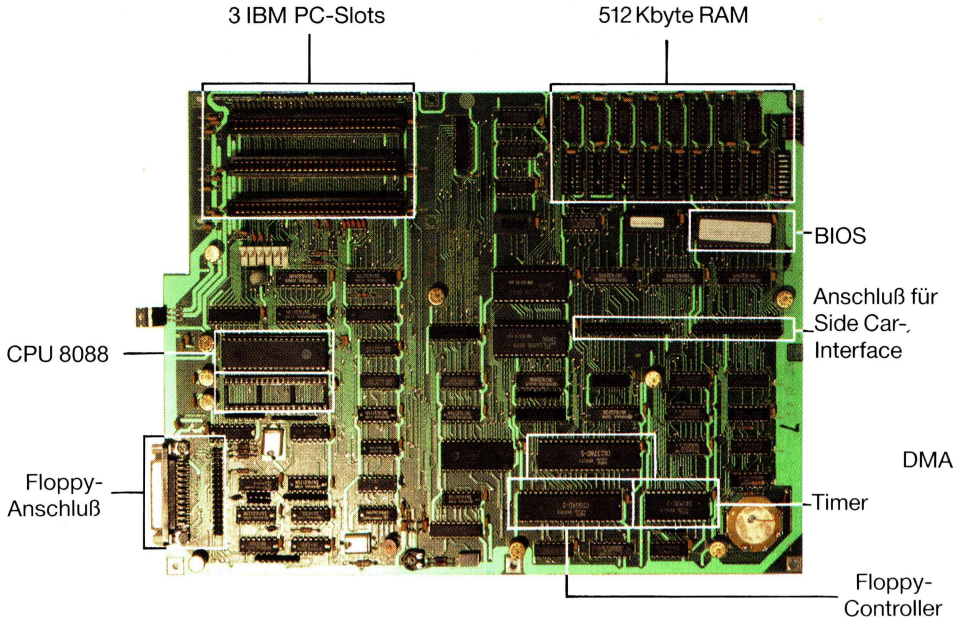


Bild 27: Das Sidecar-Motherboard.

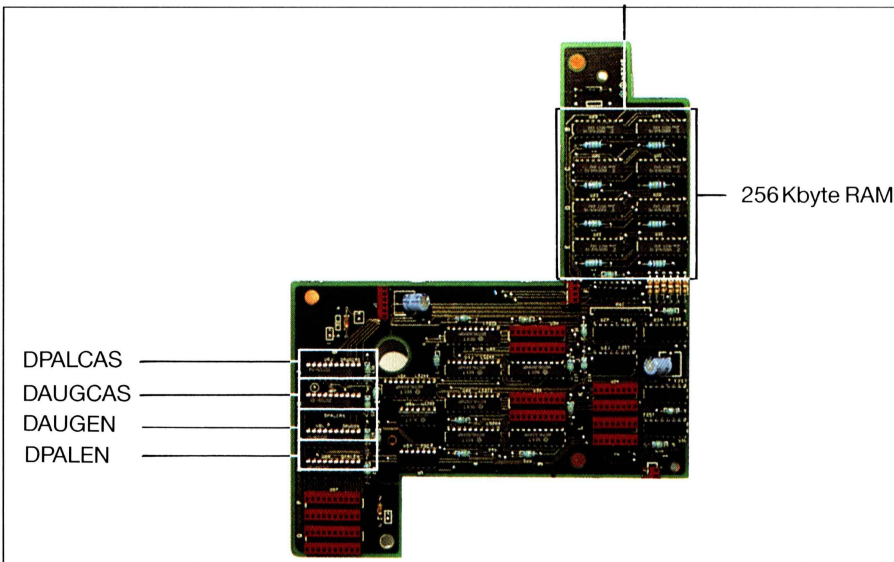


Bild 28: Das Piggy-Pack enthält vier PAL's, die wichtige Kontrollfunktionen übernehmen. Während die zwei PAL's DAUGCAS und DAUGEN fast nur für das Piggy-Pack zuständig sind, übernehmen die anderen zwei Chips Kontrollfunktionen für das Motherboard.

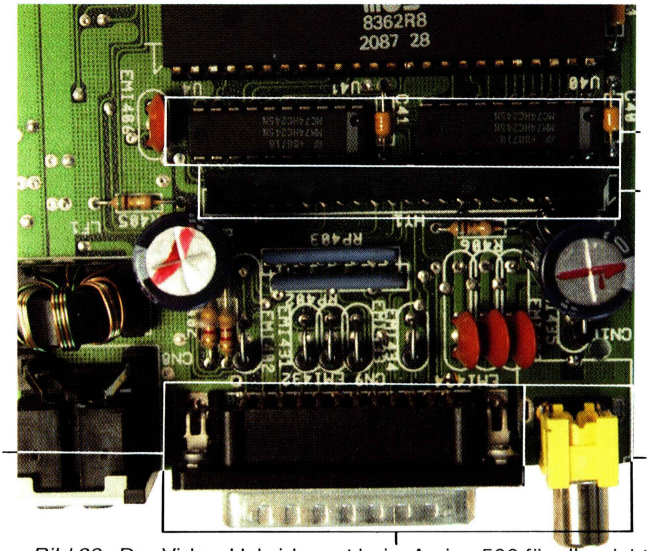


Bild 29: Das Video-Hybrid sorgt beim Amiga 500 für alle wichtigen Schaltfunktionen, um ein RGB- und Composite-Video-Signal zu erhalten. Leider liegt das Video-Signal nur in schwarz-weiß vor.

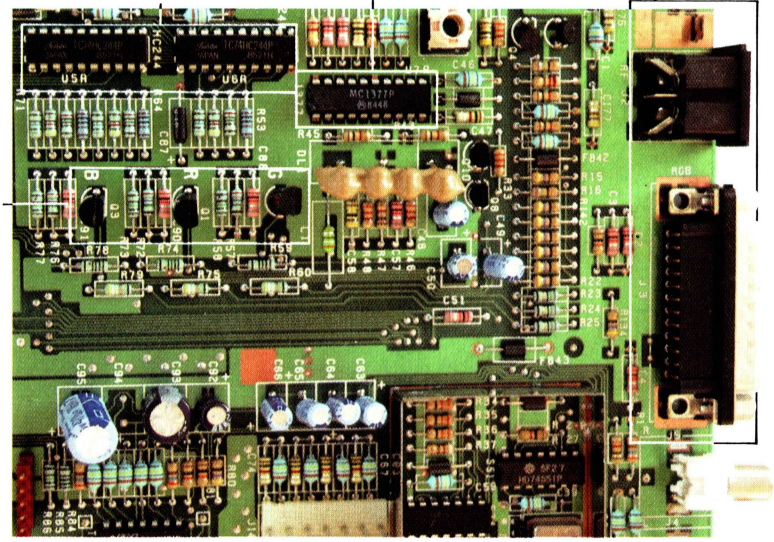


Bild 30: Der MC 1377 sorgt beim Amiga 1000 für das notwendige Composite-Signal.

CIA 8520 Port-Bausteine

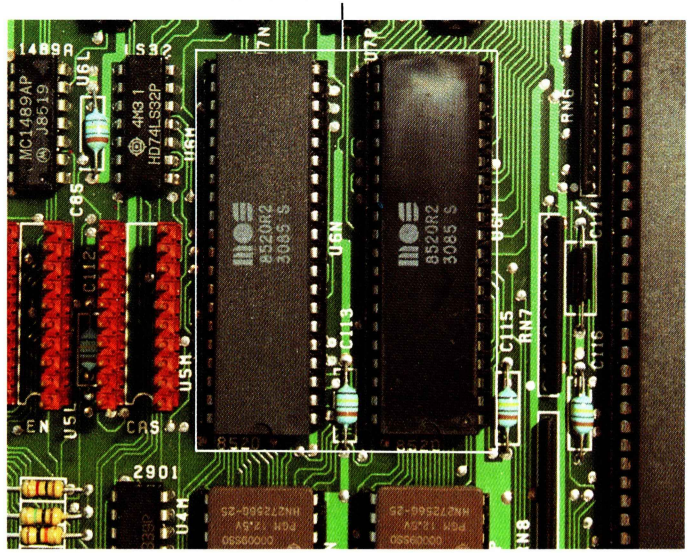


Bild 31: Die CIA 8520 Port-Bausteine.

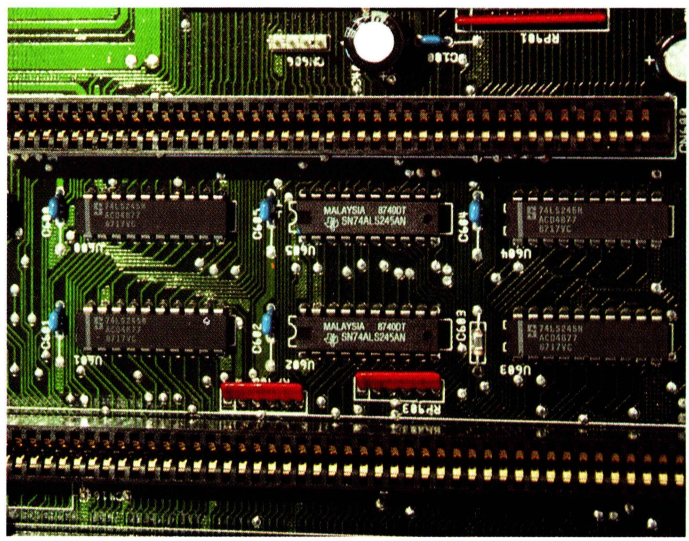


Bild 32: Der 86-Pin-, AT/PC- und Zorro-Slot im Amiga 2000.

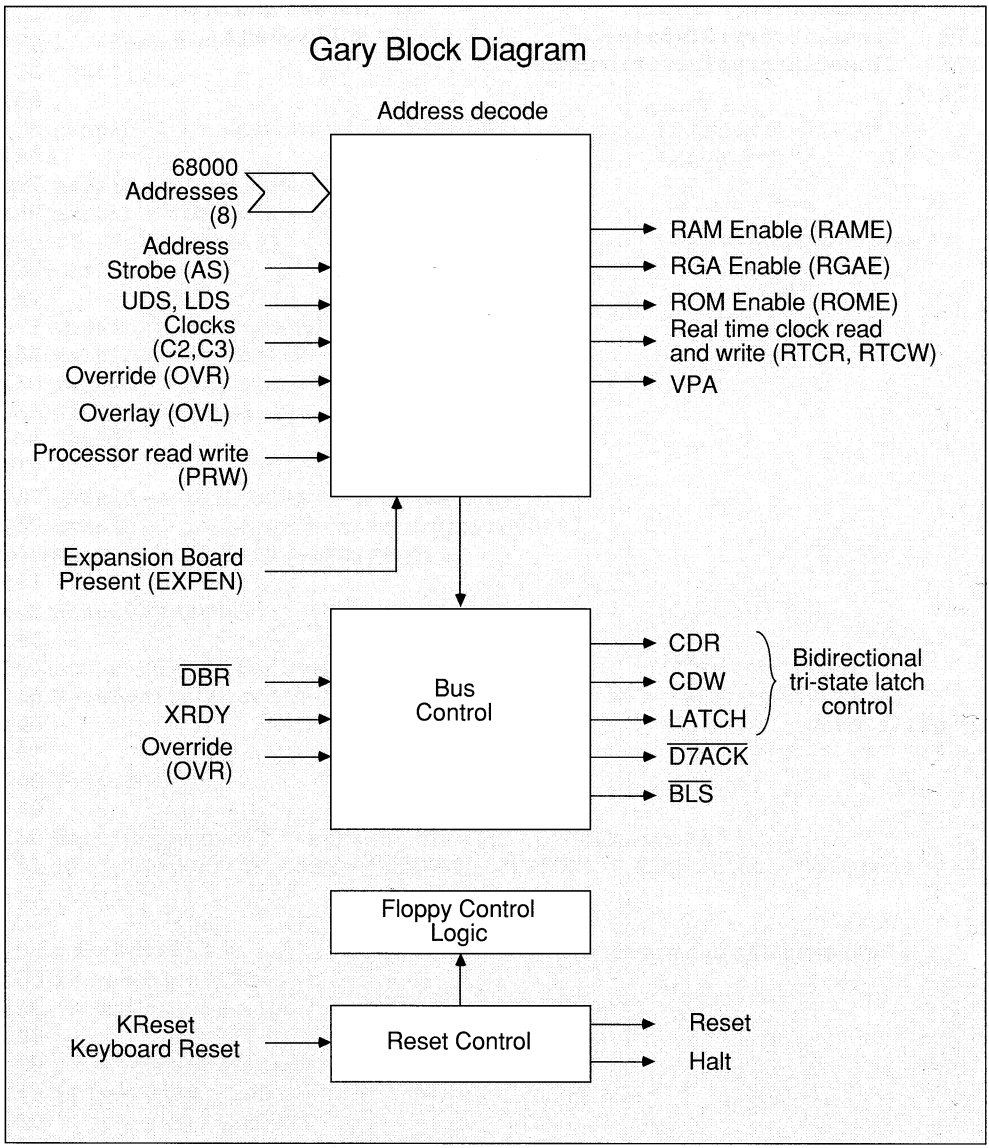
```
 63   0x7FF8, 0x4788,
 64   0x3FF0, 0x3030,
 65   0x0FC0, 0x0FC0,
 66
 67   0,0
 68  };
 69
 70 struct Simple            Sprite sprite1/* Sprite-Structure */
 71  {
 72   &Data1[0],
 73   14,                                  /* Hoehe */
 74   100,                                 /* X - Position */
 75   100,                                 /* Y - Position */
 76   2                                    /* Sprite Nummer */
 77
 78  };
 79
 80 struct NewScreen ns =                  /* Die New-Screen Structure */
 81  {
 82   0,                                   /* Linke Ecke */
 83   0,                                   /* Obere Ecke */
 84   320,                                 /* Breite */
 85   256,                                 /* Hoehe */
 86   2,                                   /* Tiefe */
 87   0,                                   /* DetailPen */
 88   1,                                   /* BlockPen */
 89   SPRITES,                             /* ViewModes */
 90   CUSTOMSCREEN,                        /* Type */
 91   NULL,
 92   NULL,
 93   NULL,
 94   NULL
 95  };
 96
 97 main() /* HAUPTPROGRAMM */
 98  {
 99   int x, y;
100   UBYTE type = GPCT_RELJOYSTICK;
101   USHORT schleife;
102
103   x = 152;
104   y = 120;
105                                        /* Libraries öffnen */
106   if ((IntuitionBase = (struct IntuitionBase *)
107    OpenLibrary("intuition.library", 0)) == 0) exit();
108
109   if ((GfxBase = (struct GfxBase *)
110    OpenLibrary("graphics.library", 0)) == 0) exit();
111                                        /* Screen öffnen */
112   if ((screen = (struct Screen*) OpenScreen(&ns)) == NULL) exit();
```

```
113 SetRGB4(&screen->ViewPort,20,9,9,9);              /* Farben setzen */
114 SetRGB4(&screen->ViewPort,21,11,11,11);
115 SetRGB4(&screen->ViewPort,22,13,13,13);
116 SetRGB4(&screen->ViewPort,23,15,15,15);
117
118 if (!(gameport = CreatePort (0,0))) /* Gameport 'öffnen' */
119  exit();
120 if (!(gameio = CreateStdIO (gameport)))
121  exit();                                          /* Device öffnen */
122 if (OpenDevice ("gameport.device", 1, gameio, 0))
123  exit();
124
125 gameio -> io_Command = GPD_SETCTYPE;          /* Request-Structure
126                                                  erstellen */
127 gameio -> io=Lenght = 1;
128 gameio -> io_Data = &type;
129 if (DoIO (gameio))                            /* Device einstellen */
130  exit();
131
132 gameio -> io_Command = GPD SETTRIGGER;
133 gameio -> io_Length = sizeof (gpt);
134 gameio -> io_Data = &gpt;
135 if (DoIO (gameio))
136  exit();
137
138 gameio -> io_Command = GPD READEVENT;
139 gameio -> io_Length = sizeof (joyreport);
140 gameio -> io_Data = &joyreport;
141
142 SendIO (gameio);
143
144 schleife = GetSprite(&spritel,3);             /* Sprite 'holen' */
145 MoveSprite(&screen->ViewPort, &spritel, 152, 120); /* Sprite
146                                                       darstellen */
147
148 for(;;)
149  {
150 WaitIO (gameio); /* Warten, bis Signal vom Gameport */
151 if (joyreport.ie_Code == IECODE_LBUTTON) /* Wenn Button, dann Ende */
152  break;
153
154 x += joyreport.ie_X;                      /* Sonst Sprite bewegen */
155 if (x < 0 || x > 304)
156     x = x<0 ? 0 : 304;
157
158 y += joyreport.ie_Y;
159 if (y < 0 || y > 240)
160     y = y<0 ? 0 : 240;
161 MoveSprite(&screen->ViewPort, &spritel, x, y);
162
```

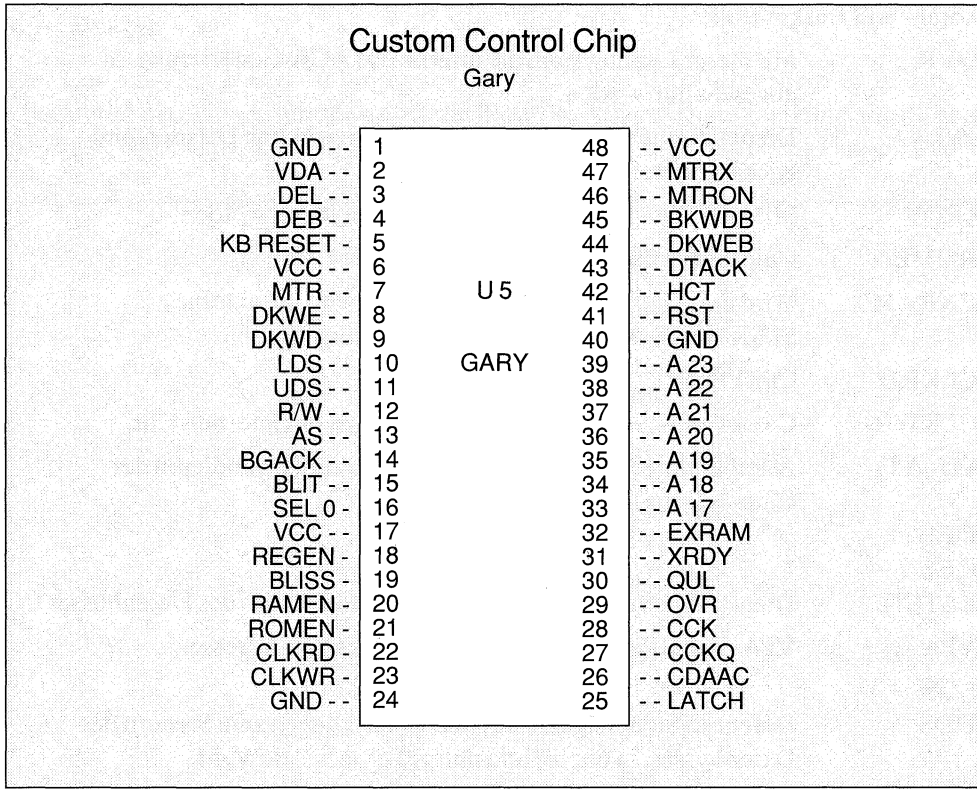
```
163 SendIO(gameio);
164 }
165
166   CloseDevice(gameio); /* Device schließen */
167   DeleteStdIO(gameio);
168   DeletePort(gameport);
169
170   FreeSprite(3);                    /* Sprite loeschen */
171   CloseScreen(screen);              /* Screen und Libs */
172   CloseLibrary(GfxBase);            /* schließen */
173   CloseLibrary(IntuitionBase);
174 }
```

4.4: Gary

Gary ist einer der neuesten Custom-Chips des Amiga. Er ist nur bei dem B2000 und A500 zu finden. Er übernimmt überwiegend verschiedene Kontrollaufgaben, die bei dem A2000 und A1000 mit diskreter Logik und PAL's gelöst wurde. Sein Aufbau kann praktisch in vier Teile gegliedert werden: Adreßdekodierung, Kontrolllogik für das Floppy DF0:, Bus- und Resetkontrolle (Bild 10 im Farbteil).



Z 4.4-1: *Das Blockdiagramm des Custom-Chip Gary*



Z 4.4-2: *Die Pinbelegung des Custom-Chip Gary*

Hier die Beschreibung der Pins:

Floppy-Funktionen:

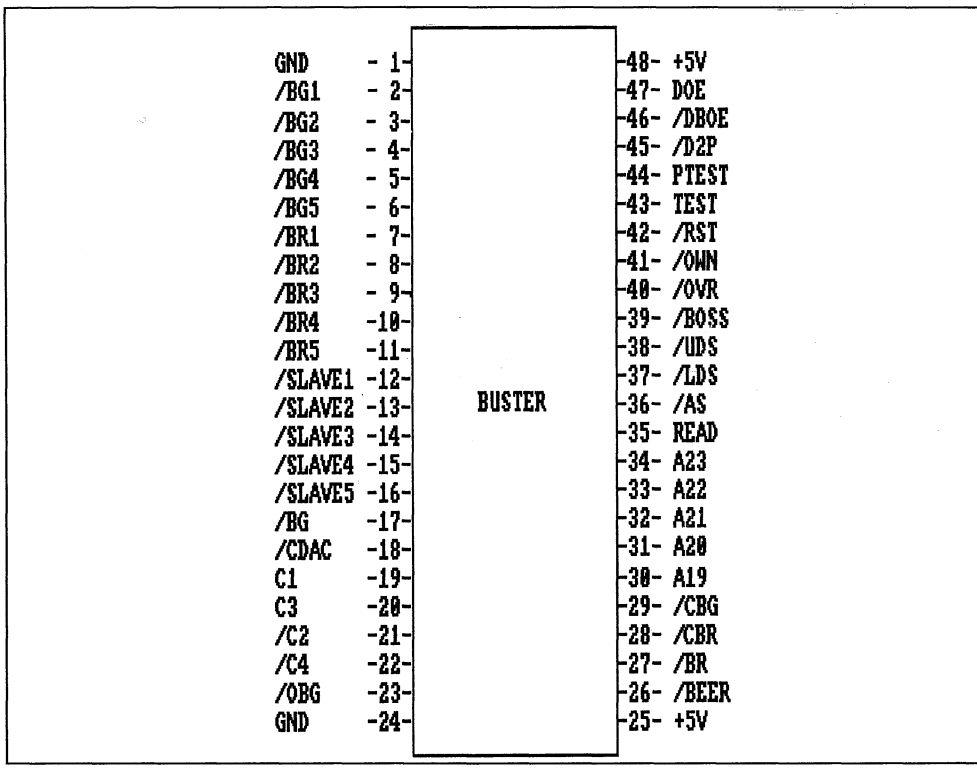
/DKWD	Diskwrite. Floppy-Schreibsignal.
DKWE	Diskwrite Enable. Dient zur Freigabe des Schreibsignals.
/MTR	Signal zum Einschalten des Motors von DF0:.
/SEL0	Select-Signal für Laufwerk DF0:.
DKWDB	Gepuffertes Floppy-Schreibsignal.
DKWEB	Gepuffertes Freigabesignal.
MTRX	Dient zum Einschalten externer Floppymotoren.
MTRON	MotorOn-Signal für internes Laufwerk. Dieses Signal wird aus dem /MTR-Signal intern durchTakten eines D-FlipFlops mit dem / SEL0-Signal gewonnen.

Adreß- und Buskontrolle:

/OVR	Mit dieser Leitung kann die interne /DTACK-Generierung abgeschaltet werden.
OVL	Dient bei einem Reset zum Umschalten zwischen Urlader- und Kickstartbereich.
XRDY	Mit XRDY kann das /DTACK-Signal verzögert werden.
/ROMEN	Chip-Select für das Betriebssystem-ROM.
/EXRAM	Wird diese Leitung auf GND gelegt, so wird die externe 512-Kbyte-Erweiterung in das System eingefügt.
/CLKRD	ClockRead.
/CLKWR	ClockWrite. Steuert das Lesen/Schreiben der internen Uhr.
A17–A23	Adreßleitungen A17 bis A23. Sie dienen zum Dekodieren der Chip-Select-Signale.
/DEB /OEL /LATCH	Dienen zum Steuern der Bus-Treiber und Latches des Datenbusses.
/VPA	VPA-Signal des MC68000. Gültige Prozessor-Adresse.
/LDS /UDS	Datenbus Steuersignale des MC68000. Dienen zum Steuern des Datenbusses, in oberes und unteres Byte, sowie Word.
R/W	R/W-Leitung des MC68000. Schreib-/Leseleitung.
/AS	/AS-Leitung des MC68000. Adreß-Strobe signalisiert, daß die Adressen gültig sind.
/DTACK	/DTACK-Signal. Dient zum Verzögern des Schreib-/Lesevorgangs bei langsamen Peripherien.
/BGACK	/BGACK-Signal des MC68000.
/HLT	/HLT-Signal des MC68000. Mit diesem Signal kann der MC68000 angehalten werden.
/RST	/RST-Signal. Dies ist das Reset-Signal.
/RAMEN	RAM Enable-Leitung für AGNUS.
/REGEN	Register Enable. Register-Gültigkeitsleitung für AGNUS.
/BLISS /BLIT	Leitungen für Blitter-Aufgaben.
/CDAC	System-Clock 7.16 MHz um 90 Grad verschoben zum System-Takt.
/CCK /CCKQ	Takte für das Farbträgersignal.

4.5: Buster

Der Custom-Chip Buster ist bisher nur bei den Amiga-B2000-Rechnern zu finden. Er übernimmt hier die »Kontrolle« über die Slots des Amiga. Diese Funktion wurde beim Amiga A2000 noch diskret mit PAL's und diverser Logik aufgebaut (siehe Farbteil Bild 9).



Z 4.5-1: *Die Pin-Belegung des Buster*

Hier die Pinbeschreibung zu Buster:

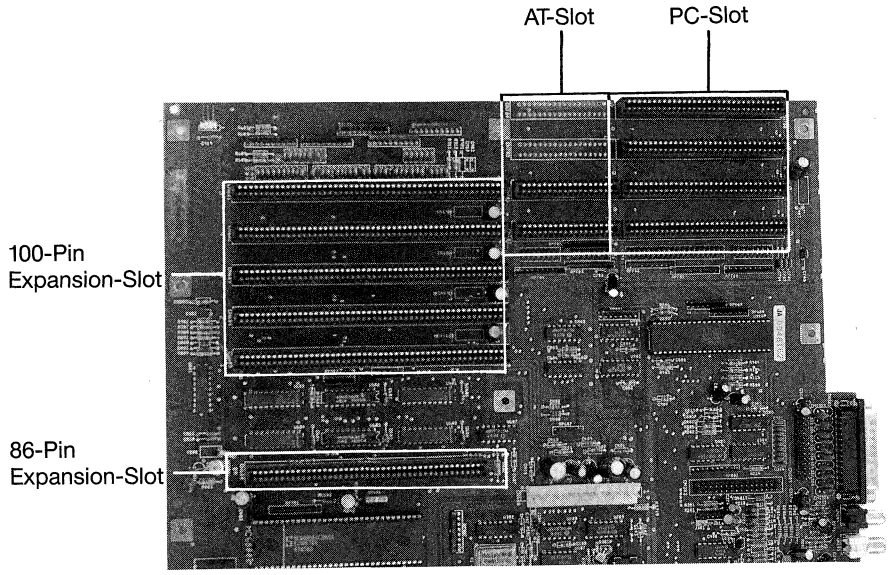
/BR1–BR5, /CBR	Damit signalisieren externe Erweiterungen, daß sie auf den Bus zugreifen möchten.
/BG1–/BG5 /CBG	Busfreigabe-Signal. Mit diesem Signal wird darauf hingewiesen, daß sein /BR-Signal berücksichtigt wird.
/SLAVE1– /SLAVE5	/SLAVE-Signal für jeden Slot. Genauere Beschreibung siehe »Die Signale des 100-Pin-Slot«.
A19–A23	Adreßleitungen A19–A20.

/BEER	Diese Leitung signalisiert dem MC68000 einen Busfehler, wenn z.B. zwei Erweiterungskarten gleichzeitig auf den Bus zugreifen wollen.
/UDS /LDS	Mit dieser Leitung wird die Zugriffsart des Datenbusses signalisiert.
READ	Lese-Signal.
/OVR	Override. Damit kann die /DTACK-Generierung ausgeschaltet werden.
/OWN	DMA-Owner. Beansprucht wird diese Leitung, wenn eine Erweiterungskarte eine Bus-DMA besitzt und sie die Kontrolle über den Bus erhält.
/AS	Adreß-Strobe-Leitung des MC68000.
/RST	Reset-Signal.
/CDAC, C1 C1, /C2 C3, /C4	System-Takte. Nähere Erläuterung finden Sie im Kapitel »Die Takte des Amiga«.
PTEST TEST	Pins zum Testen des Buster.
/DBOE DOE	Data Output Enable. Mit diesen Signalen kann der Datenbus-Puffer auf einer Erweiterungskarte geschaltet werden.

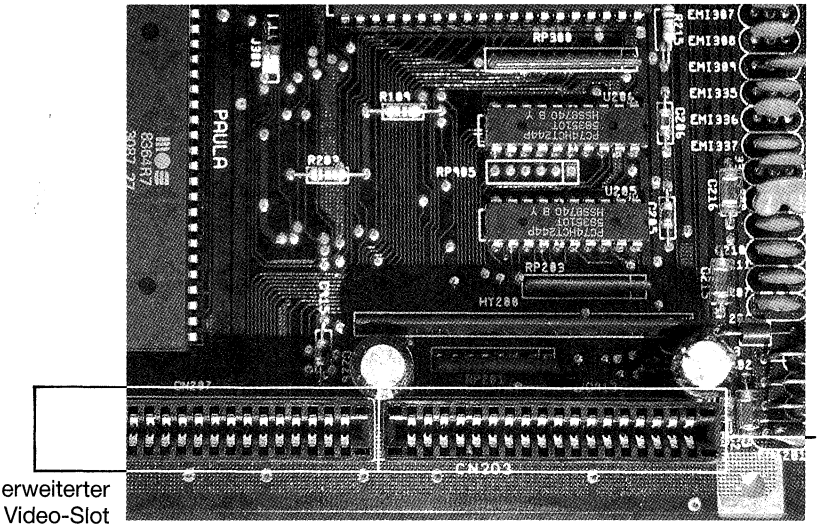
Kapitel 5

Die Amiga-Slots

Bei der Amiga-Serie stehen inzwischen 6 verschiedene Slots zur Verfügung. Während beim Amiga 500 und Amiga 1000 die Standard 86-Pin-Slots verwendet werden, kamen beim Amiga A2000 noch zusätzlich 100-Pin-Zorro-Slots und AT- bzw. PC- Steckplätze hinzu. Weiterhin enthalten die 2000er einen bzw. der B2000 zwei Video-Slots. Durch diese Vielzahl an Steckplätzen bietet der Amiga ungeahnte Möglichkeiten in Bezug auf Erweiterungen.



F5-1: Der 86-Pin-Slot des Amiga 2000.

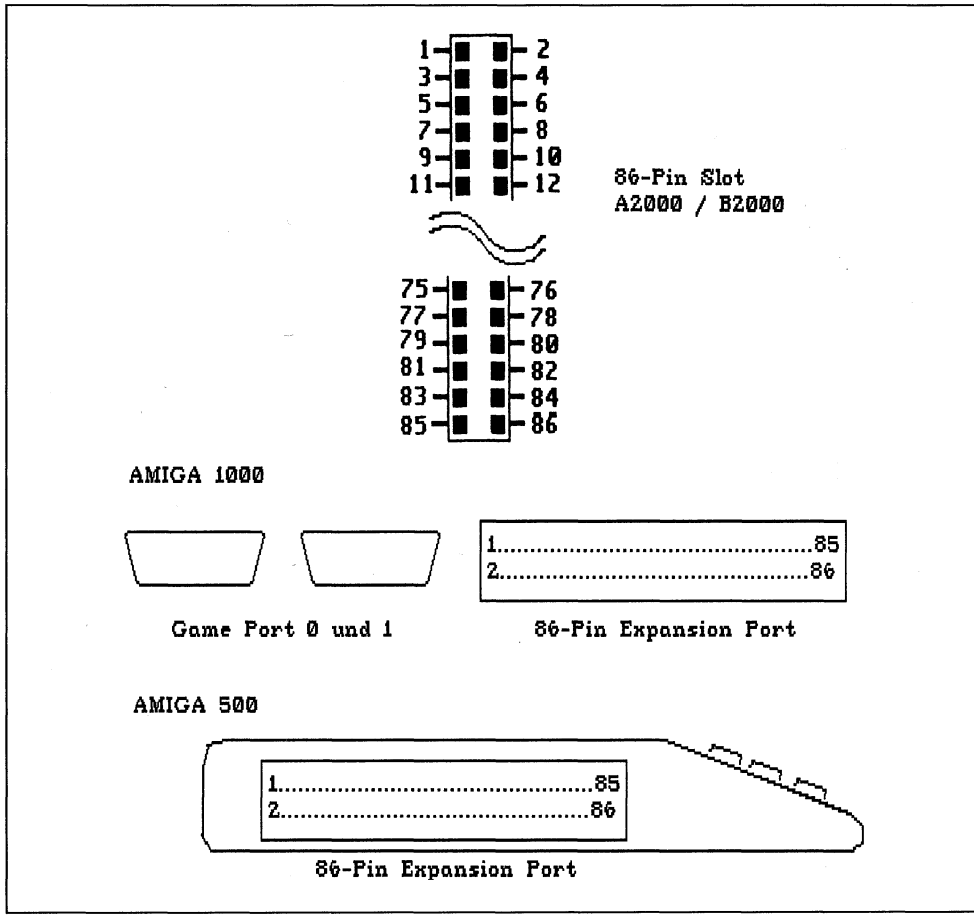


F5-2: Für Video-Freaks offen: der Video-Slot des B2000.

Untereinander wird es hingegen etwas schwierig Erweiterungskarten auszutauschen, da der 86-Pin-Erweiterungsslot beim Amiga 500 sich auf der linken Seite befindet, und somit um 180 Grad zum A1000-Slot verdreht ist. Der Austausch von 2000er-Karten zu den Geschwister A500 und A1000 ist ebenfalls etwas problematisch, da hier der 86-Pin-Slot kein Stecker, sondern eine Buchse ist. Zorro-Slots sind bei den Amiga 500- und 1000-Rechnern leider überhaupt nicht enthalten. Das heißt jedoch noch lange nicht, daß Anwender dieser Amiga's auf 2000er-Karte verzichten müssen. Inzwischen sind im Handel sogenannte Sub-Boards erhältlich, die das Einstecken von zwei 2000er-Karten ermöglicht. Da's wichtigste für Entwickler ist jedoch, daß die Signale an den Slots untereinander kompatibel sind.

5.1: Der 86-Pin-Slot

Der 86-Pin-Slot ist sowohl beim Amiga 500 und 1000 als auch beim Amiga 2000 vorhanden. Beim Amiga 1000 wird er sichtbar, wenn eine Klappe in der Mitte der rechten Seite herausgenommen wird. Beim Amiga 500 ist er in der Mitte auf der linken Seite zu suchen. Auch beim A2000 ist er leicht zu finden, da er einer der kleinsten Slots in diesem Rechner ist. Er befindet sich hier neben den 100-Pin-Zorro-Slots. An diesem Slot können alle wichtigen Signale des Amiga abgegriffen werden. Der 86-Pin-Slot ist sozusagen der direkte Draht zum System. Auf der oberen bzw. rechten Seite liegen alle Signale in ungerader Numerierung an. Auf der unteren bzw. linken Seite befinden sich alle geraden Nummern. Zeichnung 5.1-1 zeigt die Numerierung des 86-Pin-Expansionsports.



Z 5.1-1: *Die Lage und Numerierung des Expansion-Ports bei der Amiga-Serie*

Während der 86-Pin-Slot beim Amiga 500 und 1000 direkt für Erweiterungen benutzt wird, dient er beim Amiga 2000 mehr dazu, spezielle System-Karten, wie ein Turbo-Board mit 68020-Prozessor, aufzunehmen. Hierzu sind auch ein paar Signale am 2000er- Slot hinzugefügt worden. Bild 32 im Farbteil zeigt den 86-Pin-Slot des Amiga 2000.

5.1.1: Die 86-Pin-Slot-Belegung

Die nun folgende Tabelle zeigt die Funktion des entsprechenden Pins des 86-Pin-Slots bei dem jeweiligen Amiga:

Pin	Funktion	A500	A1000	A2000	B 2000
1	Ground	×	×	×	×
2	Ground	×	×	×	×
3	Ground	×	×	×	×
4	Ground	×	×	×	×
5	+5VDC	×	×	×	×
6	+5VDC	×	×	×	×
7	nicht belegt	×	×	×	×
8	−5VDC	×	×	×	×
9	nicht belegt 28 MHz Clock	×	×	×	×
10	+12VDC	×	×	×	×
11	nicht belegt /COPCFG	×	×	×	×
12	Config In	×	×	×	×
13	Ground	×	×	×	×
14	/C3 Clock	×	×	×	×
15	CDAC	×	×	×	×
16	/C1 Clock	×	×	×	×
17	/OVR	×	×	×	×
18	RDY	×	×	×	×
19	/INT2	×	×	×	×
20	/PALOPE nicht belegt /BOSS	×	×	×	×
21	A5	×	×	×	×
22	/INT6	×	×	×	×

Pin	Funktion	A500	A1000	A2000	B 2000
23	A6	×	×	×	×
24	A4	×	×	×	×
25	Ground	×	×	×	×
26	A3	×	×	×	×
27	A2	×	×	×	×
28	A7	×	×	×	×
29	A1	×	×	×	×
30	A8	×	×	×	×
31	FC0	×	×	×	×
32	A9	×	×	×	×
33	FC1	×	×	×	×
34	A10	×	×	×	×
35	FC2	×	×	×	×
36	A11	×	×	×	×
37	Ground	×	×	×	×
38	A12	×	×	×	×
39	A13	×	×	×	×
40	/IPLO	×	×	×	×
41	A14	×	×	×	×
42	/IPL1	×	×	×	×
43	A15	×	×	×	×
44	/IPL2	×	×	×	×
45	A16	×	×	×	×
46	/BEER	×	×	×	×
47	A17	×	×	×	×
48	/VPA	×	×	×	×
49	Ground	×	×	×	×

Pin	Funktion	A500	A1000	A2000	B2000
50	E Clock	×	×	×	×
51	/VMA	×	×	×	×
52	A18	×	×	×	×
53	/RST	×	×	×	×
54	A19	×	×	×	×
55	/HLT	×	×	×	×
56	A20	×	×	×	×
57	A22	×	×	×	×
58	A21	×	×	×	×
59	A23	×	×	×	×
60	/BR /CBR	×	×	×	×
61	Ground	×	×	×	×
62	/BGACK	×	×	×	×
63	D15	×	×	×	×
64	/BG /CBG	×	×	×	×
65	D14	×	×	×	×
66	/DTACK	×	×	×	×
67	D13	×	×	×	×
68	R/W	×	×	×	×
69	D12	×	×	×	×
70	/LDS	×	×	×	×
71	D11	×	×	×	×
72	/UDS	×	×	×	×
73	Ground	×	×	×	×
74	/AS	×	×	×	×
75	D0	×	×	×	×

Pin	Funktion	A500	A1000	A2000	B 2000
76	D10	×	×	×	×
77	D1	×	×	×	×
78	D9	×	×	×	×
79	D2	×	×	×	×
80	D8	×	×	×	×
81	D3	×	×	×	×
82	D7	×	×	×	×
83	D4	×	×	×	×
84	D6	×	×	×	×
85	Ground	×	×	×	×
86	D5	×	×	×	×

5.1.2: Die Signale des 86-Pin-Slots

In diesem Kapitel soll auf die Signale des 86-Pin-Slots näher eingegangen werden. An diesem Slot liegen überwiegend die ungepufferten Signale des MC68000. Neben den üblichen Spannungsversorgungen sind auch Kontrollsignale und Systemtakte an den 86-Pin-Slot herangeführt.

Die Spannungsversorgung des 86-Pin-Slots:

An diesem Slot stehen drei verschiedene Spannungen, die sich auf GND (Ground = Masse) beziehen, zur Verfügung. Je nach Amiga kann dieser Spannungsversorgung eine gewisse Leistung, die durch das jeweilige Netzteil begrenzt ist, entnommen werden. Leider lagen uns keine genauen Werte vor, so daß wir teilweise die totale, also maximale, vom Netzteil bestimmte Leistung, angeben mußten.

+12V – Dies ist die höchste Spannung, die an dem 86-Pin-Slot anliegt. Sie wird z.B. für HardDisk's oder Floppylaufwerke beim Amiga 2000 benötigt:
8 Ampere total beim Amiga 2000
1 Ampere total beim Amiga 1000 und 500

+5V – Diese Spannung wird als Hauptversorgungsspannung bezeichnet, da sie bei allen Chips in den Amigas benötigt wird. Die totale Versorgung des A2000 liegt bei 20 Ampere, beim A500 bei ca. 4.5 Ampere:

2 Ampere direkt beim Amiga 2000
1 Ampere direkt beim Amiga 1000
unter 1 Ampere beim Amiga 500

–5V – Bei dieser negativen Versorgungsspannung ist die Belastbarkeit bedeutend geringer als bei einer positiven Spannung:
ca. 0.3 Ampere total beim AMIGA 2000, 1000 und 500

Die MC68000-Signale des 86-Pin-Slots:

Die MC-68000-Signale sind direkt vom MC68000-Prozessor zum 86-Pin-Slot durchgeschleift. Sie liegen somit nicht in gepufferter Form vor.

A1–A23	A1–A23 geben die Adreßleitungen an, mit dem ein max. Speicher von 8 Mbyte adressiert werden kann.
D0–D15	Dies ist der Datenbus des MC68000. Er ist 16–Bit breit (D0–D15) und kann Word- oder Byte-Weise angesprochen werden.
FC0–FC3	Die Function-Code-Ausgänge, auch als Status-Anzeige bezeichnet, signalisieren der Hardware, auf welchen Daten- oder Programmbereich momentan zugegriffen wird. Diese Signale werden für eine MMU (Memory Managment Unit), einer sogenannten Speicherverwaltungs-Einheit benötigt.
/IPL0–/IPL2	Dies sind die 3 Interrupt-Eingänge des MC68000. Entschlüsselt ergeben sich daraus 7 Interruptprioritätsebenen.
R/W	Das Signal R/Wist der Schreib-/Lese-Signalausgang des MC68000. Ist dieses Signal log. 1, so liegt ein Lesevorgang des Datenbusses durch den MC68000 vor. Ist das Signal log. 0, so findet ein Schreibvorgang des MC68000 statt.
/AS	Adreß-Strobe (/AS) signalisiert mit abfallender Flanke des Signals, daß die Adressen auf dem Adreßbus gültig sind.
/LDS+/UDS	Mit Lower-Data-Strobe (/LDS) und Upper-Data-Strobe (/UDS) läßt sich die Steuerung des Datenbusses erkennen. /LDS zeigt hierbei an, daß es sich um einen Zugriff auf den unteren Teil des Datenbusses D0 bis D7 handelt. Dies ist gleichzusetzen mit dem Zugriff auf den Byte-Zugriff auf eine gerade Adresse. /UDS signalisiert den Zugriff auf den oberen Datenbus und somit den Byte-Zugriff auf eine ungerade Adresse.
/VMA	/VMA (Valid Memory Adress) wird für den Betrieb externer 8-Bit-Hardware, wie den 6800 benötigt. Es signalisiert, daß der MC68000 auf das E-Signal synchronisiert ist und daß die Adresse auf dem Adreßbus gültig ist.

/VPA	/VPA (Valid Peripheral Adress) bewirkt die Synchronisierung des Datentransfers über E-Clock.
/DTACK	/DTACK (DataTransfer Acknowledge) bestätigt die Datenübertragung zum MC68000. Da der Amiga ein System mit sehr schnellen Bausteinen ist, wird dieses Signal nahezu ohne Wait-States dem AMIGA zugeführt. Somit kann dieses Signal auch als ein Ausgang am Erweiterungs-Slot angesehen werden. Wird jedoch ein Bremsen des Systems verlangt, so kann mittels dem XRDY-Signal das /DTACK-Signal verlängert werden. Eine weitere Möglichkeit bietet das Abschalten der internen /DTACK-Generierung mittels /OVR.
/BERR	/BERR signalisiert dem MC68000 einen Busfehler (Buss Error). Ein solcher Fehler tritt auf, wenn z.B. während eines Zyklusablaufes auf eine ungültige Adresse zugegriffen wird.
/RST	Das bidirektionale Signal /RST (Reset) dient zur Initialisierung des Prozessors.
/HLT	Mit /HLT (Halt) kann der Prozessor am Ende des aktuellen Buszyklusses angehalten werden.
E-Clock	E-Clock dient zur Synchronisierung externer 8-Bit-Peripherie und ermöglicht somit die Zusammenarbeit des Systems mit langsameren Bausteinen.
/BR	Möchte eine intelligente Einheit auf die Busse zugreifen, so kann dies dem MC68000 durch /BR (Bus Request) mitgeteilt werden.
/CBR	/CBR ist das gepufferte /BR-Signal und liegt nur am B2000-86-Pin-Slot an.
/BG	Die /BR-Anfrage wird mit diesem Signal vom MC68000 bestätigt. Es dient sozusagen als Busfreigabe (Bus Grand).
/CBG	/CBG ist das gepufferte /BG-Signal und liegt nur an den B2000-Slot's an.
/BGACK	Ein externer Baustein signalisiert mit /BGACK (Bus Grand Acknowledge) den Empfang des /BG-Signals. Sind alle Bedingungen erfüllt, werden die Busse und ihre Steuerung von dem externen Baustein übernommen.

Die Amiga-spezifischen Signale des 86-Pin-Slots:

/INT2 /INT6 — /INT2 und /INT6 sind interne Interrupts, die von den Amiga-Chips benutzt werden. Sie werden von dem Custom-Chip Paula dekodiert.

/OVR — Mit dem /OVR (Override)-Signal kann die interne Generierung des /DTACK-Signals und die Dekodierung der RAM-Bereiche des Systems außer Kraft gesetzt werden. Die interne Verwaltung des /OVR-Signals findet beim Amiga 1000 und A2000 mit PAL's statt. Beim A500 und B2000 übernimmt hier der Custom-Chip Garry die Kontrolle. Bei dem letzteren Fall ist die Benutzung des /OVR-Signals im Bereich außerhalb von $200000 bis $9FFFFF nicht möglich.

XRDY — Durch External-Ready (XRDY) ist es möglich, das von dem System generierte /DTACK-Signal bei z.B. langsamen Speicher-Karten zu verzögern. Dieses Signal sollte nach Möglichkeit, falls Wait-States benötigt werden, sehr schnell, ca. 60 ns, nachdem die Adressen gültig sind, auf Masse gezogen werden, damit eine normale Generierung des /DTACK-Signals verhindert wird.

/COPCFG — Bei den Rechnern Amiga 500, 1000 und A2000 sind diese Pins auf Masse gelegt, da sie dort keine Bedeutung haben. Beim Amiga B2000 hingegen wird mit diesem Signal der betreffende Slot als erster Slot, der konfiguriert werden soll, festgelegt.

Die System-Clocks des 86-Pin-Slots:

/C1 — Dies ist ein Takt mit der Frequenz von 3.58 MHz, synchronisiert zu der fallenden Flanke des Systemtaktes von 7.16 MHz.

/C3 — /C3 ist im Gegensatz zu /C1 zur steigenden Flanke des Systemtaktes synchronisiert. Die Frequenz von 3.58 MHz ist gleich.

CDAC — CDAC ist ein 7.16 Takt, der in Bezug auf den Systemtakt um 90 Grad nacheilt.

28Mhz — Dies ist der Grundtakt des AMIGA, von dem alle Takte des Systems abgeleitet sind. Dieser Takt kann mit dem am RGB-Port vorhandenen, Signal /XCLKEN abgeschaltet und durch einen eigenen Takt an XCLK ersetzt werden. Dadurch ist eine Synchronisierung des AMIGAs mit externen Geräten möglich. Dieser Takt ist leider nicht beim Amiga 500 und 1000 verfügbar. Beim A2000 und B2000 findet er Anwendung bei CoProzessor-Karten, die einen hohen Takt benötigen.

5.2: Der 100-Pin-Slot

5.2.1: Die 100-Pin-Slot-Belegung

Die nun folgende Tabelle zeigt die Funktion des entsprechenden Pins des 100-Pin Slots beim Amiga A2000 und Amiga B2000:

Pin	Funktion	gepuffert	A2000	B 2000
1	Ground	–	×	×
2	Ground	–	×	×
3	Ground	–	×	×
4	Ground	–	×	×
5	+5V DC	–	×	×
6	+5V DC	–	×	×
7	/OWN	–	×	×
8	−5V DC	–	×	×
9	/SLAVEn	–	×	×
10	+12V DC	–	×	×
11	/CFGOUTn	–	×	×
12	/CFGINn	–	×	×
13	Ground	–	×	×
14	/C3 Clock	ja	×	×
15	CDAC Clock	ja	×	×
16	/C1 Clock	Ja	×	×
17	/OVR	nein	×	×
18	XRDY	nein	×	×
19	/INT2	nein	×	×
20	−12V DC	–	×	×
21	A5	ja	×	×
22	/INT6	nein	×	×
23	A6	ja	×	×

Pin	Funktion	gepuffert	A2000	B 2000
24	A4	ja	×	×
25	Ground	–	×	×
26	A3	ja	×	×
27	A2	ja	×	×
28	A7	ja	×	×
29	A1	ja	×	×
30	A8	ja	×	×
31	FC0	ja	×	×
32	A9	ja	×	×
33	FC1	ja	×	×
34	A10	ja	×	×
35	FC2	ja	×	×
36	A11	ja	×	×
37	Ground	–	×	×
38	A12	ja	×	×
39	A13	ja	×	×
40	/EINT7	nein	×	×
41	A14	ja	×	×
42	/EINT5	ja	×	×
43	A15	ja	×	×
44	/EINT4	nein	×	×
45	A16	ja	×	×
46	/BEER	nein	×	×
47	A17	ja	×	×
48	/VPA	nein	×	×
49	Ground	–	×	×
50	E Clock	nein	×	×
51	/VMA	–	×	×

Pin	Funktion	gepuffert	A2000	B 2000
52	A18	ja	×	×
53	/RST	nein	×	×
54	A19	ja	×	×
55	/HLT	nein	×	×
56	A20	ja	×	×
57	A22	ja	×	×
58	A21	ja	×	×
59	A23	ja	×	×
60	/BRn	–	×	×
61	Ground	–	×	×
62	/BGACK	nein	×	×
63	D15	ja	×	×
64	/BGn	–	×	×
65	D14	ja	×	×
66	/DTACK	nein	×	×
67	D13	ja	×	×
68	READ	ja	×	×
69	D12	ja	×	×
70	/LDS	ja	×	×
71	D11	ja	×	×
72	/UDS	ja	×	×
73	Ground	–	×	×
74	/AS	ja	×	×
75	D0	ja	×	×
76	D10	ja	×	×
77	D1	ja	×	×
78	D9	ja	×	×
79	D2	ja	×	×

Pin	Funktion	gepuffert	A2000	B2000
80	D8	ja	×	×
81	D3	ja	×	×
82	D7	ja	×	×
83	D4	ja	×	×
84	D6	ja	×	×
85	Ground	–	×	×
86	D5	ja	×	×
87	Ground	–	×	×
88	Ground	–	×	×
89	Ground	–	×	×
90	Ground	–	×	×
91	Ground	–	×	×
92	7 MHz	nein	×	×
93	DOE	–	×	×
94	/BURST	ja	×	×
95	/BG /GBG	nein ja	× 	 ×
96	/EINT1	nein	×	×
97	nicht belegt	–	×	×
98	nicht belegt	–	×	×
99	Ground	–	×	×
100	Ground	–	×	×

5.2.2: Die Signale der 100-Pin-Slots

Bei den sogenannten 100-Pin-Zorro-Slots des Amiga A/B2000 sind einige Signale im Vergleich zum 86-Pin-Slot hinzugekommen. Vollständigkeitshalber wollen wir hier jedoch nochmals auf alle Signale des 100-Pin-Slots eingehen. Dies dient der Übersichtlichkeit und erlaubt das Buch auch als Nachschlagewerk zu nutzen. Im Gegenteil zum 86-Pin-Slot sind sehr viele Signale des 100-Pin-Slots gepuffert, was die Entwicklung von Erweiterungskarten stark vereinfacht.

Die Spannungsversorgung des 100-Pin-Slots:

An dem 100-Pin-Slot stehen vier verschiedene Spannungen, die sich auf GND (Ground – Masse) beziehen, zur Verfügung. Die maximal angegebene Belastung, die abgenommen werden kann, bezieht sich auf einen Zorro-Slot des AMIGA, andernfalls ist die totale Belastung des Netzteils für die jeweilige Spannung angeben.

+12V – Dies ist die höchste Spannung, die an dem 100-Pin-Slot anliegt. Sie wird für z.B. für HardDisk's oder Floppylaufwerken beim Amiga 2000 benötigt:
8 Ampere total

−12V – Dies ist die negativ höchste Spannung, die der 100-polige Slot zur Verfügung stellt. Sie wurde zur Kompatibilität zum Zorro-Slot an den 100-poligen Expansions-Bus herangeführt:
0.3 Ampere total

+5V – Diese Spannung wird als Hauptversorgungsspannung bezeichnet, da sie bei allen Chips in den Amiga's benötigt wird:
2 bis 4 Ampere pro Expansionslot

−5V – Bei dieser negativen Versorgungsspannung ist die Belastbarkeit bedeutend geringer als bei einer positiven Spannung:
ca. 0.3 Ampere total beim Amiga 2000

Die MC68000-Signale des 100-Pin-Slots

Während beim 86-Pin-Slot die Signale direkt vom MC68000-Prozessor zum Slot durchgeschleift sind, liegen die Signale beim Zorro-Slot überwiegend in gepufferter Form vor. Die Signale sind überwiegend 100-prozentig kompatibel zu den 86-Pin-Signalen.

A1–A23 A1–A23 geben die Adreßleitungen an, mit dem der max. Speicher von 8 Mbyte adressiert werden kann. Bei Speichererweiterungskarten ist die Speichergröße von 8Mbyte möglich, welche mit der Autokonfiguration in das System eingebunden werden sollte. Ansonsten könnten Probleme mit anderen Karten auftreten. Beim Zorro-Slot liegen diese Signale in gepuffert Form vor.

D0–D15 Dies ist der Datenbus des MC68000. Er ist 16–Bit breit (D0–D15) und kann Word- oder Byteweise angesprochen werden. Die Datenleitungen sind hier im Vergleich zum 86-Pin-Slot gepuffert. Die Gültigkeit des Datenbusses beim Lesevorgang ist beim 100-Pin-Slot etwas verzögert worden, um dem Buster (bzw. den entsprechenden PAL's beim A2000) Zeit zu geben, um eine Bus-Collision herauszufinden.

FC0–FC3 Die Function-Code-Ausgänge, auch als Status-Anzeige bezeichnet, signalisieren der Hardware, auf welchen Daten- oder Programmbereich momentan zugegriffen wird. Diese Signale werden für eine MMU (Memory Managment Unit), einer sogenannten Speicherverwaltungs-Einheit benötigt.

READ Das Signal READ ist gleichzusetzen mit dem Schreib-/Lesesignal des 86-Pin-Slost. Ist dieses Signal log. 1, so liegt ein Lesevorgang des Datenbusses durch den MC68000 vor. Ist das Signal log. 0, so findet ein Schreibvorgang des MC68000 statt.

/AS Adreß-Strobe (/AS) signalisiert mit abfallender Flanke des Signals, daß die Adressen auf dem Adreßbus gültig sind.

/LDS und /UDS Mit Lower-Data-Strobe (/LDS) und Upper-Data-Strobe (/UDS) läßt sich die Steuerung des Datenbusses erkennen. /LDS zeigt hierbei an, daß es sich um einen Zugriff auf den unteren Teil des Datenbusses D0 bis D7 handelt. Dies ist gleichzusetzen mit dem Byte-Zugriff auf eine gerade Adresse. /UDS signalisiert den Zugriff auf den oberen Datenbus und somit den Byte-Zugriff auf eine ungerade Adresse.

/VMA /VMA (Valid Memory Adress) wird für den Betrieb externer 8-Bit-Hardware, wie den 6800 benötigt. Es signalisiert, daß der MC68000 auf das E-Signal synchronisiert ist und daß die Adresse auf dem Adreßbus gültig ist.

/VPA /VPA (Valid Peripheral Adress) bewirkt die Synchronisierung des Datentransfers über E-Clock.

/DTACK /DTACK (Data Transfer Acknowledge) bestätigt die Datenübertragung zum MC68000. Da der Amiga ein System mit sehr schnellen Bausteinen ist, wird dieses Signal nahezu ohne Wait-States dem Amiga zugeführt. Somit kann dieses Signal auch als ein Ausgang am Erweiterungs-Slot angesehen werden. Wird jedoch ein Bremsen des Systems verlangt, so kann mittels dem XRDY-Signal das /DTACK-Signal verlängert werden. Eine weitere Möglichkeit bietet das Abschalten der internen /DTACK-Generierung mittels /OVR.

/BERR — /BERR signalisiert dem MC68000 einen Bus-Fehler (Buss Error). Ein solcher Fehler tritt auf, wenn z.B. während eines Zyklusablaufes auf eine ungültige Adresse zugegriffen wird.

/RST — Das bidirektionale Signal /RST (Reset) dient zur Initialisierung des Prozessors.

/HLT — Mit /HLT (Halt) kann der Prozessor am Ende des aktuellen Buszyklusses angehalten werden.

E-Clock — E-Clock dient zur Synchronisierung externer 8-Bit-Peripherie und ermöglicht somit die Zusammenarbeit des Systems mit langsameren Bausteinen.

/BRn — Möchte eine intelligente Einheit auf die Busse zugreifen, so kann dies dem MC68000 durch /BR (Bus Request) mitgeteilt werden. Das n steht hier für die Slot-Nummer. Jeder Slot hat sein eigenes Signal. Dieses Signal geht zuerst zu einem Schaltkreis, der die Priorität auswertet und dann zum MC68000. Slot 1 hat hierbei die höchste, Slot 5 die kleinste Priorität. Beim Amiga B2000 ist der 86-Pin-Slot in diese Priorität mit einbezogen. Er besitzt die Priorität 0 und steht somit an erster Stelle.

/BGn — Die /BR-Anfrage wird mit diesem Signal vom MC68000 bestätigt. Es dient sozusagen als Busfreigabe (Bus Grand). Für das n steht hier ebenfalls der jeweilige Slot. Es gelten die gleichen Bedingungen wie beim /BRn.

/CBG — /CBG ist das gepufferte /BG-Signal und liegt nur an den B2000-Slots an.

/BGACK — Ein externer Baustein signalisiert mit /BGACK (Bus Grand Acknowledge) den Empfang des /BG-Signals. Sind alle Bedingungen erfüllt, werden die Busse und ihre Steuerung von dem externen Baustein übernommen.

Die Amiga spezifischen Signale des 100-Pin-Slots:

/INT2
/INT6 — /INT2 und /INT6 sind interne Interrupts, die von den Amiga-Chips benutzt werden. Sie werden von dem Custom-Chip Paula dekodiert.

/EINT1
/EINT4
/EINT5
/EINT7 — Während beim 86-Pin-Slot die nicht dekodierten Interrups IPL0 bis IPL2 anliegen, sind beim Zorro-Slot ein Teil dieser Interrupts dekodiert.

/BURST — Dieses Signal ist ein gepuffertes Resetsignal, das nur in Output-Richtung ansprechbar ist.

/OVR — Mit dem /OVR-(Override-)Signal kann die interne Generierung des /DTACK-Signals und die Dekodierung der RAM-Bereiche des Systems außer Kraft gesetzt werden. Die interne Verwaltung des /OVR-Signals findet beim Amiga 1000 und A2000 mit PAL's statt. Beim A500 und B2000 übernimmt hier der Custom-Chip Garry die Kontrolle. Bei dem letzteren Fall ist die Benutzung des /OVR-Signals im Bereich außerhalb von $200000 bis $9FFFFF nicht möglich.

XRDY — Durch External Ready (XRDY) ist es möglich, das von dem System generierte /DTACK-Signal bei z.B. langsamen Speicher-Karten zu verzögern. Dieses Signal sollte nach Möglichkeit, falls Wait-States benötigt werden, sehr schnell, ca. 60 ns, nachdem die Adressen gültig sind, auf Masse gezogen werden, damit eine normale Generierung des /DTACK-Signals verhindert wird.

/SLAVEn — Jeder Slot hat sein eigenes Slave-Signal, welches direkt am Buster anliegt. Damit wird erreicht, daß keine Busfehler o.ä. auftreten.

/CFGINn
/CFGOUTn — Über diese Pins werden die Konfigurationsprozesse gesteuert. Dabei ist zu beachten, daß der 86-Pin-Expansionport des B2000 die Priorität 0 besitzt und der letzte 100-Pin-Slot die Priorität 5.

DOE — Mit diesem Signal werden die Datenpuffer für die Datenleitungen aktiviert.

/OWN — Dieses Signal teilt dem System mit, daß die Erweiterungskarte die DMA selbstständig steuert.

Die System-Clocks des 100-Pin-Slots

/C1 — Dies ist ein Takt mit der Frequenz von 3.58 MHz, synchronisiert zu der fallenden Flanke des Systemtaktes von 7.16 MHz.

/C3 — /C3 ist im Gegensatz zu /C1 zur steigenden Flanke des Systemtaktes synchronisiert. Die Frequenz von 3.58 MHz ist gleich.

CDAC — CDAC ist ein 7.16 Takt, der in bezug auf den Systemtakt um 90 Grad nacheilt.

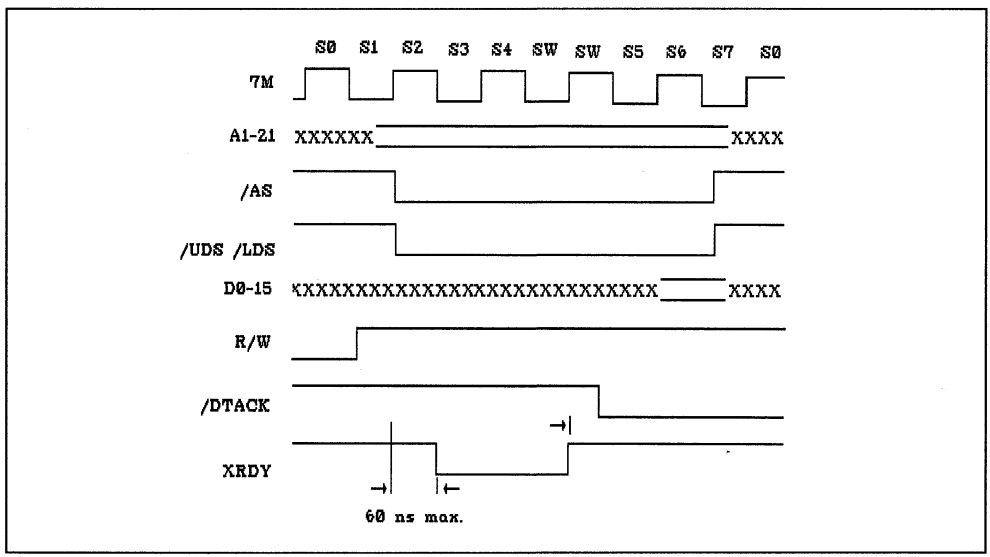
7MHz — Dies ist der 7.16-MHz-System-Takt, der gleichzusetzen ist mit dem Takt des MC68000.

5.3: Timing-Abläufe

Wichtig für Entwickler, die Hardwareerweiterungen für den 86-Pin und 100-Pin-Slot anfertigen wollen, sind die zeitlichen Abläufe der verschiedenen Signale des Amiga. Durch genaues Studieren dieser Abläufe lassen sich später bei Anwendungen wie RAM-Aufrüstungen Laufzeitprobleme der Signale meistens vermeiden. Die wichtigsten zeitlichen Abläufe eines Systems sind der Standard-Lese- und der Standard-Schreibvorgang, da hier erkannt werden kann, welche Daten zu welchem Zeitpunkt erhalten oder geschrieben werden können. Ebenfalls wichtig für Entwickler sind die System-Takte und ihre Abhängigkeit zum Grundtakt des Amiga. Insgesamt sind im System 10 verschiedene Takte vorhanden, von denen aber nur vier an die verschiedenen Erweiterungen-Slots herangeführt sind.

5.3.1: Der Standard-Lesevorgang

Interessant für den Standard-Lesevorgang sind nur die Adreßbus-, Datenbus-Signale, sowie /UDS, /LDS, R/W, /DTACK, XRDY und der System-Takt von 7.16 MHz. Ein besonderes Augenmerk sollte den Signalen XRDY und /DTACK gelten. Mit dem Signal /DTACK wird dem MC68000 mitgeteilt, ob die Datenübertragung beendet ist. Da alle Bausteine des Systems schnell genug sind, läuft der MC68000-Prozessor nahezu ungebremst. Wird eine Verlängerung des /DTACK-Signals benötigt, bei z.B. langsamen RAM-Chips, so kann mittels XRDY das /DTACK-Signal verzögert werden. Somit legt der Prozessor Wartezyklen ein. Die Taktzyklen des Systems werden durch die Kennung S0 bis S7 unterschieden. Abbildung Z 5.3.1-1 zeigt den Standard-Lesevorgang.

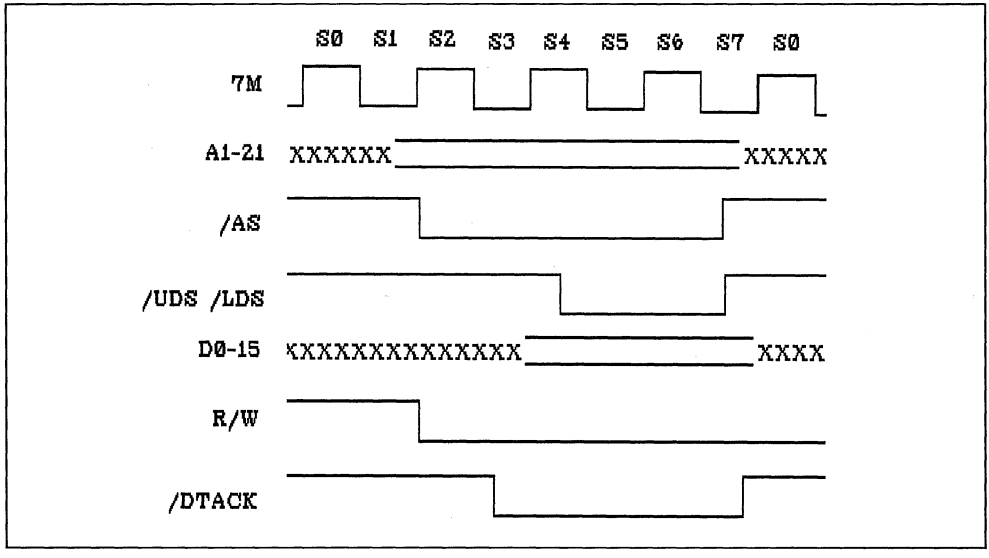


Z 5.3.1-1: *Der Standard-Lesevorgang beim Amiga mit Warteschleife*

Während S0 ist noch keine Aktivität bei diesen Leitungen zu erkennen. Bei S1 hingegen legt der MC68000 die Adreßleitungen auf den Adreßbus, von denen Daten gelesen werden sollen. Nachdem die Adressen auf dem Adreßbus eingeschwungen sind, wird mit /AS bei S2 die Gültigkeit der Adressen auf dem Adreßbus bestätigt. Gleichzeitig wird mit /LDS und /UDS die Art des Zugriffs auf den Datenbus signalisiert (hier Word-Zugriff). Ebenfalls wird R/W positiv, wodurch ein Lesezugriff signalisiert wird. Wird an dieser Stelle nun von einem externen Gerät innerhalb von 60 ns XRDY auf logisch low gelegt, verzögert sich das /DTACK-Signal. Es kann nicht mehr vor dem Zyklus S4 gesetzt werden. Somit legt der Prozessor, wo normalerweise die Zyklen S5 und S6 ablaufen, zwei Wartezyklen ein. Die Wartezyklen werden allgemein durch SW gekennzeichnet. In unserem Beispiel ist bei S5 der Wartezyklus zu Ende. Bei S6 werden vom MC68000 die Daten vom Datenbus gelesen, und bei S7 wird das Ende des Lesezyklus durch Zurücksetzen der Steuerleitungen signalisiert.

5.3.2: Der Standard-Schreibvorgang

Der Standard-Schreibvorgang ist sehr leicht an dem späten Setzen der /UDS- und /LDS-Signale, sowie dem langen Vorhandensein des Datenbusses zu erkennen. Rein theoretischen könnte der Datenbus schon nach /AS die Daten vom MC68000 enthalten. Jedoch werden die Daten erst ab dem Zyklus S4 (Zorro-Slot) gültig, weil der Buster bzw. die PAL's bei den 2000er eine Bus-Kollision überprüfen müssen.

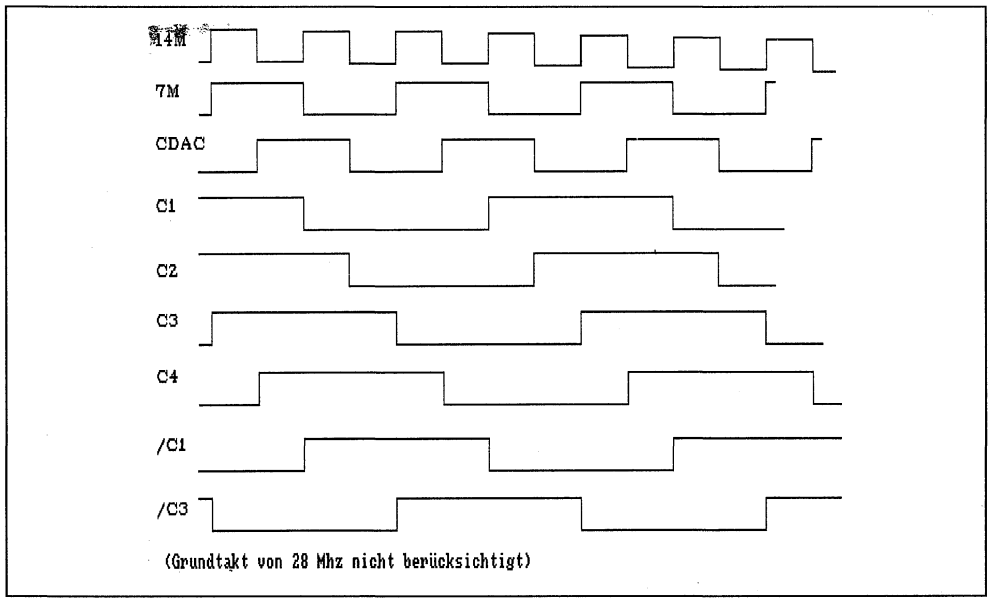


Z 5.3.2-1: *Der Standard-Schreibvorgang ohne Warteschleife*

Für eine Verzögerung des /DTACK-Signals gilt hier dasselbe wie beim Standard-Lesevorgang. Wie beim Lesevorgang schwingen bei S1 die Adreßleitungen ein und werden durch S2 mit /AS gültig. Gleichzeitig wird das R/W-Signal auf logisch low gelegt, wodurch der Schreibzyklus signalisiert wird. Die Signalisierung der Zugriffsart durch /UDS und /LDS findet hier erst ab S4 statt. Die Datenleitungen sind, wie schon erwähnt, erst ab S4 gültig.

5.3.3: Die Takte des Amiga

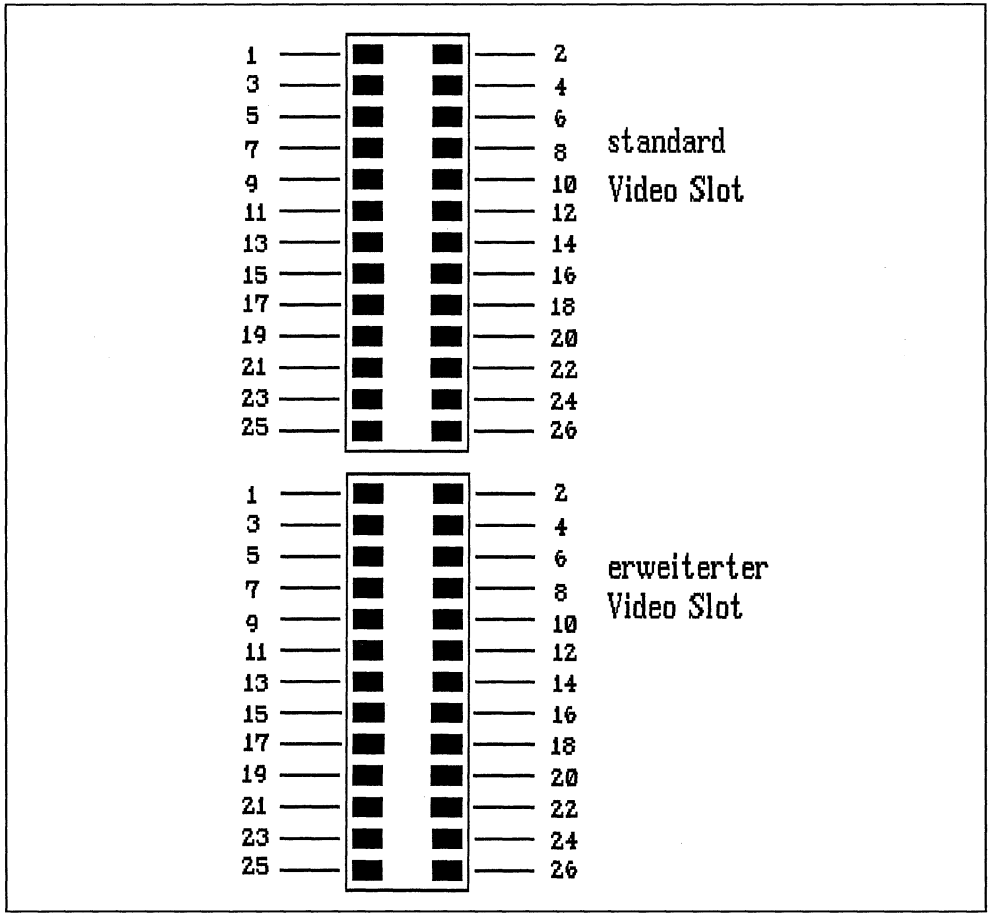
Insgesamt sind in dem Amiga 10 verschiedene Takte vorhanden. Alle Takte unter 28 MHz werden durch logisches Verknüpfen mehrerer *Datenflipflops* beim Amiga 1000 und A2000 erzeugt. Beim Amiga 500 und B2000 übernimmt *Agnus* die Teilung der Takte. Die wichtigsten *Takte* des Systems sind /C1 und /C3. Aus diesen Takten kann mit einem *Äquivalenzgatter* (XNOR) der Systemtakt von ca. 7 MHz erzeugt werden. Also sind alle Takte, die aus /C1 und /C3 resultieren, miteinander synchronisiert. Da die Signale /C1 und /C3 so überaus wichtig sind, liegen sie auch an jedem Slot an.



Z 5.3.3-1: *Die Systemtakte des Amiga*

5.4: Die Video-Slots des Amiga A/B2000

Um Anwendungen wie das Genlock-Interface oder PAL-Modulatoren intern im Amiga 2000 unterzubringen, haben die Entwickler den Amiga 2000 mit einem *Video-Slot* versehen. Dort sind alle wichtigen Signale zusammengeführt, die für eine Video-Darstellung benötigt werden. Da ein Genlock-Interface unter Umständen die parallele Schnittstelle zur Steuerung benötigt, wurde beim Amiga B2000 ein weiterer Slot zum vorhandenen Video-Slot hinzugefügt. Beide Stecker sind je ein 36-Pin-Verbinder, die auch für den 16-Bit-AT-Slot benötigt werden. Die Numerierung des Slot ist sehr einfach: auf der linken Seite befinden sich alle ungeraden, auf der rechten Seite alle geraden Pin-Zahlen.



Z 5.4-1: *Die Pinbelegung des Video-Slots beim Amiga B2000*

5.4.1: Die Pinbelegung des Standard-Video-Slots

Gemeinsam beim Amiga 2000 und Amiga B2000 ist der Standard-Video-Slot. Hier sind alle wichtigen Video-Signale herausgeführt.

Pin	Funktion	A2000	B 2000
1	nicht belegt	×	×
2	nicht belegt	×	×
3	Ton links	×	×
4	Ton rechts	×	×
5	nicht belegt	×	×
6	+5V DC	×	×
7	Analog Rot	×	×
8	+5V DC	×	×
9	Video Ground	×	×
10	+12V DC	×	×
11	Analog Grün	×	×
12	Video Ground	×	×
13	Video Ground	×	×
14	/CSYNC	×	×
15	Analog blau	×	×
16	/XCLKEN	×	×
17	Video Ground	×	×
18	BURST	×	×
19	/C4	×	×
20	Video Ground	×	×
21	Video Ground	×	×
22	/HSYNC	×	×
23	DI	×	×
24	Video Ground	×	×
25	DB	×	×

Pin	Funktion	A2000	B 2000
26	/VSYNC	×	×
27	DG	×	×
28	COMP SYNC	×	×
29	DR	×	×
30	/PIXELSW	×	×
31	−5V DC	×	×
32	Video Ground	×	×
33	XCLK	×	×
34	/C1	×	×
35	nicht belegt	×	×
36	nicht belegt	×	×

5.4.2: Die Signale des Standard-Video-Slots

Um flexibler zu sein, und da man nur einen RGB-Stecker herausführen wollte, fügte man bei der 2000er-Serie einen Video-Slot ein. Fast alle Signale, die am Standard-Video-Slot der Amiga 2000er vorhanden sind, liegen ebenfalls am RGB-Port des Amiga 1000 und 500.

Die Spannungsversorgung des Standard-Video-Slots:

An dem Standard-Video-Slot stehen, wie auch beim 86-Pin-Slot, drei verschiedene Spannungen, die sich auf GND (Ground – Masse) beziehen, zur Verfügung. Die maximale Belastung im Vergleich zum 86-Pin-Slot bzw. 100-Pin-Slot ist gleich. Die totale Belastung bezieht sich hier wiederum auf das ganze Amiga-System:

+12V – Dies ist die höchste Spannung, die an dem Standard-Video-Slot anliegt. Sie wird für z.B. RGB-Encoder-Bausteine bei PAL-Karten oder ein Genlock-Interface benötigt:
8 Ampere total

+5V – Diese Spannung wird als Hauptversorgungsspannung bezeichnet, da sie bei allen Chips in den Amiga's benötigt wird:
maximal 2 Ampere

−5V – Bei dieser negativen Versorgungsspannung ist die Belastbarkeit bedeutend geringer als bei einer positiven Spannung:
ca. 0.3 Ampere total

Die Video-Signale des Standard-Video-Slots:

Dies sind die digitalen Werte der Video-Darstellung:

DI = Digital Intensität
DR = Digital Rot
DG = Digital Grün
DB = Digital Blau

Mit diesen vier Werten können maximal 16 Farben erzielt werden. Sie werden benötigt, wenn ein IBM-Farbmonitor an den Amiga angeschlossen werden soll.

Analog Rot Grün Blau	Mit diesen analogen RGB-Werten, die eine max. Spannung von 0.7 Volt besitzen, können alle Farben des Amiga auf einem Monitor dargestellt werden, der einen analog-RGB-Eingang hat.
/HSYNC	Dies ist das bidirektionale horizontale Synchron-Signal der Video-Darstellung. Wird hier ein externes /HSYNC-Signal angelegt, versucht der Custom-Chip Agnus dieses Signal mit seinem /HSYNC-Signal zu synchronisieren.
/VSYNC	Dies ist das bidirektionale vertikale Synchron-Signal der Video-Darstellung. Wird hier ein externes /VSYNC-Signal angelegt, versucht der Custom-Chip Agnus dieses Signal mit seinem /VSYNC-Signal zu synchronisieren.
/CSYNC	/CSYNC ist das digitale Gemisch des /HSYNC- und /VSYNC-Signals.
COMPSYNC	COMPSYNC ist das gepufferte /CSYNC-Signal.
BURST	Dieses Signal ist das Farbträger-Signal der NTSC-Norm. Um ein PAL-Burst zu erhalten, muß dieses Signal mit 1.25 multipliziert werden (3.55 * 1.25 = 4.433 MHz).
/PIXELSW	Hintergrundfarben-Indikator.
Ton links rechts	Dies sind die Audio Signale des linken und rechten Kanals.

Die System-Clocks des Standard-Video-Slots:

/C1	Dies ist ein Takt mit der Frequenz von 3.58 MHz, synchronisiert zu der fallenden Flanke des Systemtaktes von 7.16 MHz.
/C4	Für NTSC-Geräte ist dies ein Takt von 3.58 MHz, synchronisiert zur ansteigenden Flanke des CDAC-Taktes von 7.16 MHz. Bei PAL-Gerä-

ten ist diese Frequnz 3.55 MHz, bezogen auf ein CDAC-Signal von 7.09 MHz.

XCLCK — Hier kann ein externer Grund-Systemtakt von ca. 28.64 MHz angelegt werden. Dies dient zur Synchronisierung des Amiga mit externen Geräten.

/XCLKEN — /XCLKEN schaltet die interne Generierung des Grund-Systemtaktes ab und ermöglicht so die Speisung des Systems mit dem XCLK-Takt.

5.4.3: Die Pinbelegung des erweiterten Video-Slots

Beim Amiga B2000 wurde der Standard-Video-Slot um einen weiteren 36-Pin-Stecker erweitert. Er dient überwiegend zur Steuerung von Video-Karten über die parallele Schnittstelle.

Pin	Funktion	A2000	B 2000
1	Ground		×
2	R0		×
3	R1		×
4	R2		×
5	Ground		×
6	G0		×
7	G1		×
8	G2		×
9	Ground		×
10	B1		×
11	B2		×
12	Ground		×
13	CompVideo		×
14	TBASE		×
15	CDAC		×
16	POUT		×
17	/C3		×

Pin	Funktion	A2000	B 2000
18	BUSY		×
19	/LPEN		×
20	/ACK		×
21	SEL		×
22	Ground		×
23	PD0		×
24	PD1		×
25	PD2		×
26	PD3		×
27	PD4		×
28	PD5		×
29	PD6		×
30	PD7		×
31	/LED		×
32	Ground		×
33	Raw Ton links		×
34	Ton Ground		×
35	Raw Ton rechts		×
36	Ton Ground		×

5.4.4: Die Signale des erweiterten Video-Slots

Der erweiterte Video-Slot enthält überwiegend Signale der parallelen Schnittstelle. Diese werden benötigt z.B. für die Steuerung der Hintergrundfarbe des Genlock-Videos oder anderer Geräte.

Die Spannungsversorgung des erweiterten Video-Slots:

Alle Spannungen, die benötigt werden stehen bei dem Standard-Video-Slot zur Verfügung. Somit wollte man hier keine weitere Spannung zuführen. Der Bezugspunkt Ground (Masse) der Spannung steht jedoch zur Verfügung. Hier wird unterschieden zwischen dem Bezugspunkt Audio-GND, Video-GND und Digital-GND. Die Einhaltung der Bezugspunkte je nach Schaltung ist erforderlich.

Die Video-Signale des erweiterten Video-Slots:

Rot 0–2 Grün 0–2 Blau 0–2	Dies sind die dekodierten 8 Bit der digitalen Video-Darstellung.
CompVideo	CompVideo ist das monochrome Composite-Video-Signal, auch BAS-Signal genannt.
/LPEN	Über diesen Pin ist der Anschluß eines Light-Pen möglich. Dies ist der direkte Light-Pen-Eingang von AGNUS. Wird dieses Signal logisch low, so ermittelt AGNUS die aktuelle Rasterstrahlposition, an der der Lichstift an den Monitor gehalten wurde.
RAW Ton links rechts	Dies sind die Audio-Signale des linken und rechten Kanals, bevor sie über einen Line-Filter gehen.
/LED	Über diesen Pin lassen sich die zweipoligen Low-Pass-Filter der Audio-Hardware abschalten.

Die Signale der parallelen Schnittstelle:

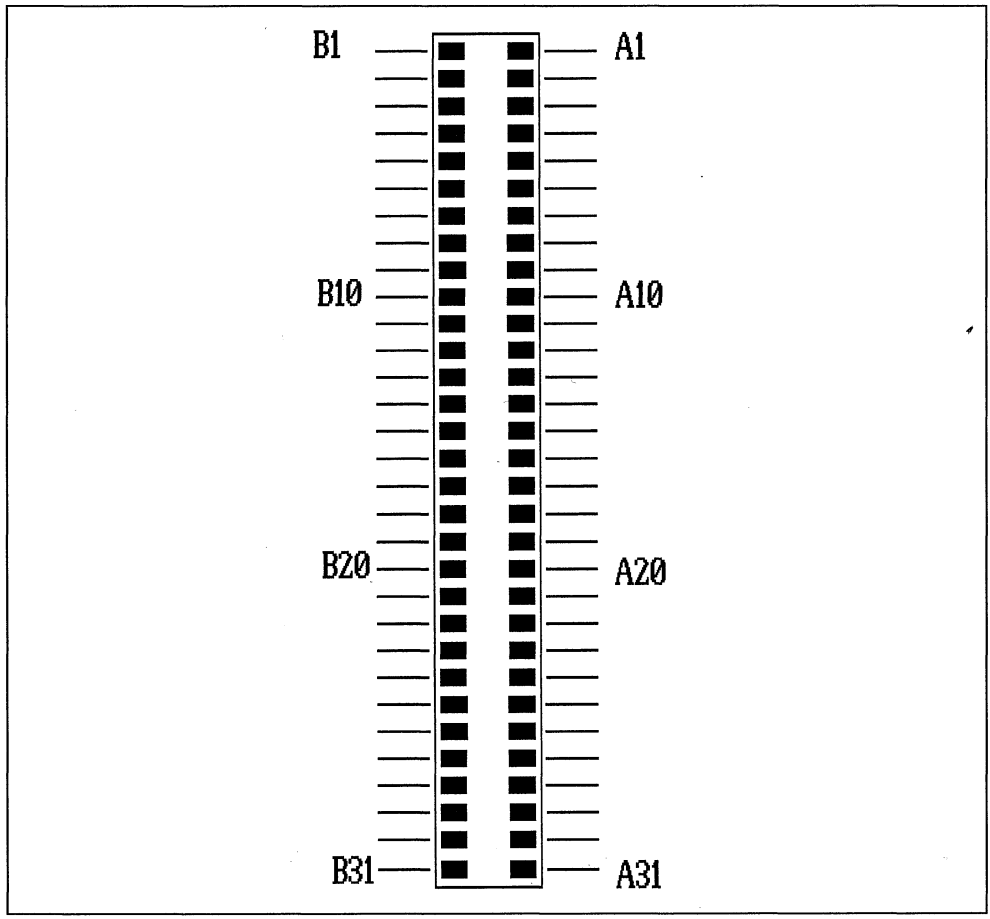
PD0–PD7	Dies sind die 8–Bit des bidirektionalen Parallel-Ports. Üblicherweise werden diese 8–Bit zum Steuern eines Centronic-Interfaces benutzt. Für Video-Anwendungen wie Genlock-Interface wird hier die Ausblend-Farbe gesteuert.
/ACK	Mit diesem Acknowledge (/ACK) kann über einen der CIA-8520-Bausteine ein Level-2-Interrupt ausgelöst werden.
BUSY	Dies ist das Beschäftigt-Signal des Druckers. Es kann hier z.B. das Beschäftigtsein eines Digitalisieres anzeigen.
POUT	POUT entspricht bei Anwendungen einer parallelen Schnittstelle dem Papier-Ende-Signal.
SEL	Select-Signal an Peripherie senden.

Die System-Clocks des erweiterten Video-Slots:

/C3	/C3 ist ein Takt von 3.58 MHz bei NTSC und 3.55 MHz bei PAL, der synchronisiert ist zu der steigenden Flanke des Systemtaktes.
CDAC	CDAC ist ein 7.16 MHz Takt bei NTSC und 7.09 MHz bei PAL, der in bezug auf den Systemtakt um 90 Grad nacheilt.

5.5: Der PC-Slot

Ein weiterer stark verbreiteter Slot bei der Amiga-Serie ist der *PC-Slot*. Er ist im Amiga 2000 und B2000 sowie im *SideCar* zu finden. Dieser Slot ist 100prozentig kompatibel zum 62poligen PC/XT-Slot von IBM- bzw. IBM-kompatiblen Rechnern. Durch ihn wird in den Amigas eine Kompatibilität erzielt, die bei entsprechender Zusatzkarte (PC- oder AT-Emulator) den Betrieb von z.B. einer HardDisk für PC-Rechner im Amiga erlaubt. Da er sozusagen schon zur Amiga-Familie zählt, werden wir nun auch näher auf diesen Slot eingehen.



Z 5.5-1: *Pinnumerierung des PC-Slots*

5.5.1: Die Pinbelegung des PC-Slots

Der Original-PC-Slot besitzt einen Standard-Stecker mit 62 Pins. Er wird auch als 8-Bit-Slot bezeichnet, da er einen 8-Bit-Datenbus besitzt. Bei der Durchnumerierung der Pins haben wir uns an den IBM-Standard gehalten. Alle Pins mit A liegen auf der rechten Seite, alle Pins mit B liegen auf der linken Seite des Slots. In der nun folgenden Tabelle steht I/O für Ein-/Ausgang (Input/Output):

Pin	Funktion	I/O
A1	/IO CH CK	I
A2	SD07	I/O
A3	SD06	I/O
A4	SD05	I/O
A5	SD04	I/O
A6	SD03	I/O
A7	SD02	I/O
A8	SD01	I/O
A9	SD00	I/O
A10	IO CH RDY	I
A11	AEN	O
A12	SA19	I/O
A13	SA18	I/O
A14	SA17	I/O
A15	SA16	I/O
A16	SA15	I/O
A17	SA14	I/O
A18	SA13	I/O
A19	SA12	I/O
A20	SA11	I/O
A21	SA10	I/O
A22	SA09	I/O
A23	SA08	I/O

Pin	Funktion	I/O
A24	SA07	I/O
A25	SA06	I/O
A26	SA05	I/O
A27	SA04	I/O
A28	SA03	I/O
A29	SA02	I/O
A30	SA01	I/O
A31	SA00	I/O
B1	GND	–
B2	Reset Drv	–
B3	+ 5V DC	–
B4	IRQ 9	I
B5	– 5V DC	–
B6	DRQ 2	I
B7	– 12V DC	–
B8	OWS	I
B9	+ 12V DC	–
B 10	GND	–
B 11	/SMEMW	O
B 12	/SMEMR	O
B 13	/IOW	I/O
B 14	/IOR	I/O
B 15	/DACK3	O
B 16	DRQ	I
B 17	/DACK 1	O
B 18	DRQ 1	I
B 19	/REFRESH	I/O
B 20	CLK	O

Pin	Funktion	I/O
B 21	IRQ 7	I
B 22	IRQ 6	I
B 23	IRQ 5	I
B 24	IRQ 4	I
B 25	IRQ 3	I
B 26	/DACK 2	O
B 27	T/C	O
B 28	BALE	O
B 29	+ 5V DC	–
B 30	OSC	O
B 31	GND	–

5.5.2: Die Signale des PC-Slots

Der PC-Slot enthält die Grundsignale für alle PC-Karten. Neben verschiedenen systemspezifischen Signalen enthält dieser Slot auch Standard-Signale, wie Adreß- und Datenleitungen. Die Spannungsversorgung der Slots ist gleichzusetzen mit den Amiga-Zorro- und 86-Pin-Slots.

Die Spannungsversorgung des PC-Slots:

+12V – Dies ist die höchste Spannung, die an dem PC-Slot anliegt.
8 Ampere total

−12V – Dies ist die negativ höchste Spannung, die der PC-Slot zur Verfügung stellt.
0.3 Ampere total

+5V – Hauptversorgungsspannung.
2 bis 4 Ampere pro Slot

−5V – Bei dieser negativen Versorgungsspannung ist die Belastbarkeit bedeutend geringer als bei einer positiven Spannung:
ca. 0.3 Ampere total beim Amiga 2000

Die PC-System-Signale des PC-Slots:

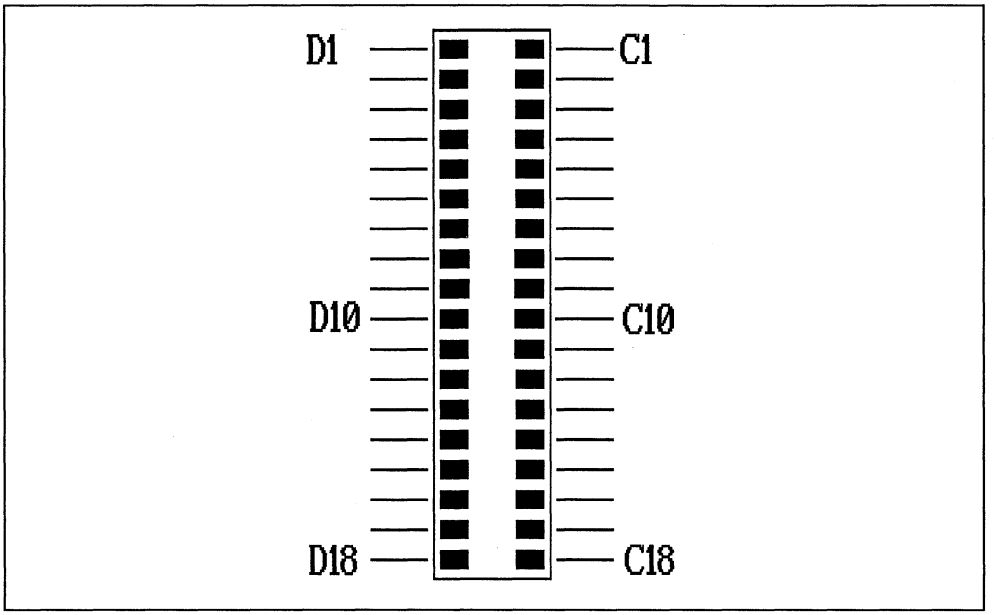
SA00–SA19	Adreßleitungen des SBusses. Als SBus bezeichnet man die Adreßleitungen, die die PC-Slots miteinander verbinden. Die Adreßleitungen werden erst mit BALE high freigegeben.
SD0–SD7	Dies sind die Datenleitungen auf dem SBus (s.o.).
/IO CH CK	Dieses Signal testet den IO-Bus und erstellt einen NMI (Non-Maskable-Interrupt) bei einem Fehler.
/IOCH RDY	Dieses Signal testet den IO-Bus auf Verfügbarkeit (Active low).
RESETDRV	Wird benutzt, um das System zu initialisieren.
IRQ 3–7 IRQ 9	Interrupt-Leitungen.
DRQ 1–3	DMA-Request-Leitungen. Teilt eine DMA-Anfrage mit.
/DACK 1–3	Bestätigt eine DMA-Anfrage.
T/C	Wird verwendet, um einen abgeschlossenen DMA-Zugriff zu signalisieren.
BALE	Zeigt an, ob ein Zugriff auf den Adreßbus derzeit möglich ist.
/SMEMW	Dieses Signal kennzeichnet, daß die Daten, die sich momentan auf dem Datenbus befinden, geschrieben werden sollen.
/SMEMR	Dieses Signal kennzeichnet, daß Daten aus der Adresse gelesen werden sollen, die sich auf dem Adreßbus befindet.
/IOW	Signalisiert der Peripherie, daß sie Daten vom Datenbus lesen soll.
/IOR	Signalisiert der Peripherie, daß sie Daten auf den Datenbus schreiben soll.
/REFRESH	Signalisiert, daß ein Refresh-Zyklus bevorsteht.

System-Takte des PC-Slots:

CLK	System-Takt (4,77 MHz bei PC-Karte).
OSC	Video-Takt mit 14,31318 MHz.

5.6: Der AT-Slot

Neben dem PC-Slot des Amiga 2000, B2000 und SideCar, ist bei der Amiga 2000er-Serie ein 16-Bit-Slot vorhanden, der mit dem PC-Slot zusammen einen AT-Slot ergibt. Er ermöglicht den Betrieb eines AT-Emulators und AT-spezieller Karten sowie beispielsweise großer RAM-Aufrüstungen. Dieser erweiterte Slot mit 32 Pin ist 100prozentig kompatibel zu den IBM-AT-Slots.



Z 5.6-1: *Der AT-Slot*

5.6.1: Die Pinbelegung des AT-Slots

Der originale AT-Slot besitzt einen Standard-Stecker von 36 Pins. Er wird allgemein auch als 16-Bit-Slot bezeichnet, da er einen *16-Bit-Datenbus* besitzt. Bei der Durchnumerierung der Pins haben wir uns, wie bei dem PC-Slot, an den IBM-Standard gehalten. Alle Pins mit C liegen auf der rechten Seite, alle Pins mit D liegen auf der linken Seite des Slots.

Pin	Funktion	I/O
C 1	SBHE	I/O
C 2	LA23	I/O
C 3	LA22	I/O
C 4	LA21	I/O
C 5	LA20	I/O
C 6	LA19	I/O
C 7	LA18	I/O
C 8	LA17	I/O
C 9	/MEMR	I/O
C 10	/MEMW	I/O
C 11	SD08	I/O
C 12	SD09	I/O
C 13	SD10	I/O
C 14	SD11	I/O
C 15	SD12	I/O
C 16	SD13	I/O
C 17	SD14	I/O
C 18	SD15	I/O
D 1	/MEM CS 16	I
D 2	/IO CS 16	I
D 3	IRQ10	I
D 4	IRQ11	I
D 5	IRQ12	I

Pin	Funktion	I/O
D 6	IRQ15	I
D 7	IRQ14	I
D 8	/DACK0	O
D 9	DRQ0	I
D 10	/DACK5	O
D 11	DRQ5	I
D 12	/DACK6	O
D 13	DRQ6	I
D 14	/DACK7	O
D 15	DRQ7	I
D 16	+ 5V DC	–
D 17	/MASTER	I
D 18	GND	–

5.6.2: Die Signale des AT-Slots

Die Spannungsversorgung des AT-Slots:

+5V – Hauptversorgungsspannung.
2 bis 4 Ampere pro Slot

Die Signale des Slots:

LA17–LA23 Erweiterte Adreßleitungen für 24-Bit-Adreßbus. Diese Leitungen sind frei, wenn BALE high ist.

SD08–SD15 Dies sind die erweiterten Datenleitungen für den 16-Bit-Datenbus.

IRQ10–14 Erweiterte Interruptleitungen.

/DACK0
/DACK5–7 DMA-Anfragebestätigung.

DRQ0
DRQ5–7 Leitungen zur DMA-Anfrage.

/SBHE Zeigt an, ob eine 8-Bit- oder 16-Bit-Datenübertragung vorliegt.

/MEMR Zeigt an, daß aus der Adresse, die auf dem Adreßbus anliegt, Daten auf den Datenbus gelegt werden sollen.

/MEMW Zeigt an, das Daten vom Datenpuffer in die Adresse, die auf dem Adreßbus anliegt, übernommen werden sollen.

/MEMCS16 Wird benutzt, um anzuzeigen, daß es sich bei der momentanen Datenübertragung um eine 16-Bit-Übertragung handelt.

/IOCS16 Kennzeichnet die Datenübertragung mit der Peripherie als 16-Bit-Übertragung.

/MASTER Kennzeichnet, daß die Kontrolle über die Speicherverwaltung übernommen werden soll.

Kapitel 6

Die Autokonfiguration

Die *Erweiterungskarten* für die Amiga-Rechner werden automatisch vom System erkannt und eingebunden. Dies gilt nicht nur für die 2000er, sondern auch schon für den Amiga 1000 und 500. Damit dies aber einwandfrei geschehen kann, benötigt das System festgelegte Informationen über die Erweiterung. Diese Informationen stehen in einem *PAL* auf der Erweiterungskarte. Nach dem Einschalten des Rechners werden die Erweiterungsplatinen zurückgesetzt. In diesem Zustand korrespondieren alle Karten mit dem 64Kbyte Konfigurations-Rambereich, der ab $E80000 beginnt, wenn *CONFIGIN* aktiv ist. Dieses Signal wird nacheinander bei allen Karten aktiv gesetzt, so daß die Karten durchlaufend konfiguriert werden. Bei den Amiga 5000- und 1000-Rechnern ist dies nur für den 86-Pin-Expansionsslot nötig, beim Amiga 2000 allerdings müssen neben diesem Slot noch die 5 100-Pin-Slots konfiguriert werden. Bei diesen Rechnern wird von rechts nach links konfiguriert, d.h. zuerst wird der 86-Pin-Slot eingebunden, dann die 5 100-Pin-Slots von rechts nach links.

Bei der Konfiguration liest die CPU *Nibbles* der *Identifikationsdaten* über die Datenleitungen D12 bis D15 aus dem PAL der Erweiterungskarte. Aus diesen Daten kann dann erkannt werden, wieviel Speicherplatz die Karte benötigt, bzw. wieviel sie bereitstellt, falls es eine RAM-Erweiterungskarte ist. Falls die Karte Speicherplatz benötigt, so konfiguriert das System die Karte auf den bereitgestellten Speicherbereich hin. Der kleinste Speicherbereich, auf den eine Erweiterungskarte konfiguriert werden kann, ist 64 Kbyte, der größte ist 8 Mbyte. Anschließend läßt das *CONFIGOUT*-Signal dieses Slots das CONFIGIN-Signal des nächsten Slots aktiv werden.

Die Karten müssen so ausgelegt sein, daß sie mit ihrem Speicherbereich in Einklang stehen. Damit ist gemeint, daß eine Karte, die 64-Kbyte-benötigt, immer mit 64-Kbyte-Grenzen bei der Adreßdekodierung arbeitet. Eine Ausnahme bilden 4- und 8-Mbyte Karten. 4-Mbyte-Karten dürfen zusätzlich bei den Startadressen $200000 und $600000 beginnen und 8-Mbyte-Karten bei der Adresse $200000. Dies gilt aber nicht nur für die Karten, die Speicherplatz benötigen, sondern auch für die Karten, die Speicherplatz bereitstellen.

Hier nun die Tabelle der Nibbles, die gesetzt werden müssen, d.h. in einem PAL festgelegt sein müssen. Dabei müssen alle Nibbles außer 00, 02, 40 und 42 invertiert werden:

Nibble 00/02	76	Kartentyp: 00,01,10 – reserviert 11 – Erweiterungskarte
	5	1 = In MemFreeList eintragen 0 = nicht eintragen
	4	Optionelles ROM vorhanden
	3	Gibt an, daß die nächste Karte zu dieser hinzu gehört
	210	Speichergröße: 000 – 8 Mbyte 100 – 512Kbyte 001 – 64 Kybte 101– 1 Mbyte 010 – 128 Kbyte 110 – 2 Mbyte 011 – 256 Kbyte 111 – 4 Mbyte
Nibble 04/06	76543210	Produktnummer. Diese Nummer kann vom Hersteller frei gewählt werden, um die Karte beispielsweise von spezieller Software identifizieren zu können.
Nibble 08/0A	Diese Nibble müssen wie angegeben gesetzt werden.	
	7	0 – kein best. RAM-Bereich
	–	1 – Speicherbereich muß im 8-Mbyte-Bereich sein
	6	0 – Karte kann abgeschaltet werden
		1 – Karte kann nicht abgeschaltet werden
	543210	müssen 0 sein
Nibble 0C/0E	Diese Nibble müssen gleich 0 sein	
Nibble 10/12	Highbyte der Herstellernummer	
Nibble 14/16	Lowbyte der Herstellernummer Anhand dieser Nummer installiert die Config-Software Driver für diese Karte.	
Nibble 18/1A Nibble 1C/1E Nibble 20/22 Nibble 24/26	Optionelle Seriennummer (MSB) (LSB)	
Nibble 28/2A	Falls ein ROM vorhanden ist und daraufhin Bit 4 im 00/02 gesetzt ist,	
Nibble 2 C/E	muß hier der Zeiger auf dieses ROM stehen (28/2A = Highbyte).	
Nibble 30/32	Reserviert, müssen im Lesezugriff gleich 0 sein, und im Schreibzugriff muß das Basisadreßregister zurückgesetzt werden.	
Nibble 34/36	reserviert und müssen gleich 0 sein.	
	Nibble 38/3A	
	Nibble 3C/3E	

Nibble 40/42 Optionales Kontroll-Statusregister

Im Lesezugriff:

7 Interrupt-Request	3 keine Bedeutung
6 Interrupt 7 steht bevor	2 muß gleich 0 sein
5 Interrupt 6 steht bevor	1 keine Bedeutung
4 Interrupt 2 steht bevor	0 Interrupt aktivieren

Im Schreibzugriff:

7 frei definierbar	3 frei definierbar
6 frei definierbar	2 lokaler Reset
5 frei definierbar	1 frei definierbar
4 frei definierbar	0 Interrupt aktivieren

Nibble 44/46 reserviert
Im Schreibzugriff nicht definiert
Im Lesezugriff muß Nibble gleich 0 sein

Nibble 48/4A Basis-Adreßregister, nur Schreibzugriff.
Diese Nibbles werden mit den Adreßregistern A16 bis A23 verglichen, um die Basisadresse der Karte zu ermitteln.

Nibble 4C/4E Falls auf dieses Nibble im Schreibzugriff zugegriffen wird, muß die Karte abgeschaltet werden, falls Bit 6 in 08/0A gesetzt ist.

Nibble 50/52 sind reserviert und müssen gleich 0 sein.
Nibble 54/56
Nibble 58/5A
Nibble 5C/5E
Nibble 60/62
Nibble 64/66
Nibble 68/6A
Nibble 6C/6E
Nibble 70/72
Nibble 74/76
Nibble 78/7A
Nibble 7C/7E

Wenn also die Nibbles mit 0 angegeben sind, so müssen sie in Wirklichkeit gleich $FF sein, da sie ja invertiert werden müssen, wie oben schon erwähnt. Die Invertierung wurde deshalb gewählt, da somit Low-Aktive-PAL's verwendet werden können, welche preiswerter sind.

6.1: Das Hardware-Beispiel

Hier wollen wir nun ein Beispiel für eine Erweiterungsplatine geben. Diese Karte stellt dem System 16 Kbyte an Speicher zur Verfügung. Da aber nur mindestens 64-Kbyte verwaltet werden können, wird diese Karte dann als 64 Kbyte-Karte in das System eingebunden. Um die folgende Beschreibung verfolgen zu können, sollten Sie die Abbildung Z 6.1-2 (siehe Anhang) zur Hand nehmen.

Das Herz der Karte sind die Bausteine U1 (Adreßregister), U2 (Adressen-Komparator) und U3 (Identifikations-PAL und Kontroll-PAL). Nach dem Reset ist CONFIGOUTinaktiv, damit bei der Konfiguration nicht automatisch zur nächsten Karte weitergesprungen wird. Solange CONFIGIN inaktiv ist, ist auch SLAVE inaktiv, was durch U11 gesteuert wird. Dadurch muß auch *BOARD_SEL* inaktiv bleiben. Das bedeutet, das die Karte vollkommen inaktiv ist, da *BOARD_SEL* ein Eingang zu U3, dem Kontroll-PAL ist, und solange dieser low ist, bleiben auch alle PAL-Ausgänge inaktiv.

Sobald CONFIGIN aktiviert wird, beginnt für die Karte die Konfigurationsphase. Wir wollen hier nur die vier wichtigsten Nibbles aufführen, die in der Konfigurationphase vom System benötigt werden, und sie unserer Karte entsprechend setzen:

Nibble 00/02 11000001
Nibble 04/06 11111001
Nibble 10/12 11111110
Nibble 40/42 00000000

Nun invertieren wir alle Nibbles, außer 00/02 und 40/42:

Nibble 00/02 11000001
Nibble 04/06 00000110
Nibble 10/12 00000001
Nibble 40/42 00000000

Nun die Nibbles im einzelnen:

Nibble 00/02
- 76 muß auf 11 gesetzt werden
- 5 0= nicht in MemFreeList eintragen, da zwar 64 Kbyte eingetragen würden, aber nur 16 Kbyte vorhanden wären.
- 4 0 = Kein ROM vorhanden
- 3 0 = Keine weiteren Karten vorhanden, die physikalisch zu dieser gehören.
- 210 001 = 64 Kbyte Karte

Nibble 04/06 = 00000110 = 6 Produktnummer

Nibble 10/12 = 00000001 = Highbyte der Herstellernummer
Nibble 14/16 = 00000000 = Lowbyte der Herstellernummer

Nibble 40/42 = 00000000 = Die Karte erzeugt keine Interrupts.
Das fertige PAL-Programm für diese Karte zeigt die Abbildung Z 6.1-1.

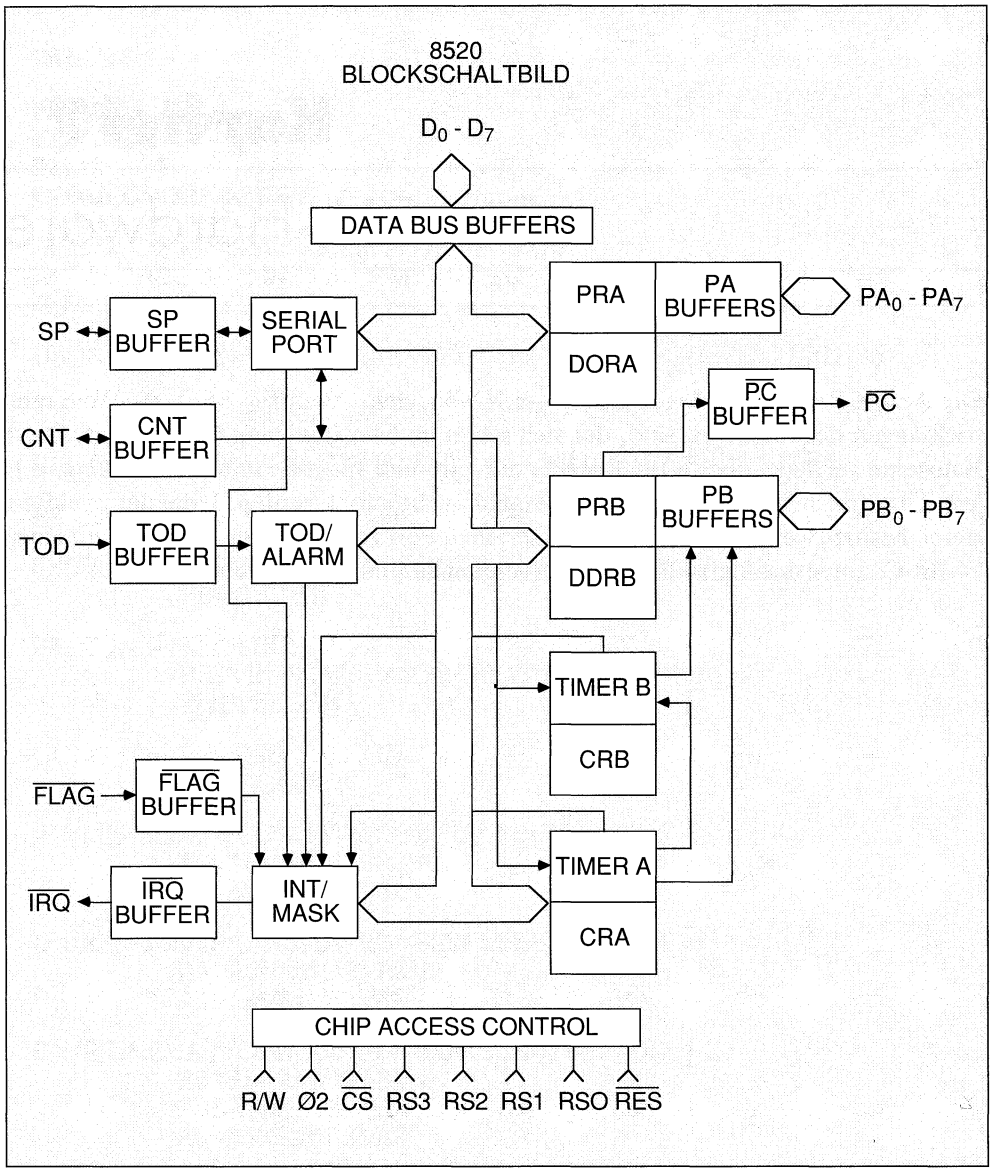
```
TABLE 3-4

PAL20L10
TESTRAM
9-11-85
COMMODORE-AMIGA

/ASQ/ASQQ RD/BDSEL/BERR A6 A5 A4 A3 A2
A1 GND /RES BD12 BD13 BD14 BD15/PRECON / CONOUT /SHUTUP
/RAMOE /WP /DBOE VCC

DBOE    = /RES*BDSEL*/BERR*/SHUTUP*/RD +      ;WRITES TURN ON
                                               EARLY
          /RES*BDSEL*/BERR*SHUTUP* RD*ASQ      ;ASQ DELAYS THE READ

WP      = /RES*ASQ*ASQQ*BDSEL*CONOUT*/SHUTUP*/RD*/BERR

RAMOE   = /RES*ASQ*RD*CONOUT*/BERR*BDSEL

SHUTUP  = /RES*BDSEL*/RD*ASQ/CONOUT*A6*/A5*/A4*A3*A2 +
          /RES*SHUTUP

PRECON  = /RES*SHUTUP +
          /RES*/RD*BDSEL*ASQQ*A6/A5*/AD*A3*/A2*/A1 +
          /RES*PRECON

CONOUT  = /RES*ASQ*PRECON +
          /RES*CONOUT

IF (/RES*BDSEL*/CONOUT*RD*/BERR*/SHUTUP)/BD15 =
          /A6*/A5*/A4*/A3*/A2*A1 +
           A6*/A5*/A4*/A3*/A2

IF (/RES*BDSEL*/CONOUT*RD*/BERR*/SHUTUP)/BD14 =
          /A6*/A5*/A4*/A3*/A2*A1 +
           A6*/A5*/A4*/A3*/A2

IF (/RES*BDSEL*/CONOUT*RD*/BERR*/SHUTUP)/BD13 =
          /A6*/A5*/A4*/A3*/A2 +
          /A6*/A5*/A4*/A3*/A2*A1 +
           A6*/A5*/A4*/A3*/A2

IF (/RES*BDSEL*/CONOUT*RD*/BERR*/SHUTUP)/BD12 =
          /A6*/A5*/A4*/A3*/A2*A1 +
          /A6*/A5*/A4*/A3*/A2*A1 +
           A6*/A5*/A4*/A3*/A2

DESCRIPTION
```

Z 6.1-1: *Das PAL-Programm für die 16-Kbyte-Erweiterungskarte*

Kapitel 7

Die CIA-Hardware

Die Amiga-Rechner verfügen über zwei IO-Bausteine vom Typ 8520, die Weiterentwicklungen des 6526-Typs sind, der sich schon im 64er jahrelang bewährt hat. Diese Bausteine verfügen über 8 Bit-Register zur Ein- und Ausgabe mit *Handshaking,* d.h. jede Übertragung muß durch eine Rückmeldung bestätigt werden. Jeder der zwei Bausteine besitzt zwei 16-Bit-Timer, einen seriellen Port zur Ein- oder Ausgabe und einen 24-Bit-Counter mit Alarm-Erkennung (vergleiche Bild 31 im Farbteil).



Z 7-1: *Das Blockschaltbild des 8520*

Um die beiden Bausteine im Amiga unterscheiden zu können, werden sie mit 8520-A und 8520-B bezeichnet. Jeder Baustein ist dabei für bestimmte Funktionen zuständig, der 8520-A beispielsweise vor allem für die Kontrollsignale der seriellen Schnittstelle und die Floppysteuerung.

Hier nun die Auflistung der 8520-Register:

8520-A		
PA0:	BUSY	Schnittstellen-Ausgang mit SP verbunden
PA1:	POUT	Schnittstellen-Eingang mit CNT verbunden
PA2:	SEL	Schnittstellen-Eingang
PA3:	-DSR	Schnittstellen-Eingang überTreiber MC1489A
PA4:	-CTS	Schnittstellen-Eingang überTreiber MC1489A
PA5:	-CD	Schnittstellen-Eingang überTreiber MC1489A
PA6:	-RTS	Schnittstellen-Ausgang überTreiber MC1488
PA7:	-DTR	Schnittstellen-Ausgang überTreiber MC1488
PB0:	-STEP	Floppy-Signal Step
PB1:	DIR	Floppy-Signal Direction
PB2:	-SIDE	Floppy-Signal Side-Select
PB3:	-SEL0	Floppy-Signal Drive0
PB4:	-SEL1	Floppy-Signal Drive1
PB5:	-SEL2	Floppy-Signal Drive2
PB6:	-SEL3	Floppy-Signal Drive3
PB7:	-MTR	Floppy-Signal Motor
SP:	BUSY	Schnittstellen-Ausgang mit PA0 verbunden
CNT:	POUT	Schnittstellen-Ausgang mit PA1 verbunden
-PC:	nicht benutzt	
-FLAG:	-INDEX	Floppy-Signal Disk Index
TOD:	-BHS	gepuffertes HSync zur Spritedarstellung
TIMER A:		frei
TIMER B:		wird zur Synchronisation des Blitters mit dem Rasterstrahl benutzt
8520-B		
PA0:	OVL	Memory-Overlay-Bit
PA1:	-LED	Power-LED-Kontrolle (Lowactive)
PA2:	-CHNG	Floppy-Signal Diskchange
PA3:	-WPRO	Floppy-Signal Writeprotected
PA4:	-TK0	Floppy-Signal Disktrack 0
PA5:	-RDY	Floppy-Signal Disk-Ready
PA6:	-JOY0	Port 0 Pin 6 (Feuerknopf)
PA7:	-JOY1	Port 1 Pin 6 (Feuerknopf)
PB0:	-P0	par. Schnittstelle Data 0
PB1:	-P1	par. Schnittstelle Data 1
PB2:	-P2	par. Schnittstelle Data 2
PB3:	-P3	par. Schnittstelle Data 3
PB4:	-P4	par. Schnittstelle Data 4
PB5:	-P5	par. Schnittstelle Data 5

PB6:	-P6	par. Schnittstelle Data 6
PB7:	-P7	par. Schnittstelle Data 7
SP:	KDAT	Keyboard-Data
CNT:	KCL	Keyboard-Clock
-PC:	-DRDY	par. Schnittstelle Eingang Strobe
-FLAG:	-ACK	par. Schnittstelle Ausgang Acknowledge
TOD:	TICK	50 Hz Signal für Real-Time-Clock
TIMER A:		für Tastatur belegt
TIMER B:		Virtual-Timer-Device für Multitasking und Interrupts

Welche Register mit welchen Speicheradressen korrespondieren, entnehmen Sie bitte dem Anhang D.

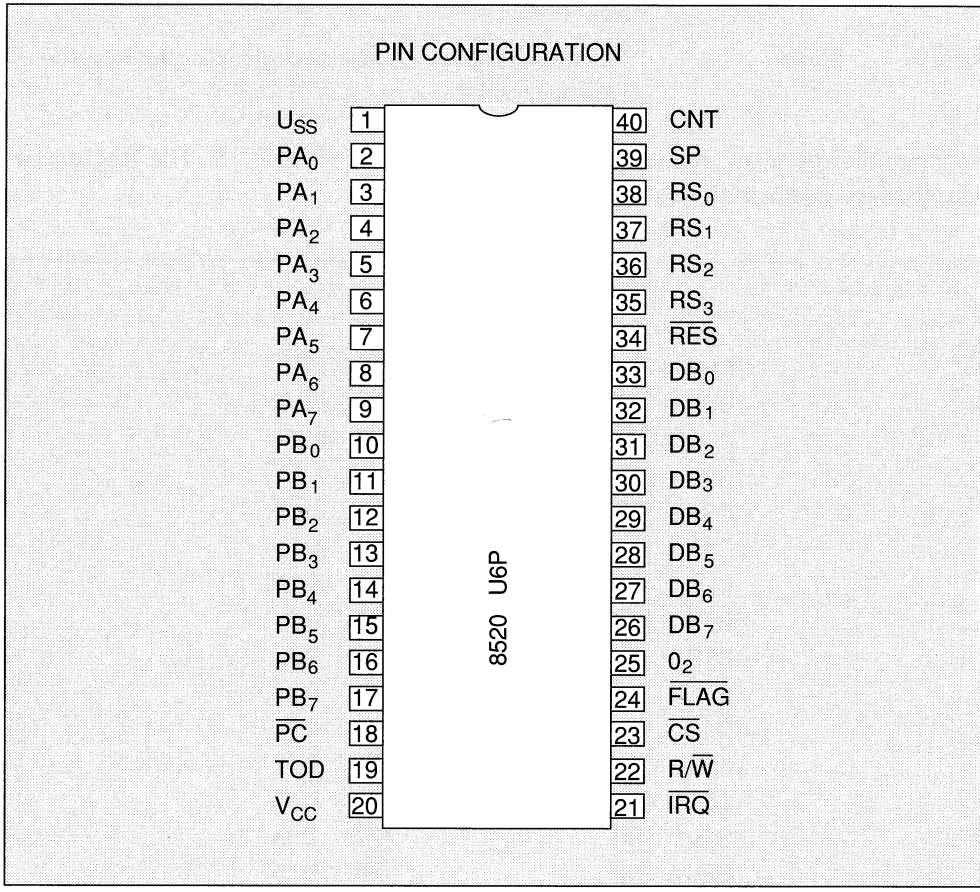
Alle Register lassen sich als Ein- oder als Ausgang programmieren. Dazu dienen die DDR *(Data-Direction-Register)*. Jedes Bit dieses Registers setzt ein bestimmtes Register als Ein- bzw. Ausgang. Bit 0 setzt PB0, Bit 1 setzt PB1 usw. Wenn ein Bit auf 1 gesetzt ist, so wird das korrespondierende Register als Ausgang gesetzt. Das Gleiche gilt auch für das Datenregister PR.

Ist ein Register als Ausgang geschaltet und auf 1 gesetzt, so liegt am angeschlossenen Pin eine Spannung von +5V an, wobei zwei *TTL-Lasten* betrieben werden können. Ist das Register auf 0 gesetzt, so liegt die Spannung nahe 0V.

```
1   ;*******************************
2   ;
3   ; LED-Demonstration
4   ; last update 10/03/88
5   ; von Frank Kremser und Jörg Koch
6   ; © Markt & Technik 1988
7   ;
8   ;*******************************
9   ;
10  ; Diese Demonstration läßt die Power-LED solange blinken, bis die
11  ; linke Mausetaste gedrückt wird.
12  ;
13  ;*******************************
14
15  start:   or.b     #2,$bfe001    ; LED hell
16           move.l   #20000,d0     ; Warteschleife
17  loop1:   sub.l    #1,d0
18           bne      loop1
19           andi.b   #253,$bfe001  ; LED dunkel
20           move.l   #20000,d0     ; Warteschleife
21  loop2:   sub.l    #1,d0
22           bne      l00p2
23           andi.b   #64,$bfe001   ; Maustaste gedrückt?
24           bne      start         ; Wenn nicht, dann weiter
25           rts                    ; Rückkehr
```

7.1: Die 8520-Bausteine

Die 8520-Bausteine haben 40 Anschlußpins. Hier die Pinbelegung dieses Bausteins:



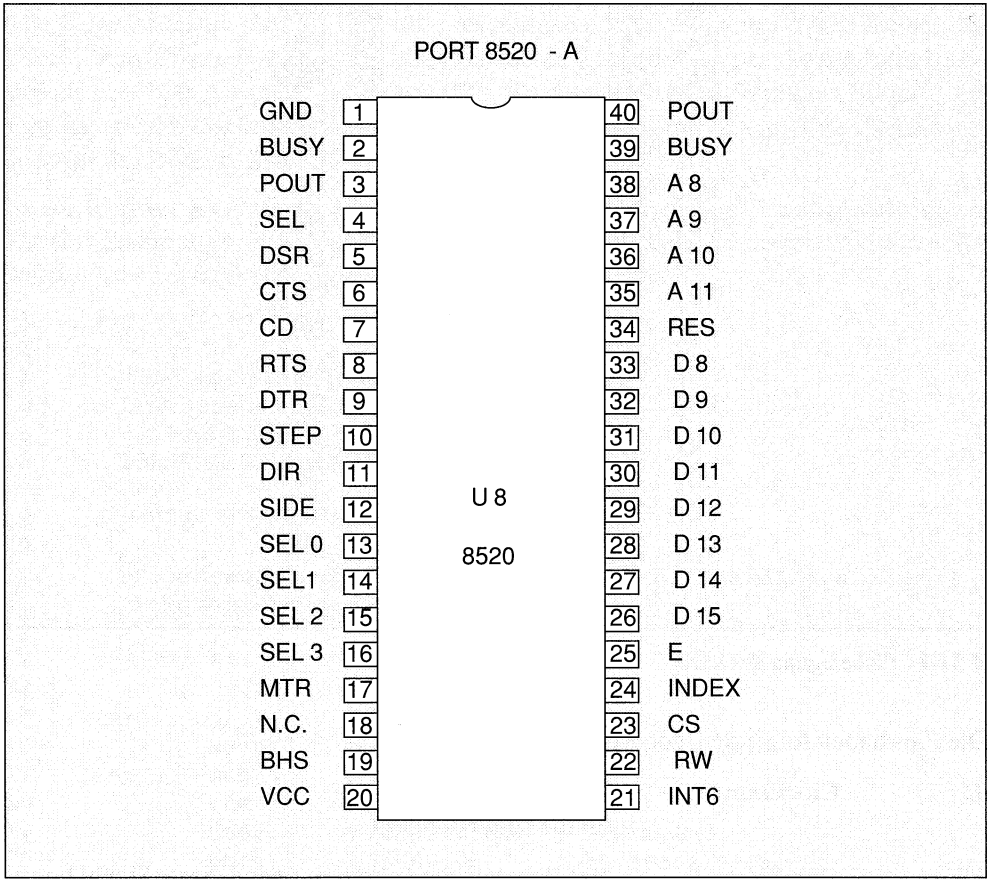
***Z 7.1-1:** Pinbelegung des 8520*

Die Pins haben folgende Bedeutung:

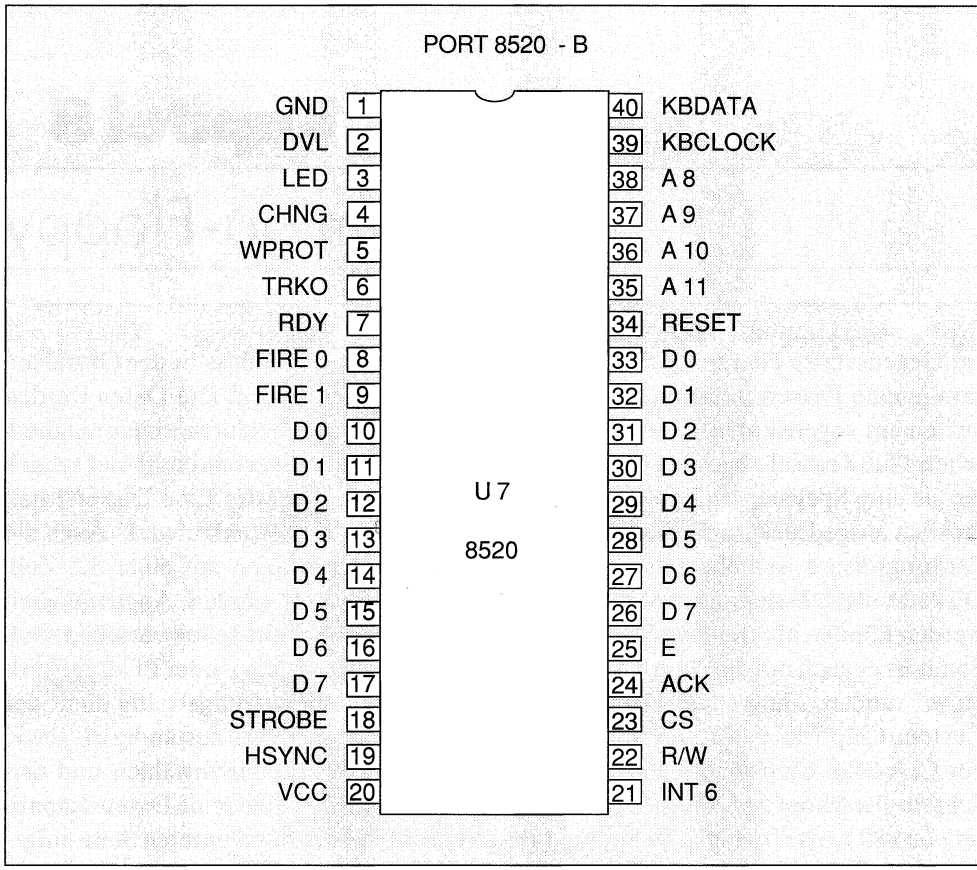
02	ClockInput	Dieses Signal wird für die Timer benötigt.
CS	ChipSelect	Der Baustein wird aktiviert, wenn dieses Signal Low ist.
R/W	Read/Write	Dieses Signal kontrolliert die Datenrichtung. Low zeigt einen Schreibzugriff an.
RS3-RS0	AdressInput	Adresse zur Decodierung der internen Register

DB7–DB0	DataBus	8-Bit-Datenbus zur Übertragung zwischen 8520 und System-Datenbus
IRQ	InterruptRequest	Bei Bedarf wird ein Interrupt-Signal an den Prozessor gesandt.
RES	Reset	Ist dieses Signal Low, werden alle internen Register zurückgesetzt.

Hier nun die speziellen Pinbelegungen des 8520-A und des 8520- B. Die Registerbeschreibungen ersehen Sie aus Kapitel 7 und Anhang D:



Z 7.1-2 *Pinbelegung des 8520-A*



Z 7.1-3: *Pinbelegung des 8520-B*

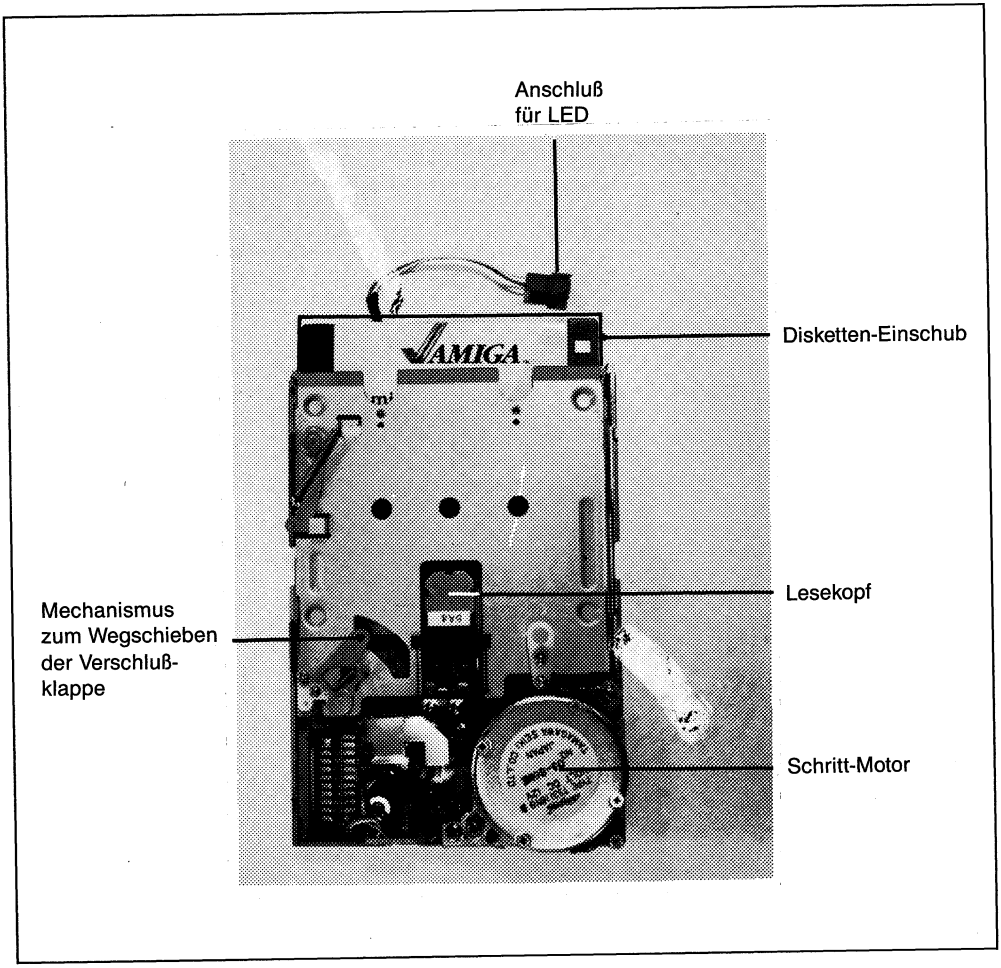
Kapitel 8

Die Amiga-Floppy

Im Gegensatz zu älteren Computer-Modellen enthalten die Amigas in der Grundversion je eine Diskettenstation zum Abspeichern von Programmen. Die Daten werden auf einem sogenannten Datenträger, einer 3.5-Zoll-Diskette, durch einen magnetischen Flußwechsel abgespeichert. Ein solches Aufzeichnungsverfahren ist viel schneller als eine Speicherung von Daten auf einer normalen Kassette. Eine Diskette hat, fachlich ausgedrückt, eine niedrigere Zugriffszeit als ein Kassettenlaufwerk. Auch die Packungsdichte ist höher als bei einer Kassette. Somit können auf einer 3.5-Zoll-Diskette mehr Daten auf einem kleineren Raum gespeichert werden. Angeschlossen werden können alle Laufwerke, die mit einer »Standard-Schnittstelle« versehen sind. Somit ist es auch möglich, ein 3.5-Zoll-Laufwerk als zweites Amiga- oder PC-Laufwerk zu verwenden, ohne einen Umbau am Floppy vorzunehmen. Amiga-seitig dient der Custom-Chip Paula als *Controller,* der für das Schreiben und Lesen zuständig ist, sowie ein CIA-8520-Chip für Steuerfunktionen, um die Laufwerke auszuwählen und den Schreib-/Lesekopf zu bewegen. Die Amiga-Laufwerke haben eine reine Daten-Kapazität von 880 Kbyte, die in je 80 Spuren (Tracks) auf der oberen und unteren Seite aufgeteilt sind. Jeder Track, auch oftmals als Zylinder bezeichnet, ist in 11 Sektoren mit je 512 Byte Speicherkapazität aufgeteilt. Diese Werte sind durch die Track-Disk-Device festgelegt und dienen als Richtwerte. Sie können aber unter anderem durch das direkte Ansprechen der Floppy-Hardware umgangen werden. Aufgezeichnet werden können die Daten wahlweise im *MFM-* oder *GCR-Format,* die man durch den Controller (Custom-Chip Paula) einstellen kann. Das erstere wird beim Amiga und auch beim IBM-PC verwendet. Das GCR-Format ist ein spezielles Apple-Format.

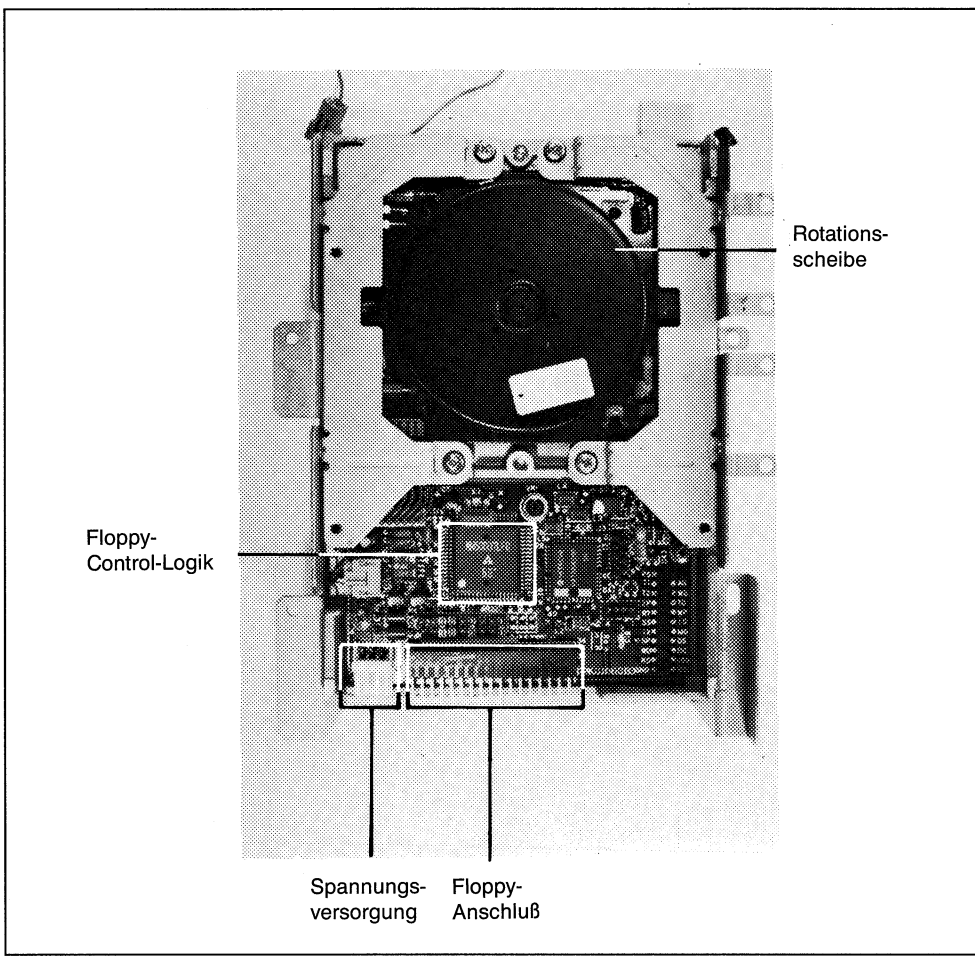
8.1: Der Aufbau des Amiga-Laufwerks

Durch die Aufzeichnungsverfahren wird eine große Genauigkeitsanforderung an die Amiga-Laufwerke gestellt. Der *Schreib-/Lesekopf* muß 100prozentig positionierbar sein und jeden Flußwechsel auf fast 1/1000 mm genau erkennen. Diese Laufwerke können schon als Wunderwerke der Technik bezeichnet werden, da hier hohe Präzision verlangt wird. Ist die Metallabdeckung des Laufwerks entfernt, können schon die wichtigsten Teile erkannt werden. Deutlich zu sehen ist der Lesekopf und der große *Schrittmotor.*



F 8.1-1: *Die Amiga-Floppy von oben*

An der Unterseite des Laufwerks sind über dem Anschluß der Spannungsversorgung und dem Verbindungsanschluß zum Amiga verschiedene LSI-Chips zu sehen, die die Regelung und Steuerung des Motors, sowie die Schreib- und Leseelektronik darstellen.



F 8.1-2: *Die Amiga-Floppy von unten*

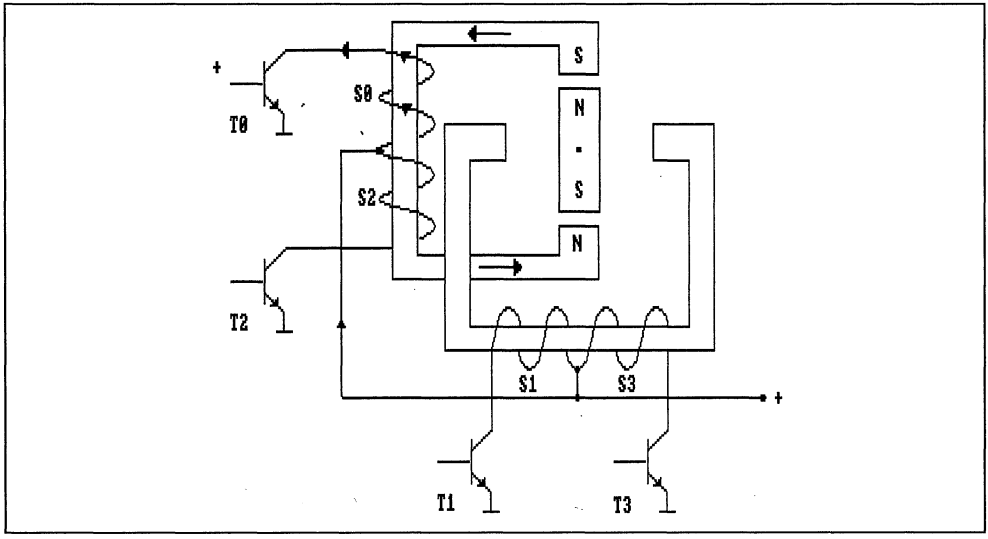
8.2: Der Diskettenantrieb

Unter dem Diskettenantrieb wird eine Vorrichtung verstanden, die die Diskette mit einer konstanten Geschwindigkeit dreht. Dies ist notwendig, damit die Abstände der verschiedenen »Bit-Zellen« der formatierten Diskette genau eingehalten werden können. Der Antrieb des Gleichstrommotors erfolgt beim Amiga-Laufwerk (auch hier gibt es verschiedene Versionen) direkt, d.h. die Einspannvorrichtung für die Diskette sitzt bei diesem Laufwerk direkt auf der Welle. Die Drehzahl läßt sich bei verschiedenen Laufwerken durch ein Poti einstellen, welches bei neueren Laufwerken nicht notwendig ist, da diese von Werk aus auf die Solldrehzahl eingestellt sind. Durch eine Elektronik und über die Schnittstelle zum Amiga läßt sich dieser Motor ein- und ausschal-

ten. Diese Aufgabe übernimmt Amiga-seitig einer der CIA-8520-Chips, worauf wir noch näher eingehen werden.

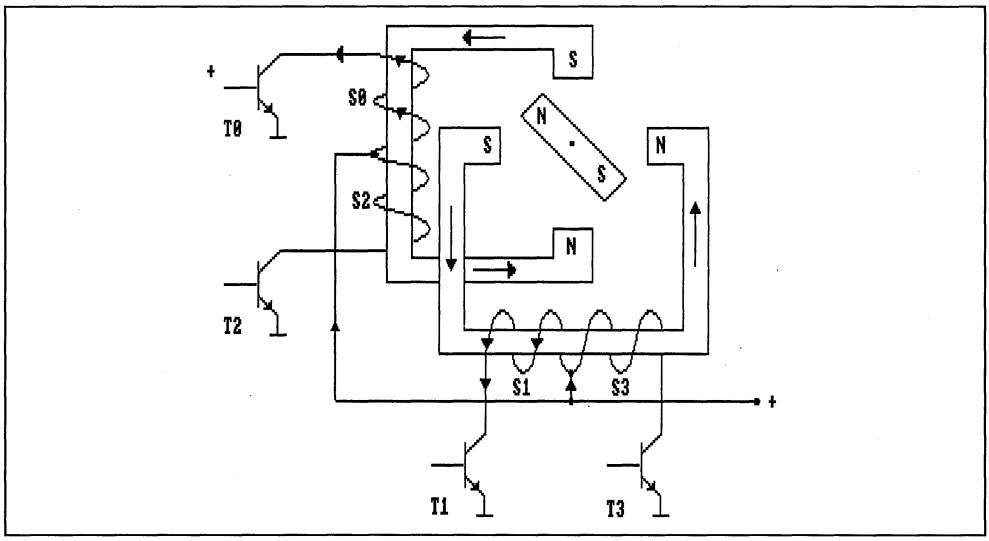
8.3: Die Positionierung des Schreib-/Lesekopfes

Zur Positionierung des Schreib-/Lesekopfes werden bei den Amiga-Laufwerken sogenannte *Schrittmotoren* benutzt. Diese Motoren sorgen dafür, daß der Schreib-/Lesekopf genau auf den gewünschten *Track* bzw. Radius gefahren wird. Abbildung Z 8.5-1 zeigt die Schematik eines solchen Motors in stark vereinfachter Form. Vereinfacht gesagt, besteht ein solcher Motor aus zwei festen Statoren, die je eine Spule mit Mittelanzapfung besitzen. An den Enden der Spulen befinden sich je zwei Transistoren, die den Stromfluß in den Spulen und somit das Magnetfeld des Stators steuern. In der Mitte des Stators befindet sich der Rotor, der aus einem Permanent-Magnet besteht. Wird der Transistor T0 positiv angesteuert, so schaltet er durch, und durch die Spule S0 kann ein Strom fließen. Somit baut sich im Stator ein magnetisches Feld auf. Da sich ungleichnamige Pole anziehen und gleichnahmige Pole abstoßen, dreht sich der Rotor in die Ausgangsposition Bild Z 8.3-1.



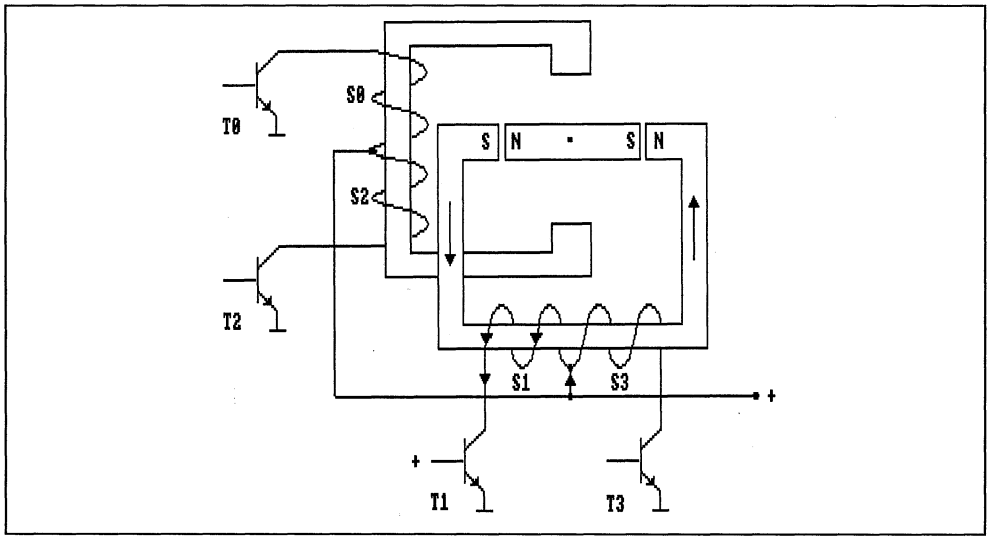
Z 8.3-1: *Transistor T0 ist durchgesteuert, durch die Spule S0 fließt Strom*

Um den Rotor nun um 45 Grad nach rechts drehen zu können, muß sich in der linken oberen Ecke beider Statoren der Südpol, und in der unteren rechten Ecke der Nordpol befinden. Dies erreicht man, indem zu dem durchgesteuerten Transitor T0, Transistor T1 durchsteuert. Dadurch fließt in der Spule S1 ein Strom, wodurch ein weiterer magnetischer Fluß in dem Stator entsteht. Der Rotor positioniert sich genau zwischen den beiden Südpolen, wie das Bild Z 8.3-2 zeigt.



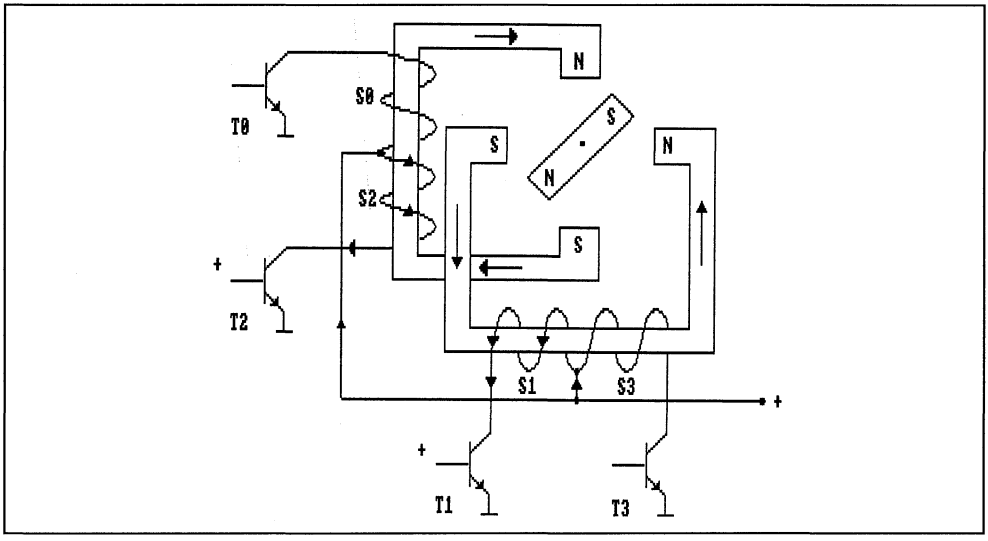
Z 8.3-2: *Transistor T0 und T1 sind durchgesteuert, durch die Spulen T0 und T1 fließt Strom*

Soll nun der Schrittmotor einen weiteren Schritt machen, so muß der Transistor T0 abgeschaltet werden. Dies hat zur Folge, daß durch die Spule T0 kein Strom mehr fließt und somit sich kein Magnetfeld bildet. Der Rotor bewegt sich um weitere 45 Grad nach links.



Z 8.3-3: *Transistor T0 ist abgeschaltet, T1 hingegen noch durchgesteuert*

Wird nun der Transistor T2 durchgesteuert und der Zustand von T1 bleibt erhalten, so dreht sich der Rotor um weitere 45 Grad.



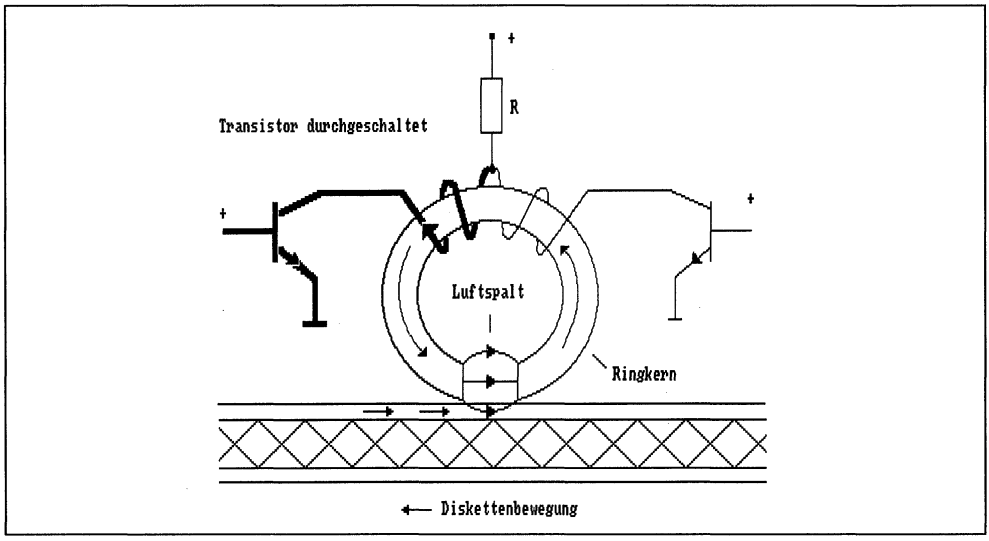
Z 8.3-4: *Transistor T1 und T2 sind durchgesteuert, durch die Spulen S1 und S2 fließt Strom*

Durch ein Wiederholen dieser Wechselspiele der Transistoren kann der Rotor um 360 Grad gedreht werden. Durch mehrere Pole am Rotor und Stator kann ein feinerer Schritt des Motors erzielt werden.

Die Drehbewegung des Schrittmotors muß nun noch in eine lineare Bewegung des Schreib-/Lesekopf-Schlittens, z.B. durch eine Gewindespindel, umgesetzt werden. Um den Schreib-/Lesekopf des Amiga-Laufwerkes zu bewegen, braucht der Hardwareprogrammierer nicht die Transistoren direkt anzusteuern. Die Amiga-Laufwerke besitzen eine Norm-Schnittstelle, wo nur die Signale »STEP« und die Richtung »DIRECTION«, nach innen oder nach außen, angegeben werden müssen. Ein Auf-/Abwärtszähler mit nachgeschalteter Dekodierlogik, die sich in einem der LSI-Chips des Floppy befindet, wertet die »STEP«- und »DIRECTION«-Signale aus und erzeugt die entsprechenden Phasensignale zum Bewegen des Schreib-/Lesekopfes. Dieses Dekodieren und Umsetzen dauert natürlich eine Weile, deshalb muß der Programmierer zwischen der Bewegung des Lesekopfes eine kleine Pause machen, damit der Kopf einwandfrei bewegt werden kann.

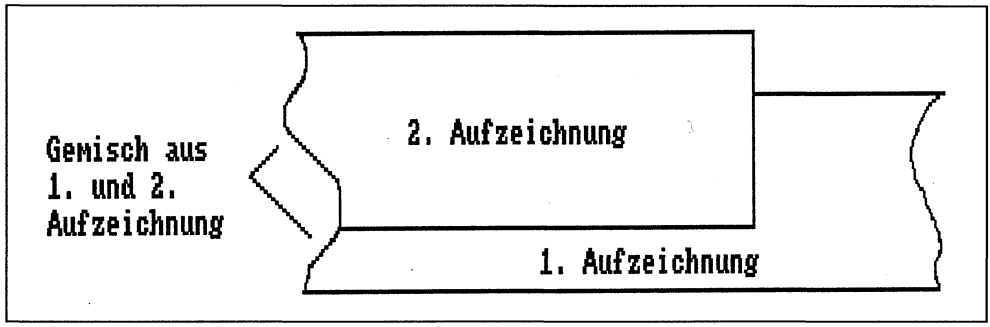
8.4: Der Schreib-/Lesekopf der Floppy

Der Schreib-/Lesekopf der Floppy dient zum Lesen und Schreiben von Daten. Er befindet sich auf einer Art Schlitten, der von dem Schrittmotor auf eine bestimmte Spur bewegt werden kann. Das Amiga-Laufwerk besitzt zwei Schreib-/Leseköpfe, da die Daten sowohl auf der oberen, als auch auf der unteren Seite der Diskette aufgezeichnet werden. Dieser Schreib-/Lesekopf besteht aus einem Ringkern mit einem geringem Luftspalt. Auf diesem Ringkern ist eine Spule mit Mittelanzapfung gewickelt. Die Enden der Spule gehen jeweils auf eine Transistor-Stufe, mit der die Richtung des Magnetfeldes, je nach Einschalten der Transistoren, gewechselt werden kann. Die Mittelanzapfung liegt über einem Widerstand an der Versorgungsspannung. Mit diesem Widerstand wird der Stromfluß zum Lesen/Schreiben reguliert.



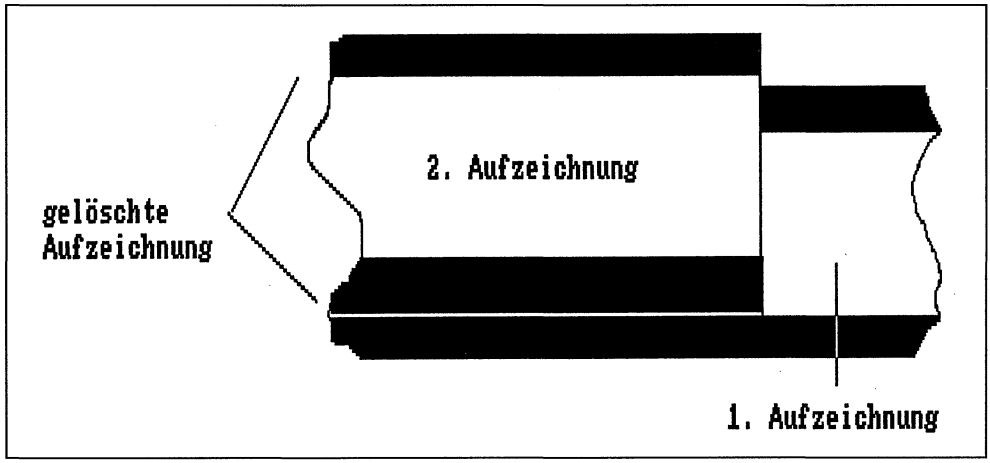
Z 8.4-1: *Der Schreib-/Lesekopf in schematischer Stellung*

Neben diesem Schreib-/Lesekopf ist ein Löschkopf vorhanden. Dieser Löschkopf hat die Aufgabe, einen kleinen Bereich links und rechts neben dem Schreib-/Lesekopf zu löschen, damit geringfügige Fehler beim Einspannen oder Positionieren der Diskette ausgeglichen werden. Ohne Löschkopf könnte beim Überschreiben einer Aufzeichnung ein Gemisch aus 1. und 2. Aufzeichnung entstehen, wenn z.B. bei der 2. Aufzeichnung ein anderes Diskettenlaufwerk verwendet wird.



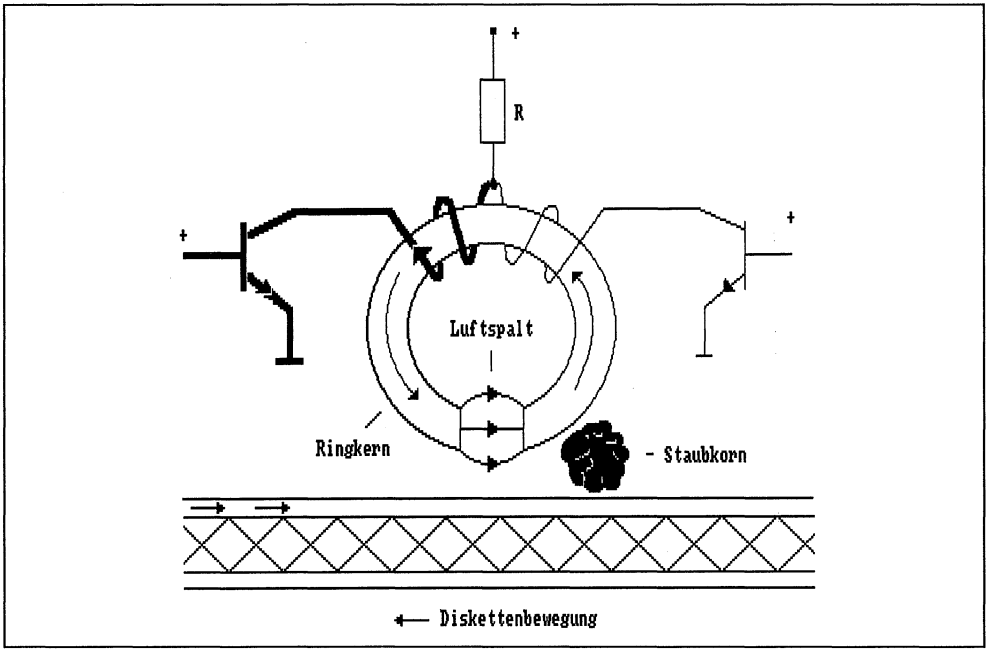
Z 8.4-2: *Ohne Löschkopf kann ein Gemisch aus 1. und 2. Aufzeichnung entstehen*

Mit dem Löschkopf wird der Rest der 1. Aufzeichnung gelöscht. So entsteht kein Gemisch aus 1. und 2. Aufzeichnung.



Z 8.4-3: *Mit dem Löschkopf werden die Reste der 1. Aufzeichnung gelöscht*

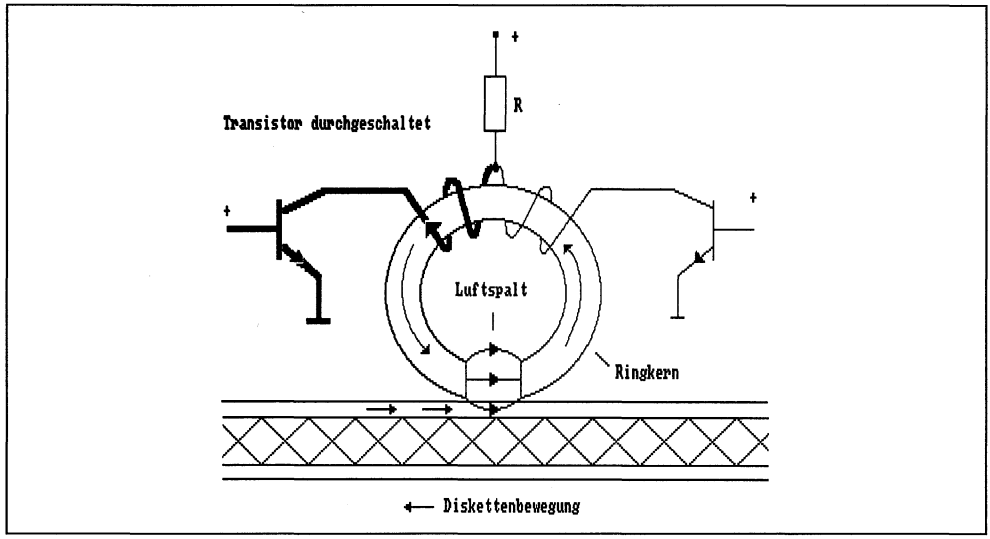
Wichtig für eine einwandfreie Aufzeichnung und einen optimalen Lesevorgang ist die Beschaffenheit der Diskette. Sie sollte frei von Staub und anderen strörenden Materialien sein. Nur dies garantiert, daß der Schreib-/Lesekopf dicht an der Diskette anliegt. Wird hingegen der Schreib-/Lesekopf nur geringfügig abgehoben, z.B. durch ein Staubkorn, kann dies Fehler beim Lesen und Schreiben verursachen.



Z 8.4-4: Ein Staubkorn hebt den Schreib-/Lesekopf leicht an. Somit kann der magnetische Fluß des Ringkerns nicht die Diskette durchsetzen

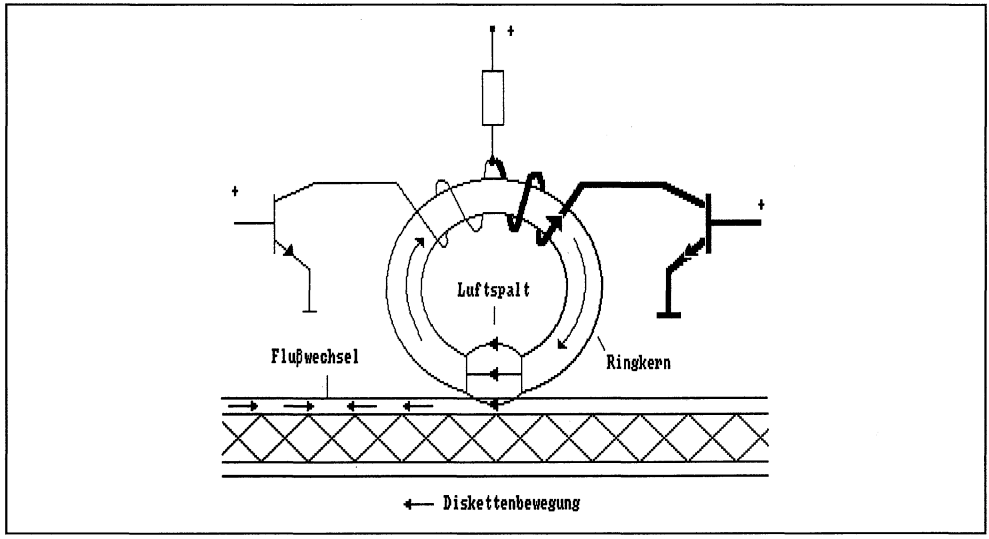
8.5: Der Schreibvorgang

Erinnern wir uns nochmal an den Aufbau des Schreib-/Lesekopfes. Er besteht aus einem Ringkern mit einer Spule mit Mittelanzapfung. An beiden Seiten der Spule befindet sich jeweils ein Transistor, der nach Masse durchgeschaltet wird. Ist nun der linke Transistor eingeschaltet, so fließt durch die Spule Strom. Der Ringkern wird dem Uhrzeigersinn entgegengesetzt magnetisiert. Da sich in dem Ringkern ein Spalt befindet, tritt an der Öffnungsstelle der magnetische Fluß aus dem Ringkern. Befindet sich der Ringkern dicht genug an der Diskette (auf den Aufbau der Diskette wird in den nächsten Kapiteln noch genauer eingegangen), werden die feinen Magnetteilchen der Diskette in Richtung des magnetischen Flusses des Ringkerns ausgerichtet. Dreht sich dabei die Diskette, bleibt die Magnetisierung in der Breite des Schreib-/Lesekopfes erhalten.



Z 8.5-1: *Die Magnetteilchen der Diskette werden nach rechts ausgerichtet*

Wird nun der linke Transistor abgeschaltet und der rechte eingeschaltet, so findet ein Flußwechsel im Ringkern statt, da die Wicklung zum rechten Transistor einen anderen Wicklungssinn hat als die zum linken Transistor. Dreht sich die Diskette immer noch, so entsteht auf derselben Spur nun ein Flußwechsel. Die Magnetteilchen richten sich auf der Diskette in die entgegengesetzte Richtung aus.

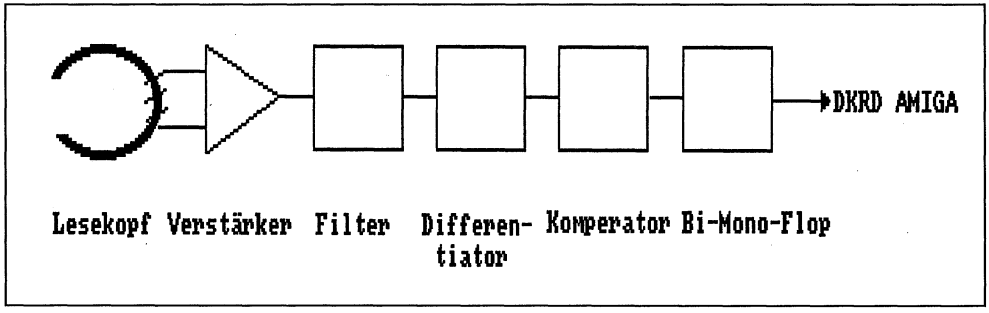


Z 8.5-2: *Die Magnetteilchen der Diskette werden nun nach links ausgerichtet. Deutlich zu erkennen ist der Flußwechsel*

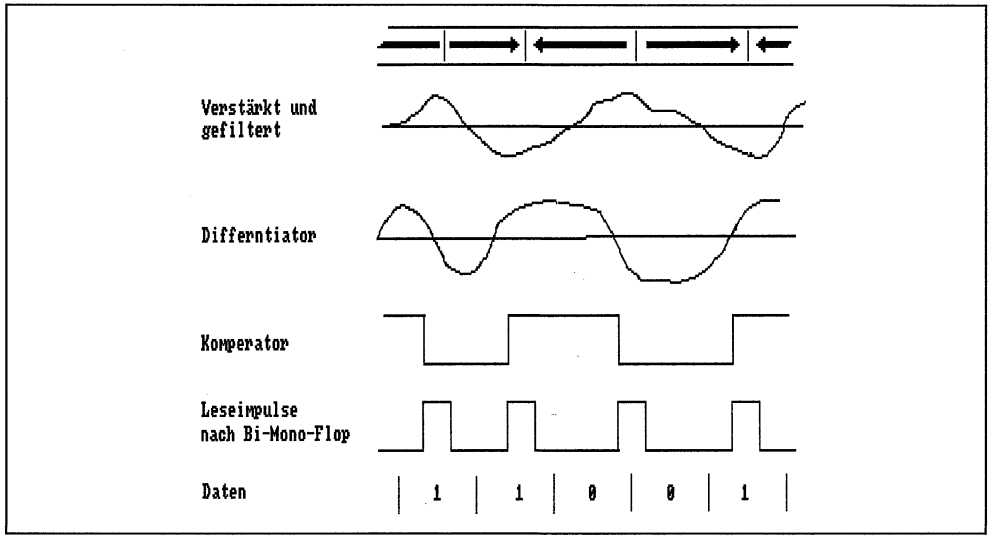
Sollen keine Magnetteilchen auf der Diskette verändert, somit keine Daten aufgezeichnet werden, so werden einfach beide Transistoren ausgeschaltet.

8.6: Der Lesevorgang

Der Lesevorgang ist etwas schwieriger als der Schreibvorgang, da hier nur eine Richtungsänderung der Magnetteilchen festgestellt werden soll. Diese Richtungsänderung der Magnetteilchen der drehenden Diskette erzeugt im Ringkern des Schreib-/Lesekopfes eine Spannung. Da diese sehr schwach ist, muß sie verstärkt werden. Nach dieser Verstärkung erhalten wir eine analoge Spannungsveränderung. Es werden jedoch für eine genaue Definition digitale Werte benötigt. Bevor jedoch dieser analoge Wert in einen digitalen umgewandelt werden kann, muß über einen Filter der gewünschte Frequenzbereich herausgefiltert werden, so daß z.B. hochfrequente Störungen von außen keinen Einfluß auf den gelesenen Datenstrom haben. Der so erhaltene analoge Wert hat seinen positivsten Wert, wenn die Magnetisierung der Diskette von links nach rechts wechselt. Findet ein Wechsel von rechts nach links statt, ist der analoge Wert am negativsten. Ein nachgeschalteter Differentiator setzt diese analogen Signale so um, daß bei jedem positven oder negativen Spannungsmaximalwert ein Nulldurchgang des Spannungsverlaufes stattfindet. Mit einem Komperator werden diese Nulldurchgänge herausgefiltert. Ist das Vergleichssignal des Komperators nicht positiver als der Signaleingang, so ist der Ausgang logisch 1. Bei umgekehrtem Verhalten der Eingangssignale ist der Ausgang gleich logisch 0. Man erhält somit ein Signal, das logisch 1 ist, wenn die Magnetteilchen nach links, und logisch 0, wenn die Magnetteilchen nach rechts auf der Diskette ausgerichtet sind. Wichtig für eine weitere Verwendung sind die Flußwechselimpulse. Diese erhält man, wenn man das Ausgangssignal des *Komperators* auf ein bidirektionales *Mono-Flop* legt. Dieses Mono-Flop gibt jeweils einen Impuls bei einer ansteigenden und auch bei einer abfallenden Flanke von sich. Abbildung Z 8.6-1 und Z 8.6-2 zeigen diesen Verlauf nochmals schematisch.



Z 8.6-1: *Das Blockschaltbild einer Leseelektronik für ein Diskettenlaufwerk*



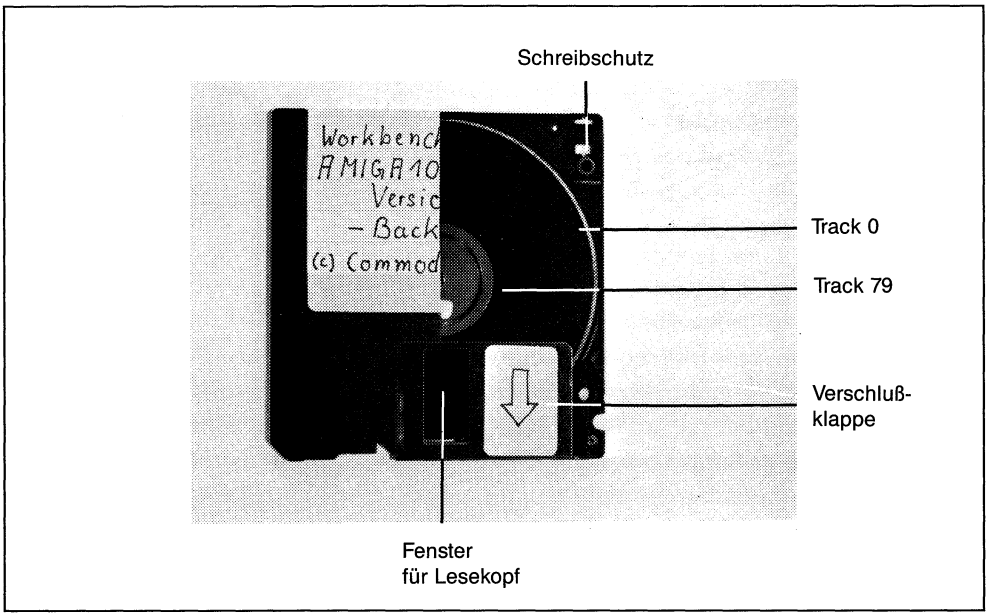
Z 8.6-2: *Viele Umwandlungen sind notwendig, um das geeignete Signal zu erhalten*

8.7: Die 3.5-Zoll-Diskette

Bevor wir zu den Kapiteln der verschiedenen Aufzeichnungsverfahren kommen, wollen wir den Aufbau einer 3.5-Zoll-Diskette kurz beschreiben.

Bei den Amiga-Laufwerken werden 3.5-Zoll-Disketten, doppelseitig, mit doppelter Dichte verwendet. 3.5 Zoll gibt hier den Durchmesser der Diskette an. Die Hülle hingegen ist 3.54 Zoll (90.0 mm) breit, 3.7 Zoll (94.0 mm) lang und 0.13 Zoll (3.3 mm) hoch. An der unteren Seite der Hülle befindet sich eine Verschlußklappe, die bei Lese-/Schreibversuchen des Diskettenlaufwerks durch eine Mechanik zurückgefahren wird. An der oberen rechten Ecke der 3.5 Zoll breiten Hülle befindet sich ein Schreibschutz, der, nach oben geschoben, einen kleinen Taster zum Öffnen zwingt (je nach Laufwerk) und somit die Diskette vor dem Beschreiben schützt. In dieser Hülle befindet sich die magnetbeschichtete Folie, die als Diskette bezeichnet wird. Sie ist in der Mitte mit einem Metallring verstärkt. Bei den Daten der Disketten besagt die Angabe doppelseitig oder double sided, daß die Magnetschicht, die nur ca. 0,002 mm dick ist, auf beiden Seiten aufgetragen ist. Einseitige Disketten können unter Umständen auch benutzt werden, da bei der Produktion von Disketten die Magnetschicht meistens beidseitig aufgetragen wird. Jedoch wird bei einseitigen Disketten nur eine Seite getestet, die zweite Seite kann durchaus noch Fehler enthalten und ist somit eventuell für den Amiga-Benutzer unbrauchbar. Doppelte Dichte gibt an, wie fein diese Magnetschicht ist. Je feiner sie ist, desto mehr Daten können auf einem kleinem Raum zusammengepackt werden. Diese Daten werden unterteilt in verschiedene Segmente, sogenannte *Tracks* bzw. *Cylinder.* Diese Cylinder enthalten nochmals eine Unterteilung in

Sektoren. Der erste Cylinder bzw. Track befindet sich fast in der Mitte der Diskette, der letzte Track am äußeren Ende der Diskette.

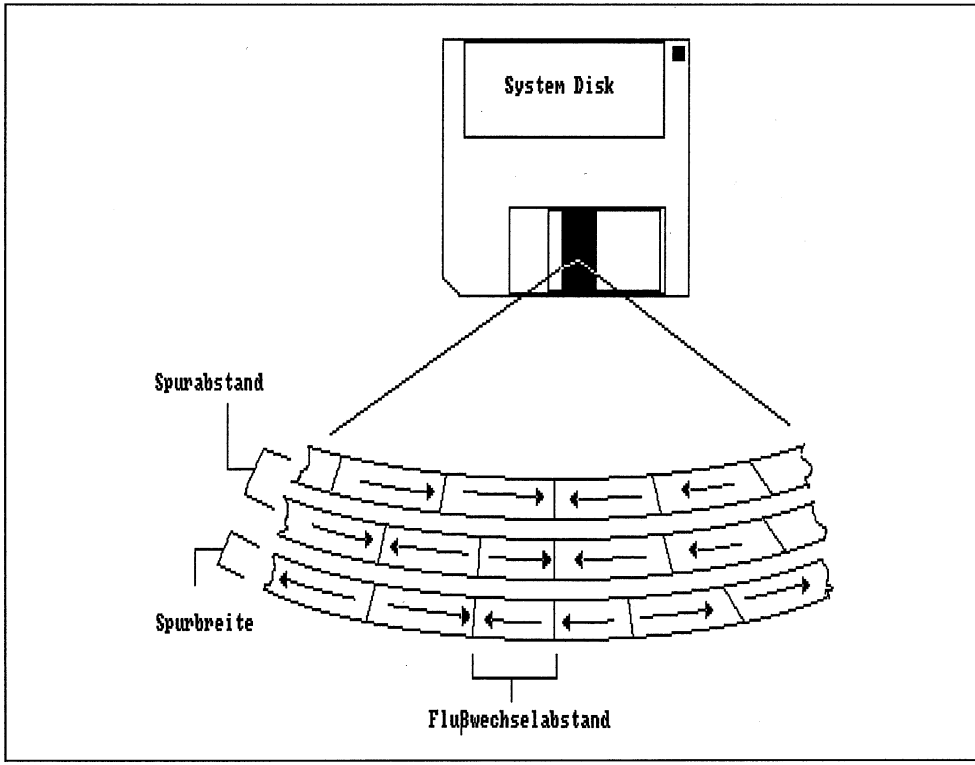


F 8.7-1: *Der Aufbau einer Diskette*

8.8: Die physikalische Aufzeichnung

Bevor wir zu den Codierungsarten *MFM* bzw. FM und *GCR* kommen, wollen wir auf die physikalische Aufzeichnung von Daten auf einer 3.5-Zoll-Diskette näher eingehen.

Werden Daten auf eine 3.5-Zoll-Diskette aufgezeichnet, so werden die feinen Magnete der Diskette in eine bestimmte Richtung ausgerichtet. Hierbei wird der Magnetlese-/schreibkopf des Diskettenlaufwerks auf einen bestimmten Track (man kann auch sagen, daß er auf einen bestimmten Radius bzw. Spur bewegt wird) gefahren. Die Diskette rotiert dabei schon mit einer konstanten Geschwindigkeit. Wird durch den Magnetkopf nun ein Strom geschickt, so werden die feinen Magnete der Diskette je nach Strom in die eine oder in die andere Richtung ausgerichtet, welches als Flußwechsel bezeichnet wird. Es entsteht dabei ein konzentrisch geschlossener Magnetfluß. Durch die Positionierung des Schreib-/Lesekopfes auf verschiedene Tracks bzw. Cylinder können mehrere dieser konzentrischen Spuren aufgezeichnet werden. Wie dicht diese Spuren gepackt werden können, wird als tpi = Tracks per Inch (Spuren pro Zoll) angegeben. Nach der Art des Diskettentyps bzw. des Diskettenlaufwerks richtet sich der Spurabstand, die Spurbreite oder der Flußwechselabstand.



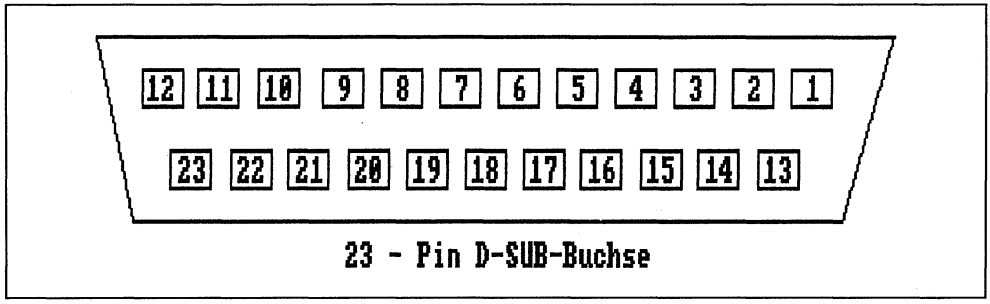
Z 8.8-1: *Die Aufzeichnung von konzentrischen Spuren*

Der Flußwechselabstand wird in fci (flux changes per inch = Flußwechsel pro Zoll) angegeben. Dies entspricht dem Kehrwert des kleinsten Abstandes zwischen zwei Flußwechseln auf der innersten Spur. Der Abstand zwischen zwei Flußwechseln ist bei 3.5-Zoll-Disketten kleiner als 0,005 mm! Je nach Aufzeichnungsverfahren geben verschiedene Flußwechsel ein Bit an. Bleibt der Flußwechsel gleich, so ist das Bit gleich Null. Die Bitdichte berechnet sich aus dem Kehrwert des Abstandes zwischen zwei Bits der innersten Spur. Diese wird auch als bpi (Bits per inch = Bits pro Zoll) bezeichnet. Bei dem MFM-Aufzeichnungsverfahren entspricht die Bitdichte dem Flußwechselabstand. Beim FM-Verfahren hingegen ist die Bitdichte halb so groß wie der Flußwechselabstand. Die Aufzeichnungsverfahren sind von dem Controller abhängig.

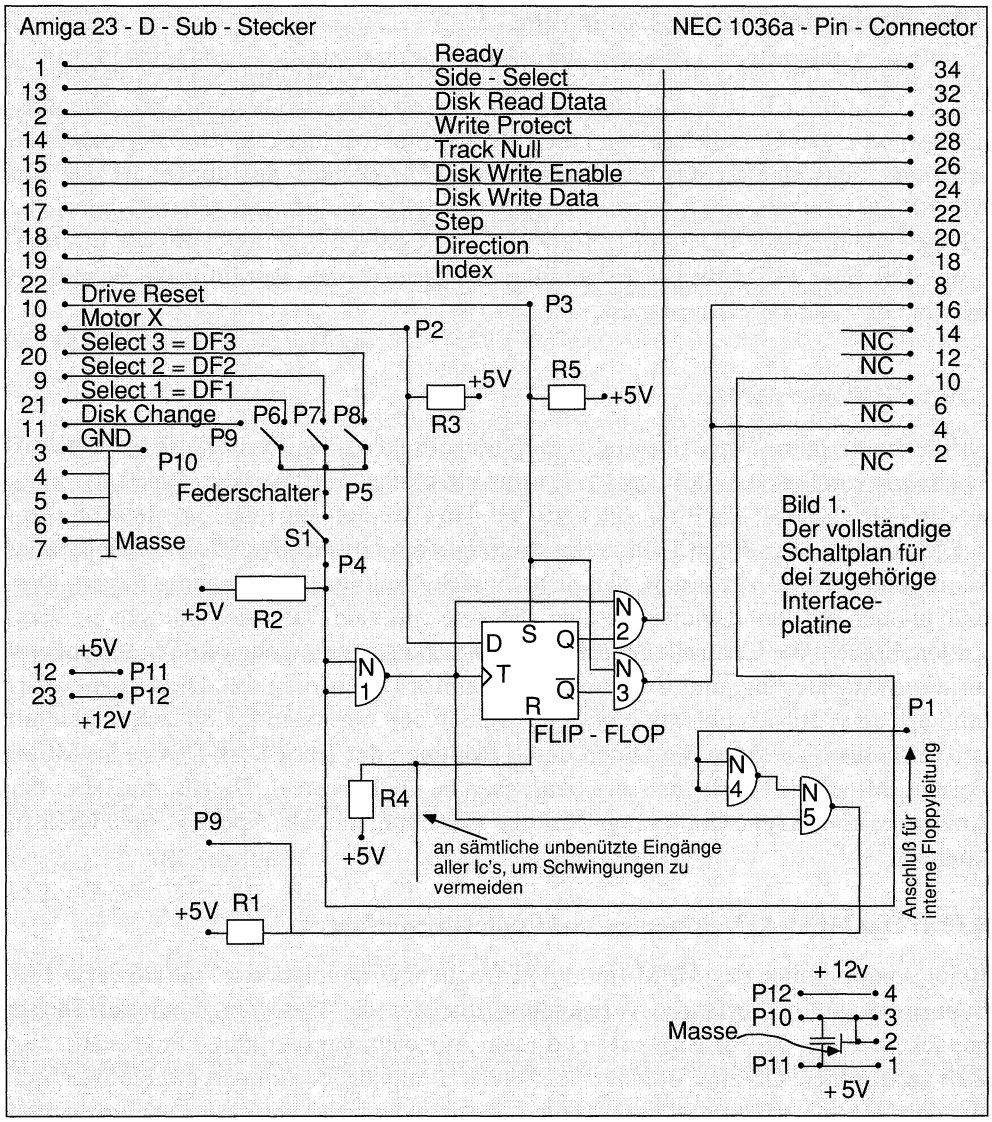
8.9: Die Standard-Floppy-Schnittstelle

Der Amiga 500 und 1000 sind leider nur mit je einem Floppy ausgestattet. Sehr schnell kommt so der Wunsch auf, beim Kopieren von Daten oder beim Arbeiten mit mehreren Disketten, ein zweites Laufwerk anzuschließen. Alle Amiga's besitzen dazu eine »Floppy-Schnittstelle«, einen sogenannten 23-D-Sub-Stecker. Hier können Standard-Laufwerke, wie das NEC 1036A angeschlossen werden. Diese Laufwerke haben einen Shugart-Bus, der den Anschluß sehr vereinfacht.

Alle Signale dieses Shugart-Busses sind low-aktiv, d.h. alle Pegel sind logisch high (+5V) und erst wenn ein Signal anliegt, geht die jeweilige Leitung auf logisch low (0V). Da der Amiga extern maximal 3 weitere Laufwerke verwalten kann, wird für das Selektieren eine kleine Logik benötigt, damit beim Ansprechen eines externen Laufwerks nicht alle Laufwerke des Amiga angesprochen werden. Ein D-FlipFlop kann hier die Funktion des Selektierens übernehmen. Select 1 (DF1), Select 2 (DF2) oder Select 3 (DF3) dient hier als Takt, der bei einer positiven Flanke das Weitergeben des »Motor On«-Signals bewirkt. Über ein NAND-Gatter wird der Ausgang des Flip-Flop an den Ready-Pin gemeldet. Daran erkennt der Amiga beim Abfragen der Laufwerke, welches Laufwerk angeschlossen ist. Diese Prozedur wird beim Initialisierungsvorgang durchgeführt. Findet der Amiga eine Rückmeldung, so wird für das jeweilige Laufwerk ca. 20 Kbyte Speicher reserviert. Schaltet man die Select-Leitung des Laufwerks nach dieser Initialisierung ab, so bleibt der reservierter Speicherbereich vorhanden. Erst bei einem Reset wird er gelöscht. Ein weiteres Problem stellt das Signal »DiskChange« dar. Dieses Signal signalisiert einen Diskettenwechsel und ist leider nicht direkt am Shugart-Bus vorhanden. Dieses Signal liegt im Laufwerk an einem Lötstützpunkt an und muß von dort an den Pin1 der Select-Logik-Platine angelötet werden. Abbildung Z 8.9-2 zeigt den Aufbau einer solchen Logik.



Z 8.9-1: *Der Amiga-Floppy 23-D-Sub-Stecker*



Z 8.9-2: *Eine kleine Logik übernimmt das Selektieren des Laufwerks*

Bauteile für das Interface:

IC1 – 74LS00
IC2–74LS74
IC3–74LS38
R1–R5 1 KOhm 1/4 Watt
C1 Kondensator 100 Nanofarad / 50 Volt
1 23poliger D-SUB-Stecker

8.10: Paula, der Floppy-Controller

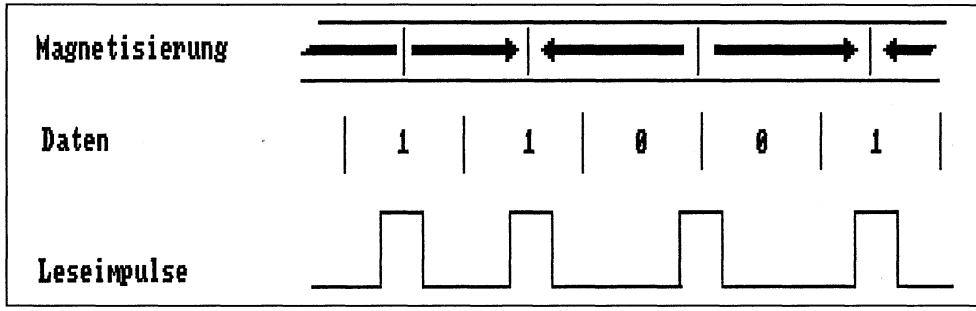
Der Custom-Chip *Paula* übernimmt im Amiga die Funktion eines Floppy-Controllers. Dieser Controller liest und schreibt die Daten von bzw. auf die Diskette. Ebenfalls übernimmt er das Umschalten von Lesen auf Schreiben. Paula kann zwei Formate »verarbeiten«. Das eine ist das schon erwähnte *MFM-Format,* das andere ist das von APPLE her bekannte *GCR-Format.* MFM- bzw. GCR-codiert/decodiert werden die zu schreibenden/lesenden Daten per Software. Der Controller benötigt nur die Information GCR oder MFM für die Erkennung des *Sync-Wortes,* sowie für die Schreibgeschwindigkeit und Prekompensation.

8.10.1: GCR oder MFM?

Wie schon erwähnt (siehe Kapitel: Die physikalische Aufzeichnung) ist bei der Aufzeichnung von Daten ein Bit logisch 1, wenn ein Flußwechsel vorliegt. Bleibt der Flußwechsel gleich, so ist das Bit gleich logisch 0. Da aber der Controller bei gleichbleibender Magnetisierung durch Laufwerksschwankungen aus der Synchronisation geraten kann, muß verhindert werden, daß nicht zu viele Nullbits hintereinander folgen. Deshalb müssen die Daten codiert werden, um eine optimale Datensicherung zu erzielen. Zudem braucht der Controller eine Start-Markierung, eine sogenannte *Sync-*(Synchronisations)Markierung, um zu erkennen, ab wann er Daten lesen soll. Dieses Sync-Wort muß aus einer Bit-Kombination bestehen, die in den normalen Daten nicht enthalten ist. Verschlüsselt werden die Daten durch Routinen der Track-Disk-Device im MFM-Format. Möglich ist auch ein Schreiben/Lesen im GCR-Format. Hierfür muß aber der Anwender eine eigene Codierungs-Routine schreiben, um z.B. Apple-Format lesen zu können.

8.10.2: Das MFM-Aufzeichnungs- und Codierungsverfahren

Beim Amiga findet das MFM-(modified frequency modulation = modifizierte Frequenzmodulation) Verfahren Verwendung, da hier die Daten in doppelter Dichte gepackt werden. Jedem Bit wird bei diesem Aufzeichnungsverfahren eine bestimmte Zeit zugeordnet. Ob eine Bitzelle eine 0 oder 1 enthält, bestimmen die Flußwechsel. Bei dem MFM-Verfahren ist die Bitzelle 1, wenn in der Mitte der Zelle ein Flußwechsel stattfindet. Findet keiner statt, so ist die Zelle gleich 0.



Z 8.10.2-1: *Die MFM-Aufzeichnung*

Zu diesen Daten-Bits werden noch zusätzlich Takt-Bits geschrieben, die garantieren, daß der Controller während des Lesens der Daten-Bits nicht aus der Synchronisation gerät. Darin liegt die eigentliche Codierung der Daten. Diese Takt-Bits werden so gesetzt, daß zwischen zwei Daten-Bits ein Takt-Bit eingefügt wird. Dieses Takt-Bit ist gesetzt, wenn das linke und rechte Daten-Bit gleich Null ist. Ist eines der beiden Daten-Bits gesetzt, so ist das Takt-Bit gleich Null.

Als Beispiel soll das Byte $1E codiert werden:

Bitmuster	Byte
%00011110	$1E

Jetzt werden die Takt-Bits eingefügt :

	Bitmuster										Byte/Wort
Daten-Bits	%	0	0	0		1	1	1	1	0	$1E
		I	I	I		I	I	I	I	I	
Takt-Bits	%	1	1	1		0	0	0	0	0	
Daten-Wort	%	10	10	10		01	01	01	01	00	$A954

Wie man sieht, wird verhindert, daß mehrere Null- und Eins-Daten-Bits hintereinander stehen. Überhaupt kann bei dieser Codierung nie ein gesetztes Daten-Bit auf ein nächstes gesetztes Daten-Bit folgen. Somit ist es möglich, einen Daten-Fluß beim Schreiben von 2 ms pro Bit zu erreichen, da kein häufiger Flußwechsel benötigt wird. Das Einzige, was nun noch benötigt wird, ist die Start-*(Sync)Markierung,* woran der Controller erkennt, daß er nun lesen darf. Dies kann eigentlich ein beliebiges Wort sein, es darf nur nicht in den vorhandenen Daten vorkommen, da sonst der *Controller* eventuell mitten im Daten-Satz mit dem Lesen beginnt. Also müssen wir nach einer Kombination suchen, die im normalen Daten-Satz nicht vorkommt. Eine Möglichkeit sind mehrere Eins-Bits hintereinander zu setzen, welches aber den Nachteil eines schnellen Wechsels des Magnetfeldes zur Folge hat, wobei der Controller nicht mehr mitspielt

und es zu Fehlern kommen kann. Eine weitere Möglichkeit ist die Folge von drei aufeinanderfolgenden Null-Bits. Diese Kombination kann nie bei einer Codierung entstehen, da immer zwischen zwei Null-Bits ein gesetztes Takt-Bit eingefügt wird. Eine solche Kombination ist z.B. $4489, die auch beim Amiga-DOS als Sync-Wort verwendet wird. Findet der Controller dieses Wort, so beginnt er von da ab die Daten zu lesen und synchronisiert erst wieder neu, wenn der Lesevorgang zuende ist.

8.10.3: Das GCR-Aufzeichnungs- und Codierungsverfahren

Ein weiteres Format, das der Controller erlaubt, ist das GCR- (Group Code Recording) Verfahren. Bei der Codierung muß hier der Binärwert in ein 5-Bit-GCR-Äquivalent umgewandelt werden:

GCR-Äquivalent	Binärwert	Hexadezimalwert
01010	0000	$0
01011	0001	$1
10010	0010	$2
10011	0011	$3
01110	0100	$4
01111	0101	$5
10110	0110	$6
10111	0111	$7
01001	1000	$8
11001	1001	$9
11010	1010	$A
11011	1011	$B
01101	1100	$C
11101	1101	$D
11110	1110	$E
10101	1111	$F

Bei dieser Codierung können nur max. acht Einserbits oder max. zwei Nullbits hintereinander stehen. Die Datenaufzeichnung nimmt zwar so weniger Platz im Vergleich zum MFM-Verfahren in Anspruch, da aber bei max. acht hintereinander folgenden Einserbits der magnetische Flußwechsel sehr oft vorkommt, muß der Controller langsamer schreiben, um die Daten fehlerfrei ablegen zu können. Die Aufzeichnungsgeschwindigkeit beträgt hier 4 ms pro Bit. Diesen Geschwindigkeitsverlust kompensiert der Apple-Macintosh mit einem speziellen Trick. Seine Laufwerke ändern die Drehzahl in Abhängigkeit von der Spur. Das ist der Grund, weshalb andere Computer im Regelfall keine Macintosh-Disketten lesen können. Die Daten müssen jeweils in Blöcke von vier Byte zu fünf Byte codiert werden, da beim Codieren von nur 2 Byte 2 Bit als Rest übrig bleiben würden. Als Sync-Markierung kann hier die Folge von neun Einserbits dienen, da

bei dieser Codierung nur max. acht Einserbits möglich sind. Nach der Sync-Markierung muß ein Nullbit folgen, damit der Controller erkennt, ab wann die Datenbits folgen.

8.10.4: Das AMIGA-Disk-Kontroll-Register ADKCON

Die Parameter-Einstellung, ob mit oder ohne Sync-Markierung, MFM- oder GCR-Verfahren, 2 ms oder 4 ms Aufzeichnungsgeschwindigkeit, kann durch das Register ADKCON und ADKCONR festgelegt werden. ADKCON ist hierbei die Schreib-, und ADKCONR die Leseadresse. Die untersten 8 Bit des Register, beziehen sich auf die Sound-Hardware und sind für unsere Anwendung in diesem Kapitel ohne Bedeutung. Das wichtigste Bit ist das Bit 15 des Registers. Mit diesem Bit kann, wie bei dem *Interrupt*-Register, das Setzen und Löschen einzelner Bits in dem Register gesteuert werden. Ist dieses CLR/SET-Bit gesetzt, so werden alle Bits des jeweiligen Wortes in das Register übernommen. Ist dieses Bit gelöscht, so werden alle Bits, die in das Register übernommen werden, gelöscht:

Setzen neuer Bits CLR/SET-Bit = 1:

	CLR/SET-Bit	
Neuer Wert	1	1100101000000000
Alter Wert	0	0110110101110000
Ergebnis	1	1110111101110000

Löschen neuer Bits CLR/SET-Bit = 1:

	CLR/SET-Bit	
Neuer Wert	0	1100101000000000
Alter Wert	0	0110110101110000
Ergebnis	1	0010010101110000

Bit	Name	Bedeutung
15	CLR/SET	Dient zum Löschen und Setzen einzelner Bits.
14	PRECOMP1	Höchstes Bit von PRECOMP.
13	PRECOMP2	Niedrigstes Bit von PRECOMP. Diese beiden Bits geben die Prekompensation beim Schreiben an:

PRECOMP2	PRECOMP1		
0	0	=	0 ns
0	1	=	140 ns
1	0	=	280 ns
1	1	=	560 ns

Bit	Name	Bedeutung
12	MFMPREC	MFMPREC = 0 MFM-Format MFMPREC = 1 GCR-Format
11		Keine Bedeutung für Floppy
10	WORDSYNC	Wird hier eine Eins gesetzt, so beginnt der Controller erst mit der Übertragung der Daten nach dem Auffinden des Sync-Wortes im Sync-Register $DFF07E.
9	MSBSYNC	Dies dient zum Einschalten der GCR-Sync-Markierung.
8	FAST	Mit diesem Bit wird die Schreibgeschwindigkeit gesteuert. Eine 0 entspricht GCR-Format, somit 4 ms pro Bit. Eine 1 entspricht MFM-Format 2 ms pro Bit.
7–0		Keine Bedeutung für Floppy.

8.10.5: Das Disk-Sync-Register

Ist das Word-Sync-Bit (Bit Nr. 10) im ADKCON-Register gesetzt, sucht der Controller zunächst nach dem Wort, das im Register DSKSYNC abgelegt ist. Erst wenn dieses Wort gefunden wurde, beginnt er mit der Datenübertragung. Beim Finden des Sync-Wortes löst der Controller einen Interrupt mit der Priorität 6 aus.

8.10.6: Die Disk-Pointer-Register DSKPTH und DSKPTL

Wenn der Controller Daten lesen bzw. schreiben soll, muß ihm vorher mitgeteilt werden, aus welchem Bereich er Daten holen soll. Dies kann mit dem Disk-Pointer-Register DSKPTH und DSKPTL dem Controller mitgeteilt werden. DSKPTL geben die unteren 8 Bit, DSKPTH die oberen 8 Bit der Startadresse an. Somit stehen rein theoretisch 32 Bit für die Adressierung zur Verfügung. Da aber die Custom-Chips des Amiga nur die unteren 512 Kbyte ansprechen können, d.h. 19 Bits für die Adressierung, sind die oberen 13 Bits des DISKPTH ohne Bedeutung. Gut gelöst ist, daß beide Register direkt hintereinander liegen. Somit kann durch einen move.l-Befehl (Long-Word = 32 Bit) die Startdresse auf einmal gesetzt werden. Wird die Übertragung gestartet, so werden die Disk-Pointer-Register hochgezählt.

8.10.7: Das DSKLEN-Register

Das DSKLEN-Register steuert den *DMA*-Zugriff, sowie die Anzahl der zu schreibenden und lesenden Daten. Transferieren lassen sich maximal 2^13 Bit Daten. Dies entspricht gleich 16 Kbyte. Bemerkenswert ist, das beim Übertragen von Daten ein Hardwarefehler vorliegt. So gehen die letzten drei Bits der transferierten Daten zur Disk einfach verloren. Ebenfalls verschwindet das letzte Wort der gelesenen Daten von der Disk.

Um die Disk-DMA zu starten, muß das Register zweimal beschrieben werden, danach muß das DMA-Bit wieder abgeschaltet werden, um Fehler zu vermeiden. Wird die Übertragung gestartet, so wird das DSKLEN-Register heruntergezählt. Erst wenn der Zählwert im DSKLEN-Register gleich Null ist, wird die Übertragung beendet.

Bit	Name	Bedeutung
15	DMAEN	Dieses Bit steuert den Disk-DMA-Zugriff. Ist dieses Bit Eins, so wird der DMA-Zugriff ermöglicht.
14	WRITE	Ist dieses Bit auf Eins gesetzt, so signalisiert dies das Schreiben der Daten, eine Null das Lesen.
13-0	LENGTH	Länge der Daten.

8.10.8: Das Disk-Byte-Read-Register DSKBYTR

Dieses Register ist eine Art Kontroll-Register, mit dem ein Teil der gesetzten Bits in dem vorhergehenden Register überprüft werden kann. Zudem kann aus diesem Register das gerade gelesene Byte von der Diskette entnommen werden. Wurde ein Byte angetroffen, so wird das BYTEREADY-Bit gesetzt. Beim Auslesen dieses Registers wird dieses Bit automatisch gelöscht.

Bit	Name	Bedeutung
15	BYTEREADY	Dieses Bit signalisiert, ob ein Byte von dem Diskettenlaufwerk angekommen ist. Bei einem Auslesen des Registers wird dieses Bit automatisch gelöscht.
14	DMAON	Ist die Disk-DMA zugelassen, so ist dieses Bit gesetzt. Dieses Bit ist erst logisch 1, wenn das DMA-Bit im DSKLEN- und auch DMACON-Register gesetzt ist.
13	DISKWRITE	Dieses Bit signalisiert, ob im DSKLEN-Register der Schreib- oder Lesemodus eingeschaltet ist.

Bit	Name	Bedeutung
12	WORDEQUEL	Wird eine Sync-Markierung gefunden, so ist dieses Bit gleich 1. Dieses Bit bleibt nur solange gesetzt, wie die Sync-Markierung erkannt wird.
11–8		Keine Funktion.
7–0	DATA	Diese untersten 8 Bit enthalten das Byte, das gerade von der Diskette gelesen wurde.

8.10.9: Die Disk-Daten-Register DSKDAT und DSKDATR

Diese Register werden nur als Zwischenpuffer von der DMA beim Lesen und Schreiben von Daten benötigt. DSKADATist das Schreib- und DSKDATR das Leseregister.

8.11: CIA 8520, die Diskettensteuerung

Neben dem *Controller,* der nur für das Lesen und Schreiben der Daten, sowie der Sync-Erkennung dient, wird ein weiterer Baustein benötigt, mit dem das Laufwerk selektiert, und der Schreib-/Lesekopf bewegt wird. Beim Amiga können diese Funktionen über die CIA-B ausgeführt werden. Zur Kontrolle, um den Status des Disk-Laufwerks zu testen, dient CIA-A, mit der überprüft werden kann, ob die Disk im Laufwerk ist, ob das Laufwerk bereit ist, oder ob sich der Schreib-/Lesekopf auf Track 0 befindet.

8.11.1: Das Drive-Select-Register

Mit dem Drive-Select-Register kann eines der vier Laufwerke ausgewählt, der Schreib-/Lesekopf bewegt, der obere oder untere Schreib-/Lesekopf ausgewählt, und die Richtung der Bewegung angegeben werden. Alle Signale, außer die Angabe der Bewegungsrichtung, sind low-aktiv. Zu beachten ist bei diesem Register der Vorgang zum Bewegen des Schreib-/Lesekopfes. Das Bit /DISKSTEP führt erst eine Positionierung des Schreib/Lesekopfes aus, wenn das Bit durch einen Flankenwechsel von logisch 1 auf logisch 0 gesetzt wird. Nach Beendigung der Positionierung sollte dieses Bit immer auf Eins gesetzt werden.

Bit	Name	Bedeutung
7	/DSKMOTOR	Dieses Bit kontrolliert den Motor des Laufwerks. Wenn dieses Bit gleich Null ist, wenn ein Diskettenlaufwerk selektiert wird, schaltet sich der jeweilige Motor an.

Bit	Name	Bedeutung
6	/DSKSEL3	Mit diesem Bit kann das Laufwerk 3 selektiert werden. Ist das Bit gleich Null, so ist das Laufwerk selektiert.
5	/DSKSEL2	Mit diesem Bit kann das Laufwerk 2 selektiert werden. Ist das Bit gleich Null, so ist das Laufwerk selektiert.
4	/DSKSEL1	Mit diesem Bit kann das Laufwerk 1 selektiert werden. Ist das Bit gleich Null, so ist das Laufwerk selektiert.
3	/DSKSEL0	Mit diesem Bit kann das Laufwerk 0 selektiert werden. Ist das Bit gleich Null, so ist das Laufwerk selektiert.
2	/DSKSIDE	Diese Bit wählt den Schreib-/Lesekopf aus. Ist das Bit gleich Null, so wird Kopf 1 (oben) ausgewählt.
1	DSKDIREC	Wenn dieses Bit gleich Eins ist, so wird der Schreib-/Lesekopf nach außen bewegt. Ist es Null, so wird der Schreib-/Lesekopf nach innen bewegt.
0	/DISKSTEP	Mit diesem Bit kann der Schreib-/Lesekopf bewegt werden. Dies geschieht durch einen Flankenwechsel von high auf low.

8.11.2: Das Drive-Status-Register

Mit dem Drive-Status-Register kann, wie der Name schon sagt, der Status der Disk überprüft werden. Mit verschiedenen Bits kann getestet werden, ob der Kopf auf Track 0 steht, ob sich eine Diskette im Floppy befindet oder ob die Disk schreibgeschützt ist. Dieses Register wird auch noch für andere Anwendungen benötigt. Wir haben hier nur die für uns wichtigen Floppy-Funktionen herausgenommen. Alle Signale sind low-aktiv.

Bit	Name	Bedeutung
5	/DSKRDY	An diesem Bit kann erkannt werden, ob das Laufwerk bereit ist. Ist das Laufwerk bereit, ist dieses Bit gleich 0.
4	/DSKTRACK0	Signalisiert, ob der Schreib-/Lesekopf auf Track 0 befindet.

Bit	Name	Bedeutung
3	/DSKPROT	Zeigt an, ob die Diskette, die sich im Laufwerk befindet, schreibgeschützt ist.
2	/DSKCHANGE	Mit diesem Bit kann getestet werden, ob sich eine Diskette im Laufwerk befindet.

```
/*****************************

Disk-Demonstration
last update 16/02/88
von Frank Kremser und Jörg Koch
© Markt & Technik 1988

*****************************

Diese Demonstration verdeutlicht den einfachen Zugriff auf die
Floppies.
Es wird eine Datei in die ersten zwei Sektoren, die Bootsektoren,
geschrieben. Besitzt das Programm die im Buch gezeigten Merkmale,
so wird das Programm beim Booten aufgerufen. Das Programm darf maximal
1024 Byte lang sein. Es ist aber recht leicht möglich, auch den
gesamten ersten Track für Bootblocks zu verwenden, so daß 5632 Byte
zur Verfügung stehen.

*****************************/

#include <exec/types.h>                 /* Include-Files laden */
#include <exec/memory.h>
#include <exec/ports.h>
#include <exec/devices.h>
#include <exec/io.h>
#include <exec/libraries.h>
#include <devices/trackdisk.h>
#include <libraries/dos.h>
#include <libraries/dosextens.h>

extern struct MsgPort *CreatePort(); /* MsgPort extern deklarieren */
extern struct FileHandle *Open();

struct IOExtTD *request;
struct MsgPort *port;
struct FileHandle *fileh;

BYTE *spbuff;
LONG count;

main() /* HAUPTPROGRAMM */
 {                                  /* Pufferspeicher bereitstellen */
```

```
43    spbuff = (BYTE *)AllocMem(1024,MEMF_CLEAR|MEMF_CHIP);
44                                        /* Datei mit Prg öffnen */
45    fileh = Open("BOOTBLOCK",MODE_OLDFILE);
46    Read(fileh,spbuff,36);          /* Die ersten 36 Byte wegschmeißen, da */
47                                    /* der SEKA-Assembler diese hinzugefügt hat */
48    Read(fileh,spbuff,1024);        /* Prg in Puffer lesen */
49    Close(fileh);                   /* Datei schließen */
50
51    opendev();                      /* Trackdisk-Device öffnen */
52    checksum();                     /* Checksumme der Bootsektoren berechnen */
53    wrboot();                       /* und Bootsektoren schreiben */
54
55    CloseDevice(request);           /* Trackdisk-Device schließen */
56    FreeMem(spbuff,1024);           /* Datenpuffer löschen */
57   }
58
59  opendev() /* Routine zum Öffnen der Trackdisk-Device */
60
61   {
62    struct IORequest *io;
63
64    port = CreatePort(0,0); /* ReplyPort erstellen */
65    if (port == 0) exit();
66
67    io = (struct IORequest *) /* Requester initialisieren */
68        AllocMem(sizeof(struct IOExtTD),MEMF_CLEAR|MEMF_PUBLIC);
69    io->io_Message.mn_Node.ln_Type = NT_MESSAGE;
70    io->io_Message.mn_Length = sizeof(struct IOExtTD);
71    io->io_Message.mn_ReplyPort = port;
72
73    request = (struct IOExtTD *) io;
74
75    OpenDevice(TD_NAME,0,request,0); /* Device oeffnen */
76   }
77
78  motoron() /* Motor des Laufwerks einschalten */
79   {
80    request->iotd_Req.io_Length = 1; /* Motor ein */
81    request->iotd_Req.io_Command = TD_MOTOR; /* Motor-Befehl */
82    DoIO(request);                          /* Befehl ausfuehren */
83   }
84
85  motoroff() /* Motor des Laufwerks ausschalten */
86   {
87    request->iotd_Req.io_Length = 0; /* Motor aus */
88    request->iotd_Req.io_Command = TD_MOTOR; /* Motor-Befehl */
89    DoIO(request);                          /* Befehl ausfuehren */
90    }
91
92    counter() /* Disketten-Identität ermitteln */
```

```
 93 {
 94   request->iotd_Req.io_Command = TD_CHANGENUM; DoIO(request);
 95   count = request->iotd_Req.io_Actual;
 96 }
 97
 98 wrboot() /* Bootsektoren schreiben */
 99 {
100   counter(); /* Disk-Identität ermitteln */
101   motoron(); /* Motor einschalten */
102   request->iotd_Req.io_Length = 1024; /* Länge der beiden Bootsek. */
103   request->iotd_Req.io_Data = (APTR)spbuff;
104                                          /* Zeiger auf Datenpuffer */
105   request->iotd_Req.io_Offset = 0; /* Sektoroffset */
106   request->iotd_Count = count; /* Disk-Identität */
107   request->iotd_req.io_Command = TD_PROTSTATUS; /*Prüfen, ob Disk*/
108   DoIO(request);                              /* schreibgeschützt */
109   if(request->iotd_Req.io_Actual == 0)
110    {
111     request->iotd_Req.io_Command = CMD_WRITE; /* Daten in Puffer */
112     DoIO(request);                                  /* schreiben */
113     request->iotd_Req.io_Command = CMD_UPDATE; /* Daten auf Disk */
114     DoIO(request);                                  /* schreiben */
115    }
116   motoroff(); /* Motor ausschalten */
117 }
118 checksum()
119 {
120   ULONGLONG check, *buf;
121   int i;
122
123   check = 0;
124   buf = (ULONG *) spbuff;
125
126   for (i = 0; i < 256; i++)
127    {
128     if (i != 1)
129      {
130       if (i != 0)
131         if ((0x0FFFFFFFF - check) < buf[i]) check += 1;
132       check += buf[i];
133      }
134    }
135   buf[1] = 0x0FFFFFFFF - check;
136 }
```

```
1  ;*******************************
2  ;
3  ; DOS-Bootblockroutine
4  ; last update 16/02/88
5  ; von Frank Kremser und Jörg Koch
6  ; © Markt & Technik 1988
7  ;
8  ;*******************************
9  ;
10 ;Diese Routine stellt einen speziellen DOS-Bootblock dar. Der
11 ;erste Teil wird benötigt, um DOS zu installieren, anschließend
12 ;steht eine eigene Routine, die beliebig ausgewechselt werden kann.
13 ;Es muß dabei aber darauf geachtet werden, das das Objekt-File
14 ;dann nicht länger als 1024 Byte, also zwei Sektoren ist.
15 ;
16 ;*******************************
17
18          dc.b    "DOS",0             ;Erkennungsmarke "DOS"-Diskette
19          dc.l    0                   ;Checksumme (wird
20          dc.l    370                 ;später nachgetragen)
21
22          move.l  4,a6
23          lea.l   dosname(pc),a1      ;"dos.library"
24          jsr     -96(a6)             ;FindResident();
25          move.l  d0,a0               ;a0:=d0->DOS-Modul
26          move.l  22(a0),a0           ;a0->DOS-Initialisierungsroutine
27
28 ;-----------------------------------------------------
29 ; Dies war der erste Teil der DOS-Bootblock-Routine.
30 ; Ab hier kann ein eigenes Programm eingefügt werden.
31 ;-----------------------------------------------------
32
33          movem.l d0-d7/a0-a6,-(a7) ;Register retten
34          lea.l   $50000,a0         ;a0:=ab $5000
35          move.l  #6645,d0          ;d0:=6645 Longwords
36 clear:    clr.l   (a0)+             ;löschen
37          dbf     d0,clear          ;dekrementiere, teste
38                                    ;d0:=0, nein dann clear
39
40          bsr     copperinit        ;Copperliste initialisieren
41
42          jsr     -132(a6)          ;Multitasking abschalten
43          lea.l   $dff000,a5        ;a5:=Customchipbase
44          move.w  #$03e0,$96(a5)    ;DMA-Control write
45                                    ;BitPlane-DMA disable,
46                                    ;Copper-DMA disable,
47                                    ;Blitter-DMA disable,
48                                    ;Sprite-DMA disable
49
50          move.l  #$53000,$80(a5)   ;Neue Copperliste ab $53000
```

```
51          clr.w   $88(a5)             ;Copper mit neuer Adresse starten
52          move.l  #$1d7133d1,$8e(a5) ;DIWSTRT/DIWSTOP
53                                      ;Start- und Endposition des
54                                      ;darzustellenden Windows
55          move.l  #$003000f8,$92(a5) ;DDFSTRT/DDFSTOP
56                                      ;DataFetchStart/Stop
57          move.w  #$1100,$100(a5);    ;BPLCON0
58          clr.l   $102(a5)            ;BPLCON1
59          clr.l   $108(a5)            ;BPL1MOD
60          move.w  #$8380,$96(a5)      ;DMA control write
61                                      ;Bit-Plane DMA enable,
62                                      ;Coprocessor DMA enable
63
64          move    #10,d5
65 vpos:    move.l  4(a5),d2            ;d2 := VPOSR
66          and.l   #$0001ff00,d2       ;Ist Zeile $10
67          cmpi.l  #$00001000,d2       ;erreicht
68          bne     vpos                ;nein, dann warte weiter
69
70          move.l  d5,d0
71          bsr     put                 ;ja, dann hole und
72                                      ;zeichne brush
73          andi.b  #64,$bfe001         ;Ist linke Maustaste
74          beq     ende                ;gedrückt, wenn ja, dann ende
75
76 loop1:   add.w   d(pc),d5
77          lea.l   d(pc),a3
78          cmp.w   #1,(a3)             ;Wird nach oben bewegt,
79          bne     loop2               ;dann entsprechende Routine aufrufen
80
81          cmpi.w  #210,d5             ;Ist Brush schon am unteren Rand
82          bne     vpos                ;nein, dann weiter
83
84 loop2:   move    #$ffff,(a3)         ;Ist Brush schon am oberen Rand
85          cmpi.w  #1,d5
86          bne     vpos                ;nein, dann weiter
87
88          move    #1,(a3)
89          bra     vpos
90
91 ende:    lea.l   gfxname(pc),a1
92          jsr     -408(a6)            ;OldOpenLibrary();
93          move.l  d0,a4               ;alte Copperliste holen
94          move.l  38(a4),$80(a5)      ;und in COP1LCH setzen
95          clr.w   $88(a5)             ;und starten
96          move.w  #$8060,$96(a5)      ;Sprite- und Blitter-DMA
97                                      ;setzen
98          jsr     -138(a6)            ;Multitasking ein
99
100         movem.l (a7)+,d0-d7/a0-a6 ;Register zurückgeben
```

```
101           movem.l a7)+,d0-d7/a0-a6  ;Register zurückgeben
102           move    #$8200,$96(A5)    ;DMA einschalten
103           clr.l   d0
104           rts                       ;Rückkehr
105
106 copperinit:
107           clr     dl
108           lea.l   $53000,a0         ;Startadresse für Copperliste
109           move.l  #$000ffffe,(a0)+  ;Warten, bis Zeile $0 erreicht
110           move.l  #$00e00005,(a0)+  ;BitPlanel Zeiger setzen
111           move.l  #$00e20000,(a0)+
112           move.l  #$01800000,(a0)+  ;Hintergrundfarbe auf schwarz setzen
113           lea.l   coltab(pc),al     ;Zeiger auf Farbtabelle setzen
114           move    #1,d0             ;Ab Zeile 1 Waits setzen
115 loop3:     move.b  d0,(a0)+          ;VP setzen
116           move.b  #15,(a0)+         ;HP + Wait-Flag setzen
117           move.l  #$fffe0182,(a0)+  ;WAIT und COLOR1
118           clr     d6
119           move.b  (al,dl),d6        ;aktuelle Farbe aus Tabelle holen
120           eori    #$0ff0,d6         ;und umwandeln
121           move    d6,(a0)+          ;und in CListe schreiben
122           addq    #1,dl
123           cmpi    #32,dl            ;Sind alle Farben gesetzt?
124           bne     loop4             ;Wenn nein, dann weiter
125           clr.l   dl                ;Sonst Farbindex zurücksetzen
126 loop4:     addq    #1,d0             ;VP-Index um eins erhöhen
127           cmpi    #256,d0           ;Prüfen, ob alle Zeilen gesetzt
128           bne     loop3             ;Wenn, dann weiter
129           move.l  #$fffffffe,(a0)+  ;Endekennzeichnung der Copperliste
130           rts
131
132 put:      mulu    #44,d0            ;Addresse der BitPlanes
133           add.l   #$5000E,d0        ;ermitteln
134           move.l  d0,a0
135           lea.l   brush(pc),a2
136           move    #25,d2            ;Höhe des Brush's
137 ploop2:    move.l  a0,al
138           move    #4,d3             ;1 Longword = 4 Byte
139 ploop3:    move.l  (a2),(a0)+        ;Brush setzen
140           move.l  (a2)+,(al)+
141           dbf     d3,ploop3
142           add.l   #24,a0
143           dbf     d2,ploop2
144           rts
145
146 brush:    dc.l  $00000000,$00000000,$00000000,$00000000,$00000000
147           dc.l  $00000000,$00000000,$00000000,$00000000,$00000000
148           dc.l  $00040383,$e781fc00,$40000000,$00000000,$00000000
149           dc.l  $000c01c7,$c3070c00,$c0000000,$0000e280,$00000000
150           dc.l  $001c01c7,$830c0801,$c0000000,$00002280,$00000000
```

```
151          dc.l  $003c02c9,$83180003,$c0000000,$00002300,$187e07e0
152          dc.l  $006c02c9,$86180006,$c0000000,$0000a280,$187e07e0
153          dc.l  $00cc04d3,$0618000c,$c0000000,$00004aa0,$31819818
154          dc.l  $018c04d3,$06187818,$c0000000,$00000000,$01819818
155          dc.l  $03fc08e3,$0618303f,$c0000000,$00000000,$01819818
156          dc.l  $060c08e6,$0c0c3060,$c0000000,$0000e280,$01819818
157          dc.l  $0c0c18c6,$0c0e60c0,$c0000000,$00008280,$007e07e0
158          dc.l  $3f3f3c9f,$1e03c3f3,$f0000000,$0000c300,$007e07e0
159          dc.l  $00000000,$00000000,$00000000,$00008280,$01801800
160          dc.l  $00000000,$00000000,$00000000,$00008aa0,$01819818
161          dc.l  $633f9f87,$8619fcfc,$3e7e18c3,$cc600000,$01819818
162          dc.l  $633f9f87,$8619fcfc,$3e7e18cf,$cc600000,$01819818
163          dc.l  $63319866,$66198cc3,$306198cc,$0c60891f,$007e07e0
164          dc.l  $7f319866,$66d98cc3,$387f98cc,$0fe0da84,$007e07e0
165          dc.l  $7f3f9f86,$66d9fcfc,$387e18cc,$0fe0aa84,$00000000
166          dc.l  $633f9f86,$67f9fcfc,$306198cc,$0c608944,$00000000
167          dc.l  $633199c7,$87f98cce,$3e7f9fcf,$cc608a84,$00000000
168          dc.l  $633198c7,$86d98cc6,$3e7f0703,$cc608944,$00000000
169          dc.l  $00000000,$00000000,$00000000,$00000000,$00000000
170          dc.l  $00000000,$00000000,$00000000,$00000000,$00000000
171          dc.l  $00000000,$00000000,$00000000,$00000000,$00000000
172
173 coltab:  dc.b  $00,$10,$20,$30,$40,$50,$60,$70,$80,$90
174          dc.b  $a0,$b0,$c0,$d0,$e0,$f0,$f0,$e0,$d0,$c0
175          dc.b  $b0,$a0,$90,$80,$70,$60,$50,$40,$30,$20,$10,0
176
177 d:       blk.w 1,1                    ;Zähler für Bewegung des Brush's
178 gfxname: dc.b  "graphics.library",0
179 dosname: dc.b  "dos.library",0


 1  ;*******************************
 2  ;
 3  ; 1.Floppy - Demonstration
 4  ; last update 10/03/88
 5  ;von Frank Kremser und Jörg Koch
 6  ; © Markt & Technik 1988
 7  ;
 8  ;*******************************
 9  ;
10  ;Diese Demonstration fährt den Schreib-/Lesekopf des 1. Laufwerks
11  ;auf Track 0 zurück.
12  ;
13  ;*******************************
14           CIA_A = $bfe001
15           CIA_B = $bfd100
16
17  spurn:    btst    #4,CIA_A           ;Kopf auf Spur 0?
18            beq     ende               ;Wenn ja, dann beenden
19            move.b  #%01110010,CIA_B   ;Step Richtung Track 00, drive 0
```

```
20          move.l  #250,d0             ;Warteschleife
21  wait1:  tst.l   (a6)                ;Dummy-Befehl zur Verzögerung
22          dbra    d0,wait1
23          move.b  #%01110011,CIA_B    ;Bit 0 von CIA B muß erst 0,
24                                      ;dann 1 gesetzt werden, damit der
25                                      ;Head bewegt wird
26          move.l  #250,d0             ;Warteschleife
27  wait2:  tst.l   (a6)                ;Dummy-Befehl
28          dbra    d0,wait2
29          bra     spurn               ;Nocheinmal
30  ende:   rts                         ;Rückkehr


 1  ;*******************************
 2  ;
 3  ; 2.Floppy-Demonstration
 4  ; last update 16/02/88
 5  ;von Frank Kremser und Jörg Koch
 6  ; © Markt & Technik 1988
 7  ;
 8  ;*******************************
 9  ;
10  ;Diese Demonstration liest einen kompletten Track ein, decodiert ihn
11  ;aber nicht, so daß noch alle Steuer- und Synchronisierdaten vor-
12  ;handen sind.
13  ;
14  ;*******************************
15
16        Device     = 350
17        Port       = 36               ;Offset für Laufwerk 0
18        ReplyPort  = 174
19        SigTask    = 16
20        Task       = 276
21        FindName   = -276
22
23        Anzahl     = $397c
24        Track      = 20
25
26                  move.l  $4,a6             ;ExecBase setzen
27                  lea     Name,a1           ;Zeiger auf Trackdisk-Name se
28                  lea     Device(a6),a0     ;Zeiger auf Deviceliste setze
29                  jsr     FindName(a6)      ;Trackdisk-Device suchen
30                  tst.l   d0                ;Wenn nicht gefunden,
31                  beq     Error             ;dann Fehler
32                  move.l  Task(a6),a0       ;Zeiger auf Task ermitteln
33                  move.l  d0,a6
34                  move.l  Port(a6),a3       ;Laufwerk 0 setzen
35                  lea     ReplyPort(a3),a1  ;ReplyPort setzen
36                  move.l  SigTask(a1),-(a7) ;Task retten
37                  move.l  a1,-(a7)          ;ReplyPort retten
```

```
38                move.l  a0,SigTask(a1)      ;eigenen Task eintragen
39                bset    #0,34(a3)           ;Trackdisk-Task stoppen
40
41                move.l  #1,d0
42                jsr     $fea462             ;Motor einschalten
43                move.l  #Track,d0
44                move.w  #Track,74(a3)       ;Track eintragen
45                jsr     $fea3da             ;Kopf auf Position fahren
46                move.l  78(a3),a0           ;Zeiger auf Lesepuffer setzen
47                move.l  #Anzahl,d0          ;Anzahl der zu lesenden Bytes
48                jsr     $fea524             ;Track einlesen
49                clr.l   d0
50                jsr     $fea462             ;Motor abschalten
51                bclr    #0,34(a3)           ;Task wieder freigeben
52                move.l  (a7)+,a1            ;Zeiger von Task holen
53                move.l  (a7)+,SigTask(a1)   ;und eintragen
54        Error:  rts                         ;Rückkehr
55
56        Name:   dc.b    trackdisk.device',0


 1        ;*******************************
 2        ;
 3        ; 3.Floppy-Demonstration
 4        ; last update 16/02/88
 5        ;von Frank Kremser und Jörg Koch
 6        ; © Markt & Technik 1988
 7        ;
 8        ;*******************************
 9        ;
10        ;Diese Demonstration liest einen Track ein und decodiert diesen,
11        ;so
12        ;daß die Daten in lesbarer Form vorliegen.
13        ;
14        ;*******************************
15
16        Device      =350
17        Port        =36                 ;Offset für Laufwerk 0
18        ReplyPort   =174
19        SigTask     =16
20        Task        =276
21        FindName    =-276
22
23        Track       =00
24        Ziel        =$50000
25
26            move.l  $4,a6               ;ExecBase
27            lea     Name,a1             ;Zeiger auf Trackdisk-Name setzen
28            lea     Device(a6),a0       ;Zeiger auf Device-Liste setzen
29            jsr     FindName(a6)        ;Trackdisk-Device suchen
```

```
30              tst.l   d0                  ;Wenn Device nicht
31              beq     Error               ;gefunden, dann beenden
32              move.l  Task(a6),a0         ;Zeiger auf eigenen Task holen
33              move.l  d0,a6               ;Zeiger auf Task nach a6
34              move.l  Port(a6),a3         ;Zeiger auf Laufwerkport holen
35              lea     ReplyPort(a3),al    ;Zeiger auf ReplyPort holen
36              move.l  SigTask(al),-(a7)   ;Zeiger auf Task sichern
37              move.l  al,-(a7)            ;Zeiger auf ReplyPort sichern
38              move.l  a0,SigTask(al)      ;eigenen Task eintragen
39              bset    #0,34(a3)           ;Trackdisk-Task stoppen
40
41              move.l  #Track,d0           ;Tracknummer
42              move.w  d0,74(a3)           ;in Structure eintragen
43              move.l  78(a3),a2           ;Zeiger auf Datenpuffer
44              move.w  d0,(a2              ;Tracknummer eintragen
45              bclr    #0,2(a2)
46              clr.b   66(a3)              ;Fehlerzahl löschen
47              jsr     $fea99e             ;Track lesen
48              clr.l   d0
49              jsr     $fea462             ;Motor ausschalten
50              move.b  3(a2),d0            ;erste Blocknummer
51              cmp.b   #$0b,d0             ;Prüfen, ob Nummer größer als 11
52              bcc     Ende                ;Wenn ja, dann Fehler
53              move.b  #$0b,d6             ;Sektoranzahl
54              clr.l   d0                  ;Sektor Null
55              sub.b   3(a2),d0
56              bpl     loopl
57              addi.b  #$0b,d0             ;Adresse des Blockes
58  loopl:      mulu    #$440,d0            ;Null errechnen
59              lea     1664(a2),a4         ;Zeiger auf Datenanfang
60              adda.l  d0,a4               ;Zeiger auf Block Null
61              lea     Ziel,a              ;Zeiger auf Ziel setzen
62              clr.l   d7                  ;Anfang bei Sektor Null
63  loop3:      lea     64(a4),al           ;Zeiger auf Datenblock
64              move.l  a5,a0               ;Ziel nach a0
65              move.l  #$200,d0            ;Anzahl der zu decodierenden Daten
66              jsr     $feacb2             ;Daten decodieren
67              adda.l  #$200,a5            ;Zielzeiger erhöhen
68              sub.b   #1,d6               ;Anzahl der Blöcke dekrementieren
69              beq     Ende                ;wenn fertig, dann Ende
70              add.b   #1,d7               ;Blocknummer erhöhen
71              cmp.b   3(a2),d7            ;Anfang des Puffers
72              bne     loop2               ;wenn nein, dann weiter
73              lea     1664(a2),a4         ;Zeiger auf Pufferanfang setzen
74              bra     loop3               ;weiter decodieren
75  loop2:      add.l   #$440,a4            ;Zeiger auf nächsten Block
76              bra     loop3               ;weiter decodieren Ende:
77              bclr    #0,34(a3)           ;Task wieder freigeben
78              move.l  (a7)+,al            ;Zeiger auf Task holen
79              move.l  (a7)+,SigTask(al)   ;und in Port eintragen
```

```
80  Error:    rts                          ;Rückkehr
81
82  Name:     dc.b    trackdisk.devic',0


 1  ;*******************************
 2  ;
 3  ; 4.Floppy-Demonstration
 4  ; last update 16/02/88
 5  ;von Frank Kremser und Jörg Koch
 6  ; © Markt & Technik 1988
 7  ;
 8  ;*******************************
 9  ;
10  ;Diese Demonstration schreibt einen kompletten Track auf Diskette.
11  ;
12  ;*******************************
13
14        Device      =350
15        Port        =36                ;Offset für Laufwerk 0
16        ReplyPort   =174
17        SigTask     =16
18        Task        =276
19        FindName    =-276
20
21        Adresse     =$50000
22        Track       =20
23
24            move.l  $4,a6              ;ExecBase
25            lea     Name,a1            ;Zeiger auf Trackdisk-Name setzen
26            lea     Device(a6),a0      ;Zeiger auf Device-Liste setzen
27            jsr     FindName(a6)       ;Trackdisk-Device suchen
28            tst.l   d0                 ;wenn nicht gefunden,
29            beq     Error              ;dann beenden
30            move.l  Task(a6),a0        ;Zeiger auf eigenen Task suchen
31            move.l  d0,a6              ;Zeiger nach a6
32            move.l  Port(a6),a3        ;Zeiger auf Laufwerksport holen
33            lea     ReplyPort(a3),a1   ;Zeiger auf ReplyPort
34            move.l  SigTask(a1),-(a7)  ;Zeiger auf Task sichern
35            move.l  a1,-(a7)           ;Zeiger auf ReplyPort sichern
36            move.l  a0,SigTask(a1)     ;eigenen Task eintragen
37            bset    #0,34(a3)          ;Trackdisk-Task stoppen
38
39            move.l  #1,d0
40            jsr     $fea462            ;Motor einschalten
41            move.l  #Track,d2          ;Tracknummer nach d2
42            move.l  d2,d0              ;und d0
43
44            jsr     $fea3da            ;Kopf positionieren
45            lea     Adresse,a5         ;Zeiger auf Puffer
```

```
46
47              move.l  $52(a3),a2          ;Zeiger auf Schreibpuffer
48              lea     4(a2),a2
49              move.w  #$fff,d0            ;Zählwert löschen
50              move.l  #$aaaaaaaa,d1       ;Löschwert
51  loop1:      move.l  d1,(a2)+            ;Schreibpuffer löschen
52              dbra    d0,loop1            ;solange, bis fertig
53              lea     82(a3),a2           ;Zeiger auf Schreibpuffer
54              lea     1664(a2),a2         ;Zeiger auf Datenanfang
55              moveq   #$0b,d4             ;Blockanzahl
56              moveq   #0,d5               ;Blockzähler auf Null
57  loop2:      move.l  #$ff000000,d0       ;DOS-Kennung des Headers
58              move.l  d5,d1               ;Blocknummer nach d1
59              lsl.l   #8,d1               ;Nummer auf richtige Pos. schieben
60              or.l    d1,d0               ;und in Header eintragen
61              or.l    d4,d0               ;Anzahl der Blöcke bis zur Lücke
62              move.l  d2,d1               ;Tracknummer nach d1
63              swap    d1                  ;Nummer auf richtige Position
64              or.l    d1,d0               ;Tracknummer eintragen
65              move.l  a2,a1               ;Schreibpuffer nach a1
66              move.l  a5,a0               ;Zeiger auf Daten nach a0
67              jsr     $feaadc             ;Daten codieren und in Schreibpuffer
68              addq.l  #1,d5               ;Blockzähler erhöhen
69              adda.l  #$440,a2            ;Zeiger auf nächsten Block im Puffer
70              adda.l  #$200,a5            ;Zeiger auf nächsten Datenteil
71              subq.l  #1,d4               ;Anzahl der Blöcke dekrementieren
72              bne.s   loop2               ;weiter, bis fertig codiert
73              lea     $52(a3),a0          ;Zeiger auf Schreibpuffer
74              lea     4(a0),a0
75              move.w  #$353e,d0           ;Anzahl der Schreibdaten
76              jsr     $fea5b4             ;Track schreiben
77              move.l  #0,d0
78              jsr     $fea462             ;Motor abschalten
79
80              bclr    #0,34(a3)           ;Task wieder freigeben
81              move.l  (a7)+,a1            ;Zeiger auf Task holen
82              move.l  (a7)+,SigTask(a1)   ;und in Port eintragen
83  Error:      rts                         ;Rückkehr
84
85  Namer:      dc.b    trackdisk.device,0
```

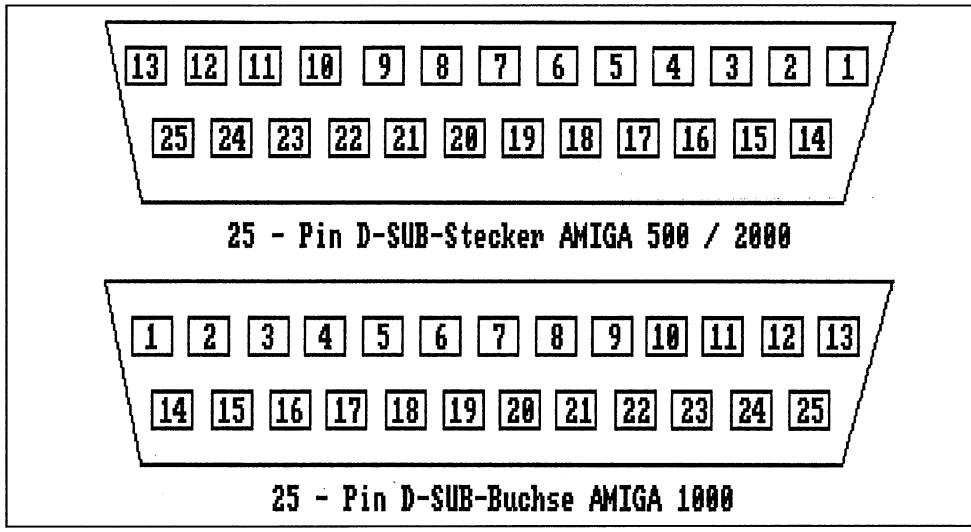
Kapitel 9

Die Schnittstellen

Die Schnittstellen stellen den Draht zur Außenwelt dar. Bei Computern früherer Generationen waren Standard-Schnittstellen Luxus, der separat und teuer nachgekauft werden mußte. Der Amiga besitzt gleich zwei Standard-Schnittstellen ab Werk. Eine *serielle Schnittstelle* zum Übertragen von Daten per Modem oder zum Betrieb eines Plotters und eine *parallele Schnittstelle* für den Betrieb eines Druckers. Während die parallele Schnittstelle überwiegend durch die CIA-Portbausteine gesteuert wird, übernimmt bei der seriellen Schnittstelle Paula die Kontrolle.

9.1: Die parallele Schnittstelle

Die parallele Schnittstelle des AMIGA, auch als *Centronics*-Schnittstelle bezeichnet, hat sich im Lauf der Zeit etwas verändert. Die ausgelieferten 1000er besitzen im Vergleich zu den 500er und 2000er eine nicht-IBM-kompatible-Schnittstelle. Dies erkennt man daran, daß anstatt der DSUB-Buchse ein Stecker verwendet wird. Beim Verwenden von Druckerkabel stößt der Amiga-1000-Besitzer daher auf große Probleme, da an Pin 23 der Centronics-Schnittstelle des A1000 +5V anliegt, der bei handelsüblichen Kabeln mit Masse verbunden ist, welchen zu einem fatalen Kurzschluß beim Amiga 1000 führen kann. Hier ist man gezwungen, ein Kabel selbst zu löten. Beim Amiga 500 und 2000 können normale IBM-Druckerkabel verwendet werden.



Z 9.1-1: *Diese Abbildung zeigt die Pin-Belegung des D-SUB-Steckers des A1000 und der D-SUB-Buchse des A500 und 2000*

Belegt sind die Pins des Steckers bzw. der Buchse wie folgt:

Pin	A1000	A500/2000	Ein-/Ausgang
1	/DRDY	/STROBE	Ausgang
2	Data 0	Data 0	Ein-/Ausgang
3	Data 1	Data 1	Ein-/Ausgang
4	Data 2	Data 2	Ein-/Ausgang
5	Data 3	Data 3	Ein-/Ausgang
6	Data 4	Data 4	Ein-/Ausgang
7	Data 5	Data 5	Ein-/Ausgang
8	Data 6	Data 6	Ein-/Ausgang
9	Data 7	Data 7	Ein-/Ausgang
10	/ACK	/ACK	Eingang
11	BUSY	BUSY	Ein-/Ausgang
12	POUT	POUT	Ein-/Ausgang
13	SEL	SEL	Ein-/Ausgang
14	GND	+5V	
15	GND	NC	
16	GND	/RESET	Ausgang
17	GND	GND	
18–22	GND	GND	
23	+5V	GND	
24	NC	GND	
25	/RESET	GND	Ausgang

Die Beschreibung der Signale:

/DRDY, /STROBE	Mit Strobe bzw. Data Ready wird signalisiert, daß die Daten bereit sind.
Data0–Data7	8 Datenbits, über die der Daten-Transfer läuft.
/ACK	Dieses Signal dient als Datenübernahmesignal.
BUSY	Busy signalisiert, daß der Drucker beschäftigt ist.
POUT	Paper Out – Papierende-Signal.
SEL	Select signalisiert, daß der Drucker On-Line ist.
+5V	+5V Spannungsversorgung.
/RESET	Gepufferte Reset-Leitung des AMIGA.

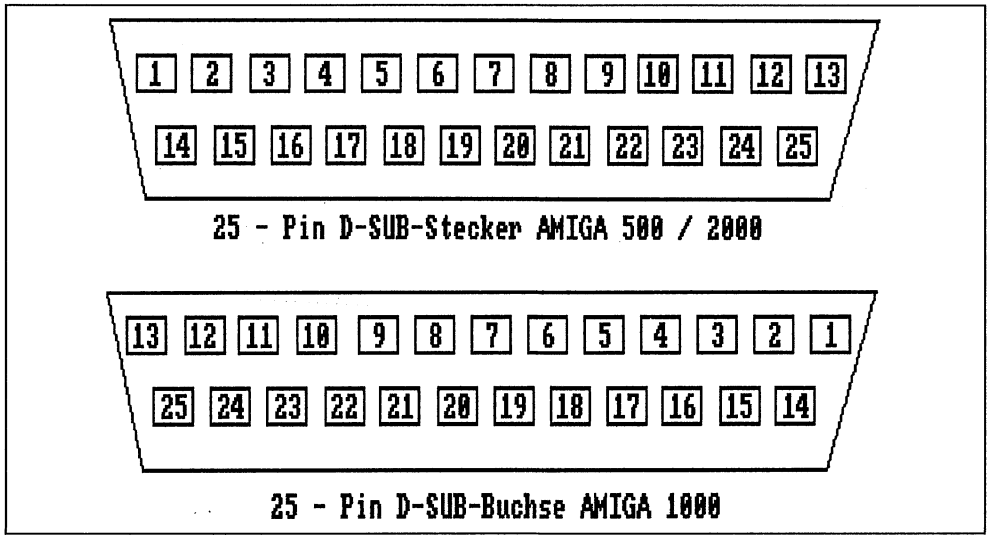
Bei der Centronics liegt das Datenbyte an den 8 Datenleitungen an. Dadurch wird eine parallele Übertragung (daher kommt auch der Name Parallel-Schnittstelle) erreicht. Neben diesen Datenleitungen stehen verschiedene Leitungen zur Steuerung bereit. Die Busy-Leitung signalisiert dem Amiga, daß der Drucker (Druckerpuffer) keine Daten mehr annehmen kann und der Amiga mit dem Senden der Daten warten soll. Ebenso steuert auch der Drucker das Papier-Ende-Signal POUT und das SEL-Signal, welches anzeigt, daß der Drucker ON-Line (SEL logisch 1) oder OFF-Line (SEL logisch 0) ist. Strobe und Acknowledge dienen zum Datentransfer. Hat der Amiga ein gültiges Byte auf den Datenleitungen, so wird die Strobe-Leitung auf Null gelegt. Der Drucker erkennt dies und quittiert den Empfang der Daten mit dem Acknowledge-Signal. Erst wenn der Amiga dieses Signal erhält, wird das nächste Byte auf die Datenleitungen gelegt. Die Parallel-Schnittstelle kann über die CIA-A und CIA-B angesprochen werden. Während die CIA-A das Übertragen der Datenbits der parallelen Schnittstelle übernimmt, dient die CIA-B für Steuerungszwecke. Hier stehen verschiedene Bits zur Verfügung, die die Signalzustände signalisieren. CIA-A übernimmt das /STROBE- und die Datenbits, CIA-B die Steuerleitungen Busy, PaperOut und Select. Die parallele Schnittstelle eignet sich nicht nur für den Betrieb eines Druckers. Sie ist auch ideal für den Betrieb eines Sound-Digitizers, EPROMer oder anderer ähnlicher Anwendungen geeignet. Die Adressen, um die einzelnen Bits der Register abfragen zu können, finden Sie im Anhang D. Ein praktisches Beispiel für die Entfremdung des Parallel-Ports zeigt der M&T-Sounddigitizer, den Sie im Kapitel Hardware-Erweiterungen finden.

```
 1  ;*******************************
 2  ;
 3  ; Parallel - Demonstration
 4  ; last update 16/02/88
 5  ;von Frank Kremser und Jörg Koch
 6  ; © Markt & Technik 1988 ;
 7  ;*******************************
 8  ;
 9  ;Diese Demonstration zeigt die Programmierung des Parallel-Ports.
10  ;Dieses Programm stellt gleichzeitig die Software zum Digitalisierer
11  ;dar. Will man den Sound nicht sofort abspielen, sondern nach einem
12  ;Digitalisierungsdurchlauf stoppen und dann auf Diskette speichern,
13  ;muß man nur die mit '#' gekennzeichneten Zeilen löschen.
14  ;
15  ;*******************************
16
17        buffer=$50000                 ;Adresse des Soundpuffers
18        len   =$ffff                  ;Länge des Puffers in Bytes
19                                      ;              (max. $1fffe)
20        speed=10                      ;Abspielrate
21
22        PRA   =$bfd000                ;CIA-Register für Parallelport
23        PRB   =$bfe101
24        DDRA  =$bfd200
25        DDRB  =$bfe301
26
27            move.l  #buffer,a0        ;Puffer löschen
28            move.l  #len,d0
29  clear:    move.b  #0,(a0)+
30            dbra    d0,clear
31
32            move.l  4,a6
33            lea     super,a5          ;In den Supervisormodus schalten,
34            jmp     -30(a6)
35  super:    addq.l  #8,a7
36            move    #$2700,sr         ;um Interrupts sperren zu können
37            move.b  #%00000000,DDRB   ;Parallel-Port zurücksetzen
38            move.b  DDRA,d0           ;und initialisieren
39            andi.b  #%11111010,d0
40            ori.b   #%00000100,d0
41            move.b  d0,DDRA
42
43            move.l  #buffer,$dff0a0   ;#Wavedatazeiger Kanal 0
44            move.l  #buffer,$dff0b0   ;#Kanal 1 setzen
45            move.w  #64,$dff0a8       ;#Lautstärke Kanal 0
46            move.w  #64,$dff0b8       ;#Kanal 1 setzen
47            move.w  #speed,$dff0a6    ;#Abspielgeschw. (Period) Kanal 0
48            move.w  #speed,$dff0b6    ;#Kanal 1
49            move.w  #len/2,$dff0a4    ;#Wavelänge Kanal 0
50            move.w  #len/2,$dff0b4    ;#Kanal 1 setzen
```

```
51          move    #$8003,$dff096    ;#Audio-DMA ein
52 loop:    lea     buffer,a0
53          move    #len,d3 digit:
54          ori.b   #%00000100,PRA    ;Prüfen, ob schon komplettes Byte
55          andi.b  #%11111011,PRA    ;übertragen
56 busy:    andi.b  #%00000001,PRA
57          beq.s   busy              ;Wenn nein, dann weiter warten
58          clr     d1
59          move.b  PRB,d1            ;Byte übernehmen
60          eori.b  #128,d1           ;umwandeln
61          move.b  d1,(a0)+          ;und in Puffer speichern
62 ;Der folgende Teil stellt lediglich eine Aussteuerungsanzeige dar,
63 ;kann also auch gelöscht werden!
64          cmp.b   #200,d1           ;Ist der gelesene Wert kleiner
65          bpl     cont              ;als 200
66          move.w  #$0f00,$dff180    ;dann Hintergrundfarbe auf rot setzen
67
68 cont:    dbra    d3,digit          ;Wenn noch nicht der komplette
69                                    ;Puffer voll ist, dann weiter
70 ;An dieser Stelle ist die eigentliche Digitalisierungsroutine am Ende.
71 ;Hier kann nun eine Abspeicherroutine o.ä. eingefügt werden.
72          btst    #6,$bfe001        ;Ist die linke Maustaste gedrückt,
73          bne     loop              ;Wenn nein, dann weiter
74
75          move    #$0003,$dff096    ;#Audio-DMA abschalten
76          move    #0,sr             ;User-Modus einschalten
77
78          rts                       ;Rückkehr
```

9.2: Die serielle Schnittstelle

Die serielle Schnittstelle des Amiga entspricht den üblichen Standard-Seriellen-Schnittstellen, auch als RS232 bezeichnet. Auch hier wurde die eigenwillige Belegung der RS232 beim A1000 bei den Amiga 500 und 2000 an den PC-Standard angepaßt. Zudem besitzt die serielle Schnittstelle des Amiga noch weitere Signale, die mit einer seriellen Übertragung nichts zu tun haben.



Z 9.2-1: *Die Pin-Belegung der RS232-Buchse des A1000, sowie des Steckers des Amiga 500 und 2000.*

Belegt sind die Pins des Steckers bzw. der Buchse wie folgt:

Pin	A1000	A500/2000	Ein-/Ausgang
1	GND	GND	
2	TxD	TxD	Ausgang
3	RxD	RxD	Eingang
4	RTS	RTS	Ausgang
5	CTS	CTS	Eingang
6	DSR	DSR	Eingang
7	GND	GND	
8	DCD	DCD	Eingang
9	---	+12V	
10	---	−12V	
11	---	AUDO	Ausgang
12	---	---	
13	---	---	
14	–5V	---	
15	AUDO	---	Ausgang
16	AUDI	---	Eingang
17	EB	---	Ausgang
18	/INT2	AUDI	Eingang
19	---	---	
20	DTR	DTR	Ausgang

Pin	A1000	A500/2000	Ein/Ausgang
21	+5V	---	
22	---	RI	Eingang
23	+12V	---	
24	/C2	---	Ausgang
25	/RESB	---	Ausgang

Sende-/Empfangsleitungen:

TxD Transmit Data. Dies ist die Sendeleitung der RS232, worüber die Daten gesendet werden.

RxD Receive Data. Dies ist die Empfangsleitung der RS232, worüber die Daten empfangen werden können.

Steuerleitungen/Handshake-Leitungen:

RTS Request to send. Diese Leitung signalisiert dem externen Gerät, daß der AMIGA nun Daten senden will.

CTS Clear to send. CTS signalisiert, daß das externe Gerät Daten senden will.

DSR Data set ready. Mit diesem Signal teilt das angeschlossene Gerät mit, daß es bereit ist für den Datenaustausch.

DCD Carrier detect. DCD wird eigentlich nur bei Modems benutzt. Es stellt die Trägerfrequenz dar.

DTR Data Terminal Ready. Dieses Signal teilt mit, daß der Amiga bereit ist, Daten zu senden.

RI Ring indicator.

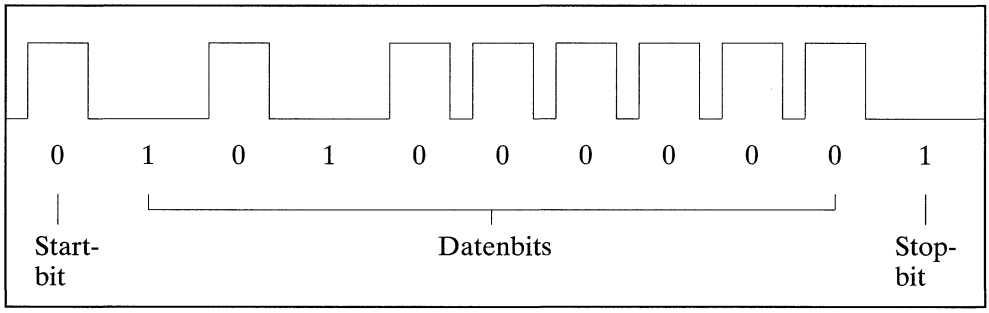
Besondere Signale:

AUDO Audio out.
AUDI Audio in.
/INT2 Interrupt 2.
/RESB Gepuffertes Reset.
/C2 3.58 MHZ Takt.
+12,–12V, +5V,–5V Spannungsversorgungen.

Die Steuerleitungen sind direkt mit der CIA-B verbunden. Ihre Registeradressen können Sie im Anhang Registeradressen der Portbausteine erfahren. Die RI-Leitung ist über einen Transistor mit der SEL-Leitung der Parallel-Schnittstelle verbunden,

kann somit ebenfalls über die CIA abgefragt werden. Die wichtigsten Leitungen sind TxD und RxD. Über sie wird die eigentliche Datenübertragung abgewickelt. Sie sind über zwei Pegelumsetzer direkt mit Paula verbunden. Die Pegelumsetzer MC1488/1489 setzen die Signale +5V/0Vind +12V/–12Vum, da bei RS232 meistens eine lange Übertragungsstrecke vorliegt. Pegel im Bereich von +12V/–12Vsind nicht so störanfällig wie +5V/0V. RxD dient zum Empfang der seriell gesendeten Daten undTxD zum Senden von Daten. Soll nun der AMIGA mit einem Terminal verbunden werden, so muß die TxD-Sendeleitung mit der Empfangsleitung des Terminals und die RxD-Empfangsleitung des Amiga mit der Sendeleitung des Terminals verbunden werden. Bei der seriellen Übertragung wird über diese Leitungen dann ein Bitstrom geschickt. Wie schnell dieser Bitstrom ist, wird in der Baudrate festgelegt, da das Terminal wissen muß, wie schnell der Amiga die Daten sendet. Üblich sind die Baudraten 300, 1200, 2400, 4800 und 9600. Damit der Empfänger weiß, wann eine Übertragung beginnt, werden alle 8 Datenbits ein Startbit vorgesetzt. Das Ende der Übertragung signalisieren ein oder zwei Stopbits. Die Startbits sind immer gleich Null. Die Stopbits immer gleich Eins. An dieser Stelle sei erwähnt, daß eine logische 1 bei der seriellen Übertragung gleich –12V, +12V dagegen eine logische 0 ergeben:

Die Übertragung eines Byte:



Ein Baustein, der eine solche serielle Übertragung ausführen kann, wird als UART, Universal Asynchronous ReceiveTransmit bezeichnet. Ein solcher UARTbefindet sich beim Amiga im Custom-Chip Paula.

9.2.1: Paula, der UART des Amiga

Paula hat als UART des Amiga verschiedene Register zur Verfügung, die das Empfangen und Senden von Daten erheblich vereinfachen. Die empfangenen Datenbits werden vom Amiga im Takt der eingestellten *Baudrate* in ein Schieberegister geladen. Anschließend werden sie zu einem parallelen Datenbyte zusammengesetzt (eine Übertragung von 9 Bit ist auch möglich, dazu muß im SETPER das LONG-Bit gesetzt werden). Ist das Schieberegister überfüllt, werden die Daten in dem Eingangs-Datenpuffer abgelegt. Ausgelesen kann nur der Eingangs-Datenpuffer, nicht das Schieberegister. Das Datenbyte kann dem SERDATR-Register entnommen werden, wobei die ersten 8 Bit die Datenbits darstellen, die letzten zwei die Stopbits. Das Startsignal zum Auslesen des Registers gibt das RBF (Receive-Buffer-Full) in SERDATR und im INTREQ-Register an. Beide müssen nach dem Auslesen der Daten von SERDATR gelöscht werden. Wird dieses Auslesen und Rücksetzen unterlassen, so werden solange Daten empfangen, bis das Schieberegister voll ist. Ist dies der Fall, so wird das OVRUN-Bit gesetzt. OVRUN wird gelöscht, wenn RBF gelöscht wird. Der Sendevorgang des UART läuft in der Art wie der Lesevorgang ab. Die Daten werden hier in das SETDAT-Register geschrieben. Diese Daten werden an das Ausgangsschieberegister übertragen. Diese Übertragung wird durch das TBE-Bit (Transmit-Buffer-Empty) signalisiert. Das TBE-Bit muß wie das RBF-Bit gelöscht werden. Denken Sie hierbei daran, daß es auch im INTEREQ-Register vorhanden ist. Wenn die Daten nicht schnell genug nachgereicht werden, wird das TSRE-Bit gesetzt (Transmit-Shift-Register-Empty). Dieses Bit wird auch automatisch beim Zurücksetzen von TBE zurückgesetzt. Beim Schreiben der Daten in SERDAT dürfen Sie nicht das Setzen der Stopbits vergessen.

9.2.1.1: Das serielle Datenregister SERDATR und SERDAT

SERDATR wird benötigt, wenn Daten gelesen, und SERDAT, wenn Daten gesendet werden sollen.

Bit	Name	Funktion
15	OVRUN	Mit diesem Bit wird der Überlauf des Empfangschieberegisters signalisiert.
14	RBF	Dieses Bit ist gleich 1, wenn der Empfangspuffer voll ist.
13	TBE	TBE signalisiert, daß der Sendepuffer leer ist.
12	TSRE	TSRE signalisiert, daß das Sendeschieberegister leer ist.
11	RxD	Signalisiert den logischen Zustand der RxD-Leitung.
10	---	Nicht benutzt
9	STP	Stopbit

Bit	Name	Funktion
8	STP/DB8	Stop- oder Datenbit 8. Dies wird mit dem SERPER-Register festgelegt.
7	DB7	Datenpuffer Datenbit 7
6	DB6	Datenputter Datenbit 6
5	DB5	Datenpuffer Datenbit 5
4	DB4	Datenpuffer Datenbit 4
3	DB3	Datenpuffer Datenbit 3
2	DB2	Datenpuffer Datenbit 2
1	DB1	Datenpuffer Datenbit 1
0	DB0	Datenpuffer Datenbit 0

9.2.1.2: Das serielle Perioden-Register SERPER

Mit SERPER wird die Länge der Empfangs-/Sendedaten eingestellt (ob 8 oder 9 Bit). Zudem enthält dieses Register die Baudrate mit der gesendet/empfangen wird. Für die Baudrate stehen 15 Bits zur Verfügung. Sie kann nicht direkt eingetragen werden. Als Grundmaß für die Baudrate werden beim Amiga die Buszyklen verwendet, die zwischen zwei Bits liegen. Ein Buszyklus entspricht 2.79365 * 10^-7 Sekunden. Folgende Berechnung wird empfohlen:

$$\text{SERPER} = 1 \quad \frac{1}{\text{Baudrate} * 3 * 10\text{^}-7} \quad - 1$$

Das Ergebnis wird einfach auf- oder abgerundet. Somit kann der Benutzer jede beliebige Baudrate wählen.

Bit	Name	Funktion
15	LONG	Mit diesem Bit kann die Länge der Empfangs-/Sendedaten auf 9, LONG = 1, oder auf 8, LONG = 0, gesetzt werden.
14–0	RATE	RATE enthält die berechnete Baudrate.

9.2.1.3: Die UART-Unterbrechung

Soll eine Datenübertragung abgebrochen werden, so kann dies mit dem Bit UARTBRK im ADKCON-Register vollzogen werden.

Bit	Name	Funktion
11	UARTBRK	Wird dieses Bit gesetzt, so stoppt die serielle Ausgabe. TXD wird auf 0 gesetzt.

Kapitel 10

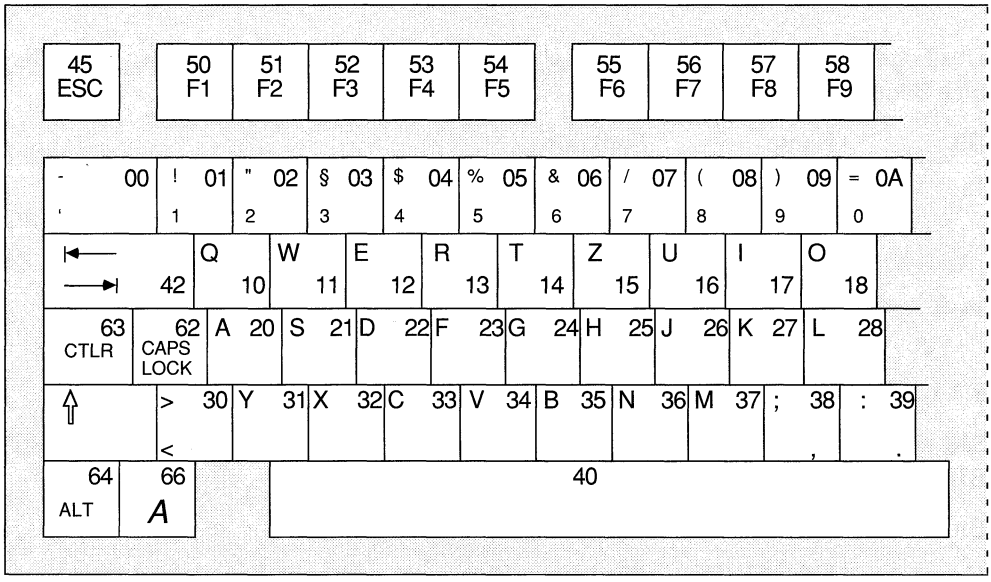
Die Tastatur

Bei der Amiga-Serie können wir von zwei verschiedenen Tastaturen ausgehen. Während der Amiga 1000 mit einer 89-Tasten-Tastatur geliefert wurde, sind bei dem Amiga 500 und Amiga 2000 wegen der Kompatibilität zur PC-Tastatur 5 Tasten hinzugefügt worden. Sie enthält nun 94 Tasten.

Zudem besitzt die Tastatur des Amiga einen eigenen Prozessor, der die Tastenkombinationen auswertet. Sie kann sozusagen als »intelligent« bezeichnet werden.

Die Daten, die vom Tastaturprozessor gesendet und von einem der CIA-8520-Portbausteine empfangen werden, liegen nicht in ASCII-Form vor, was einigen Lesern nun bestimmt sehr ungewöhnlich vorkommt. Stattdessen erhält man für jede Taste einen *Raw-Key-Code,* was eine größere Flexibilität erlaubt, da man nicht an eine länderspezifische Tastaturbelegung gebunden ist.

Die Abbildung Z 10-1 zeigt die AMIGA-Tastatur mit dem jeweiligen Raw-Key-Wert.



Z 10-1: *Jede Taste hat ihren eigenen Raw-Key-Wert (Teil 1).*

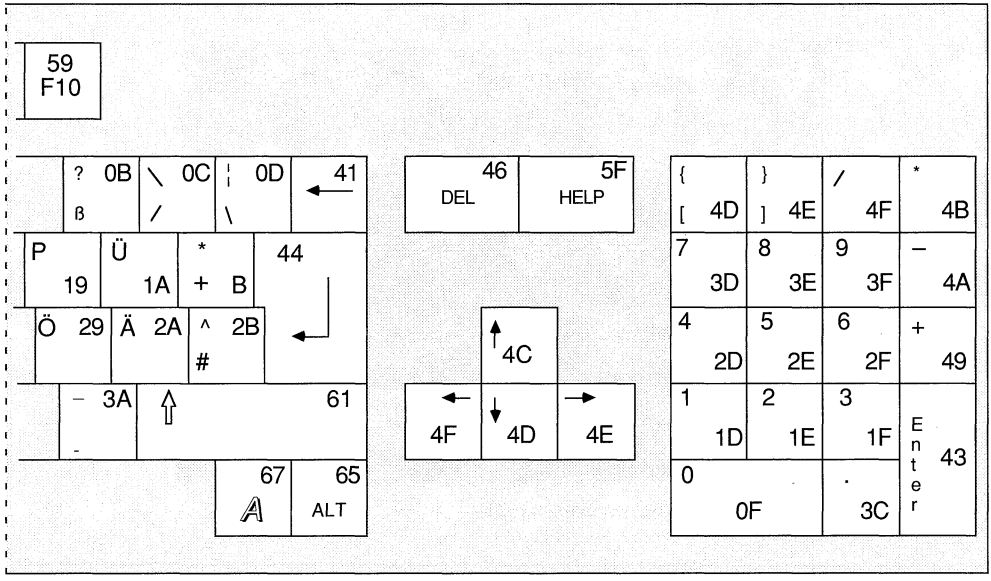
10.1: Der Tastaturprozessor

Die Tastatur des Amiga kann als »intelligent« bezeichnet werden, denn zur Verwaltung der verschiedenen Tastenkombinationen enthält sie einen eigenen Prozessor. Verwendung findet ein 6500, bei der Amiga-1000-Serie, oder ein 6502-Tastatur-Prozessor beim Amiga 500. Der AMIGA 2000 fällt hierbei ganz aus der Rolle, denn bei seiner Tastatur wird ein 8049 eingesetzt, der sonst nur bei IBM bzw. IBM-kompatiblen Tastaturen Verwendung findet.

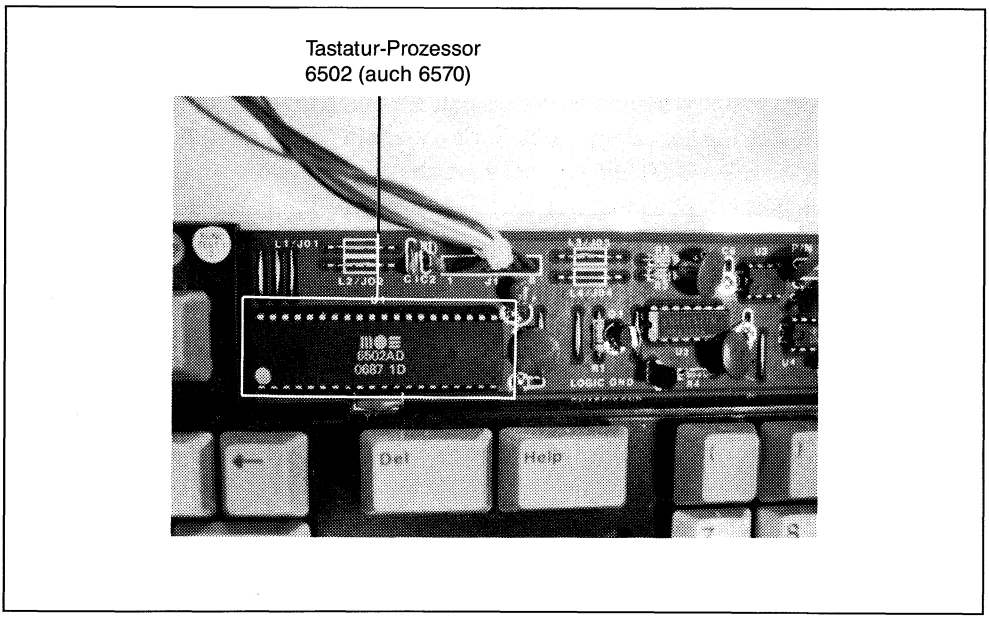
Diese Bausteine werten die entsprechenden Tastenkombinationen aus und geben die jeweiligen Signale, je nach Tasten-Code, zum Amiga weiter.

Hier werden wir nur auf die 6500- bwz. 6502-Tastaturprozessoren näher eingehen, da der 8049 Prozessor in den Funktionen weitgehend identisch ist.

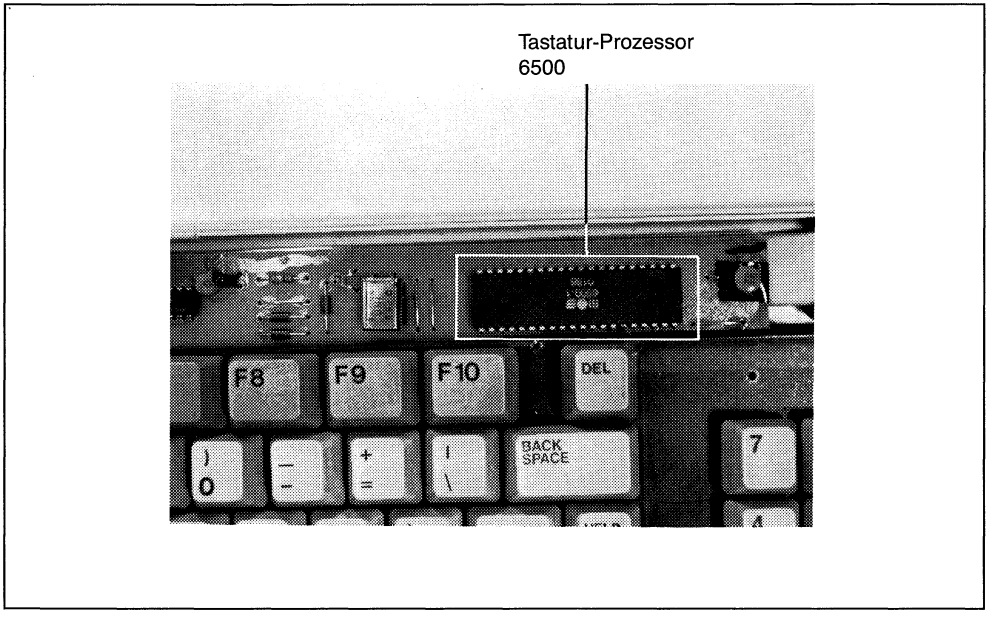
Beide Chips sind 8-Bit- »single chip« Mikroprozessoren. Sie enthalten eine 6502 CPU, also einen kleinen »64er«, sowie einen internen Clock-Oszillator, 2 Kbyte ROM, 64 Byte RAM und eine flexible Interface-Schaltung. Diese Interface-Schaltung enthält einen 16 Bit programmierbaren Zähler/Latch für 4 Operations-Modi, 32 bidirektionale Input-/Output-Eingänge, an denen überwiegend die Tasten der Tastatur angeschlossen sind, sowie fünf Interrupts und einen Zähler-I/O-Eingang. Die Abbildung Z 10.1-1 veranschaulicht die Pinbelegung des 6500/1-Prozessors:



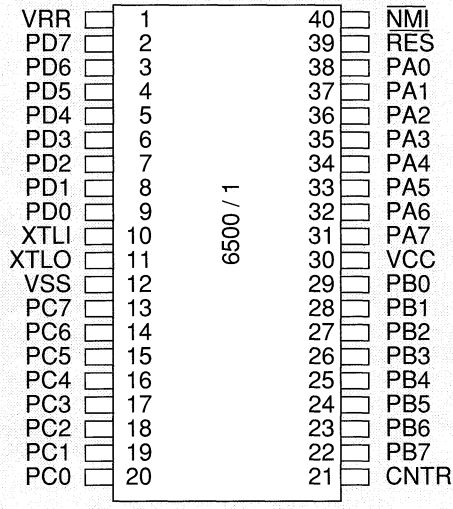
Z 10-1: *Jede Taste hat ihren eigenen Raw-Key-Wert (Teil 2).*



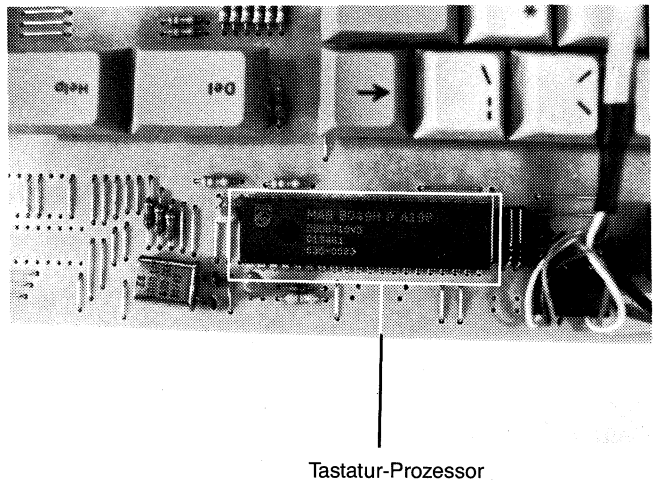
F 10.1-1: *Der Tastaturprozesor des Amiga 1000*



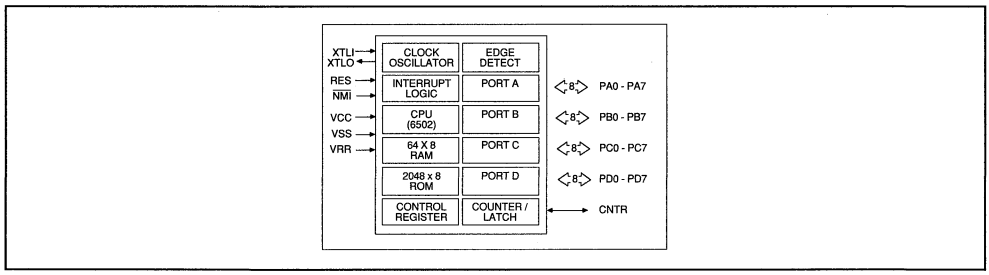
F 10.1-2: *Der Tastaturprozessor des Amiga 500*



Z 10.1.-1: *Die Pinbelegung des 6500/1*



F 10.1-3: *Der Tastaturprozessor des Amiga 2000*



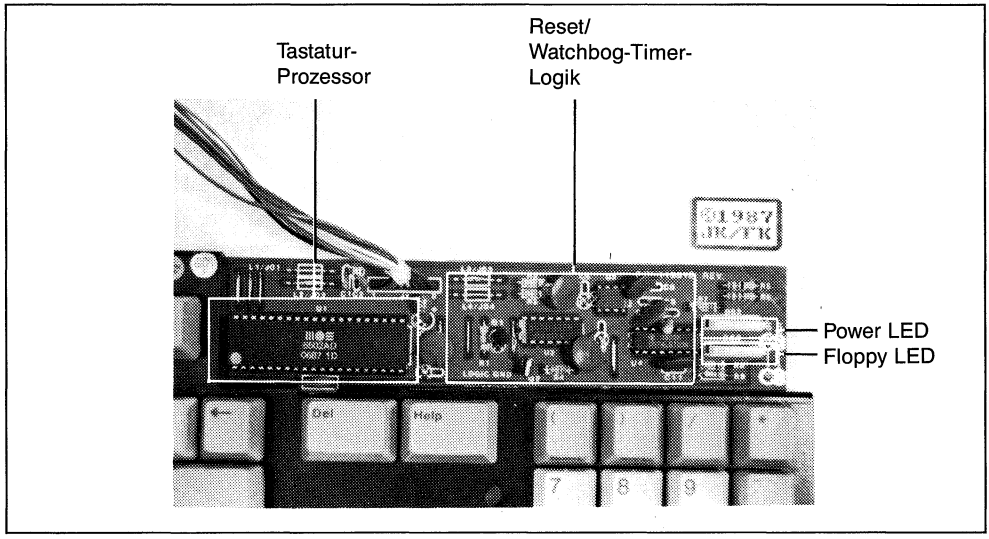
Z 10.1-2: *Das Blockdiagramm des Tastaturprozessors 6500/1*

Pinbeschreibung MOS 6500/1:

Signal	PinNr.	Erläuterung
VCC	30	+5V
VRR	1	Externe Spannungsversorgung für das interne RAM.
VSS	12	GND
XTLI	10	Quarz oder RC-Netzwerk. Eingang für internen Oszillator.
XTLO	11	Quarz oder RC-Netzwerk-Ausgang für internen Oszillator.
/RES	39	Dieser Eingang initialisiert den Prozessor (Reset).
/NMI	40	Non-Maskable-Interrupt, ist auf log. high gelegt und wird bei der Amiga-Tastatur nicht verwendet.
PA0–PA7	38–31	Vier 8 Bit-Ports für Ein- und Ausgabe.
PBO–PB7	29-22	Hier sind die Tasten der Amiga-Tastatur angeschlossen,
PCO–PC7	20–13	wobei PA1 als Clock-Ein-/Ausgang und PA0 als
PDO–PD7	9–2	Keyboard-Daten Ein-/Ausgang dient. Auf diese Ports wird in den nächsten Kapiteln noch ausführlicher eingegangen.
CNTR	21	Dieser Pin wird als Zähler Ein-und Ausgang verwendet. Er ist bei der AMIGA-Tastatur auf GND gelegt.

10.2: Der Watch-Dog-Timer

Neben dem oben beschriebenen Tastatur-Prozessor sind noch weitere wichtige Schaltungen in der Tastatur enthalten. Eine dieser Schaltungen ist ein Watch-Dog-Timer. Dies ist ein sogenannter »Wachhund««, der vom Ein-/Ausgabe-Port PD7 getriggert (ausgelöst) wird. Bei den ersten Tastaturen der Amiga-Serie war dieser »Wachhund« leider nicht enthalten. Grob beschrieben bewahrt er die Tastatur vor dem »Aufhängen«, er ist sozusagen ein Reset-Schalter. Der Tastatur-Prozessor überprüft ständig die Key-Matrix, dabei bekommt auch der Watch-Dog-Timer sein Signal. Fehlt dieses Signal für 54 ms, so sendet diese Schaltung einen Reset-Impuls von 450 us an den Tastaturprozessor.



F 10.2-1: *Die Reset/WatchDog-Timer Logik in der A500-Tastatur*

Die Tastatur-Matrix:

Die Tastatur-Matrix des Amiga besteht aus 6 Reihen und 15 Spalten. Hinzu kommen noch 7 Sondertasten. Somit sind insgesamt 97 verschiedene Tastenkombinationen möglich. Jede Reihe ist als ein Eingang zu betrachten und auf +5V gehalten. Die Spalten hingegen sind Ausgänge. Diese Sondertasten, Spalten und Reihen liegen an den Ein- und Ausgabeports des Tastaturprozessors an. Er übernimmt die Auswertung, an welcher Spalte und Reihe eine Taste betätigt wurde und gibt diese Werte als serielle Daten an den Amiga weiter. Insgesamt besitzt der Tastaturprozessor, wie schon erwähnt, 4 Ports mit je 8 Ein- oder Ausgängen. Hier die Belegung der Ein- und Ausgabeports:

Port A			Port B		
PA0	Ein-/Ausgang	KDAT – Keyboard-Daten	PB0	Eingang	rechte Shift-Taste
PA1	Ausgang	KCLK – Keyboard-Clock	PB1	Eingang	rechte Alt-Taste
PA2	Eingang	Reihe 0	PB2	Eingang	rechte Amiga-Taste
PA3	Eingang	Reihe 1	PB3	Eingang	CTRL-Taste
PA4	Eingang	Reihe 2	PB4	Eingang	linke Shift-Taste
PA5	Eingang	Reihe 3	PB5	Eingang	linke Alt-Taste
PA6	Eingang	Reihe 4	PB6	Eingang	linke Amiga-Taste
PA7	Eingang	Reihe 5	PB7	Ausgang	Caps-Lock LED

Alle Tasteneingänge sind low, wenn die jeweilige Taste betätigt ist. Wenn der Caps-Lock LED-Ausgang log. high ist, leuchtet die LED.

Port C		Port D	
PC0 Ausgang	Spalte 0	PD0 Ausgang	Spalte 8
PC1 Ausgang	Spalte 1	PD1 Ausgang	Spalte 9
PC2 Ausgang	Spalte 2	PD2 Ausgang	Spalte 10
PC3 Ausgang	Spalte 3	PD3 Ausgang	Spalte 11
PC4 Ausgang	Spalte 4	PD4 Ausgang	Spalte 12
PC5 Ausgang	Spalte 5	PD5 Ausgang	Spalte 13
PC6 Ausgang	Spalte 6	PD6 Ausgang	Spalte 14
PC7 Ausgang	Spalte 7	PD7 Ausgang	Spalte 15

Aktiv ist der jeweilige Ausgang, wenn er log. low ist.

Die Tasten werden nur im Port A und Port B (Reihen) des Tastaturprozessors gelesen. Eine Auswahl findet sozusagen durch Ausgänge (Spalten) statt. Somit hat jede Taste ihre Reihe und Spalte:

Über Port A des Tastaturprozessors können folgende Kombinationen gelesen werden, wobei das oben stehende Zeichen gültig ist, wenn die Shift-Taste (das Shift-Bit in Port B gesetzt ist):

Spalte	Bit 7 Reihe 5	Bit 6 Reihe 4	Bit 5 Reihe 3	Bit 4 Reihe 2	Bit 3 Reihe 1	Bit 2 Reihe 0
15/PD.7						
14/PD.6			Caps Lock	TAB TAB	ß `	ESC ESC
13/PD.5		Z z	A a	Q q	! 1	
12/PD.4		X x	S s	W w	a 2	F1 F1
11/PD.3		C c	D d	E e	# 3	F2 F2
10/PD.2		V v	F f	R r	$ 4	F3 F3
9/PD.1		B b	G g	T t	% 5	F4 F4
8/PD.0		N n	H h	Y y	^ 6	F5 F5
7/PC.7		M m	J j	U u	& 7	

Spalte	Bit 7 Reihe 5	Bit 6 Reihe 4	Bit 5 Reihe 3	Bit 4 Reihe 2	Bit 3 Reihe 1	Bit 2 Reihe 0
1/PD.7						
6/PC.6		< ,	K k	I i	* 8	F6 F6
5/PC.5		>> .	L l	O 0	(9	
4/PC.4		? /	: ;	P p	) 0	F7 F7
3/PC.3			" ;	ä Ä	– -	F8 F8
2/PC.2		 Space		ü Ü	+ =	F9 F9
1/PC.1		Back Space	Delete	Return	~ Ö	F10 F10
0/PC.0		Cursor runter	Cursor rechts	Cursor links	Cursor hoch	 Help

Diese Belegung der Tastatur bezieht sich auf die erste 1000er-Serie. Beim Amiga 500 kann diese Belegung leicht abweichen, da hier die Tastatur erweitert wurde. Die wichtigsten Tasten des Amiga sind an einem zusätzlichen Port, an den Port B des Tastaturprozessors, angeschlossen:

Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Amiga links	ALT links	Shift links	CTRL	Amiga rechts	ALT rechts	Shift rechts

10.3: Die Initialisierung der Tastatur

Nach jedem Einschalten bzw. Neustarten des Amiga, oder kurzzeitigem Abklemmen der Tastatur, während der Amiga eingeschaltet ist, beginnt der Tastaturprozessor mit einem Selbsttest. Dies ist an dem Aufleuchten der Caps-Lock LED erkennbar. Bei diesem Selbsttest wird das interne ROM getestet, das 64 Byte große RAM überprüft und zuguterletzt die Funktion des Watch-Dog-Timers kontrolliert. Nach diesem Test findet eine Synchronisierung zwischen Tastatur und Amiga statt. Die Tastatur geht hierzu in den »Resync Modus«. Dabei wird die KCLK-Leitung so lange auf log. 1 gehalten, bis ein Handshake-Signal, d.h. ein Quittungssignal vom Amiga eintrifft. Ist die Synchronisierung beendet, teilt die Tastatur durch spezielle Codes dem Amiga mit, wie der

Selbsttest ausgefallen ist. Mißlingt der Selbsttest, so wird der Spezialcode Hex $FC gesendet. Dieses Fehlschlagen können Sie aber auch an dem Blinken der Caps-Lock LED erkennen. Blinkt die Caps-Lock LED einmal nach dem Aufleuchten beim Anklinken der Tastatur an den eingeschalteten Rechner oder beim Neustart des Amiga, so bedeutet dies, daß ein Fehler im ROM des Tastaturprozessors vorhanden ist, zweimal Blinken entspricht einem RAM-Testfehler, dreimal einem Watch-Dog-Timer-Fehler und viermal einen Fehler in der Key-Matrix. Gelingt hingegen der Selbsttest, so wird der Code Hex $FD und anschließend die Tastencodes der momentan betätigten Tasten gesendet. Sind alle Tastencodes gesendet, so wird dies mit dem Schlußcode Hex $FE signalisiert. Damit ist der Selbsttest der Tastatur beendet und die Caps-Lock LED wird ausgeschaltet.

10.4: Die Kommunikation zwischen Tastatur und Rechner

Die Kommunikation zwischen Tastatur und Rechner findet hauptsächlich durch zwei Leitungen statt. Die eine, KDAT, enthält die Daten, die seriell gesendet werden. Sie dient zur Ein- und Ausgabe. Die zweite, KCLK, enthält den Takt, der zum Senden und Empfangen der Daten notwendig ist. Beide Leitungen sind mit einem Widerstand auf +5 Volt gehalten und können so nur auf GND gelegt werden.

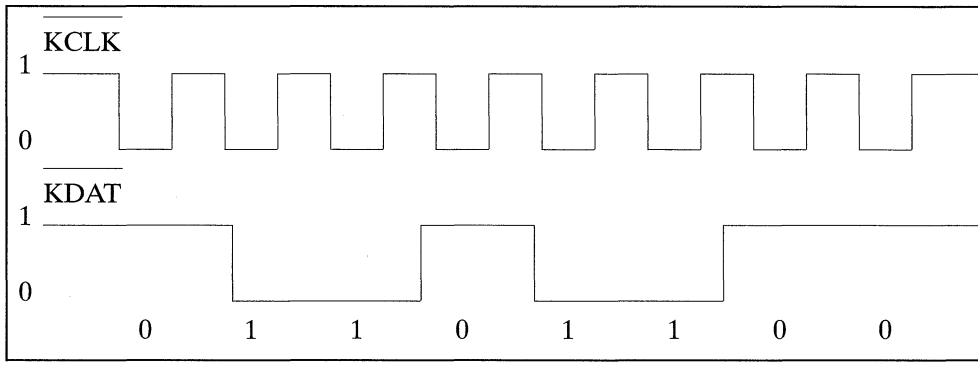
Bevor mit dem Senden der Daten begonnen wird, werden KDAT und KCLK high gesetzt. Dann beginnt die Übertragung von 8 Datenbits von der Tastatur zum Amiga. Jedes Datenbit wird dabei von einem Clockimpuls begleitet. Die 8 Datenbits werden links rotiert gesendet. Daraus ergibt sich eine Reihenfolge von 6-5-4-2-1-0-7. Die ersten 7 Bits erhalten die Keyinformation. Das Bit 7 ist das up/down-Flag. Ist dieses Flag gleich 0, bedeutet dies, daß die jeweilige Taste gedrückt ist. Eine 1 hingegen, daß die jeweilige Taste losgelassen wurde. Sind die 8 Bits beim Amiga angekommen, muß dieser der Tastatur ein *Handshake* (Quittierungssignal) senden. Hierzu wird die KDAT-Leitung für 75 Mikrosekunden auf low gehalten.

Die Übertragungsrate zwischen Tastatur und Amiga beträgt 17Kbits pro Sekunde. Das entspricht 60 Mikrosekunden pro Bit. Hier ein Beispiel für die Übertragung eines Buchstabens:

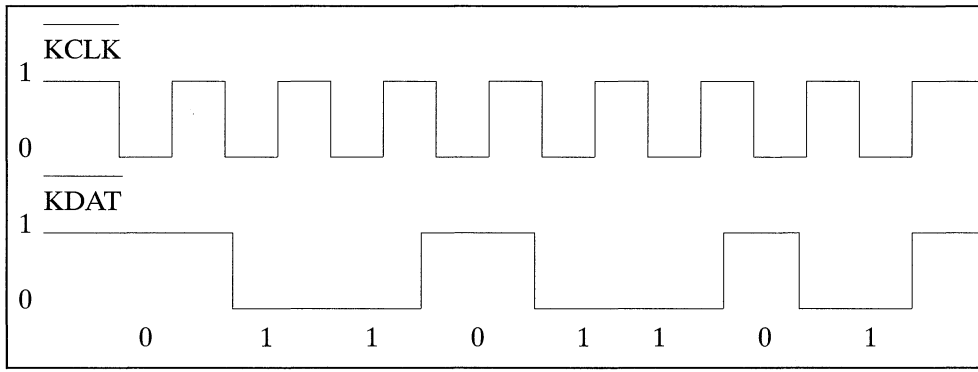
Jeder Buchstabe hat einen Raw-Key-Code, der 7 Bit lang ist. So entspricht der Buchstabe V dem Raw-Key-Code $34 Hex. Das achte Bit (up/down-Flag) bestimmt, ob die Taste noch gedrückt, Flag = 0, oder bereits losgelassen wurde, Flag = 1. Beachten müssen wir dabei, daß KDAT logisch Eins ist, wenn sein Pegelwert gleich Null ist. Das bedeutet, daß hier die negative Logik angewendet werden muß.

Der Raw-Key-Code $34 Hex enspricht dem Bit-Muster 00110110, wenn die Taste gedrückt wurde. Wird sie losgelassen, wird das achte Bit gesetzt. Der Wert lautet dann 10110110. Gesendet werden die Daten links rotiert, somit muß unserer Bit-Muster nach links rotiert werden. Dabei ergibt sich die Kombination 01101100, bzw. 01101101.

Wenn Taste V gedrückt ist, dann ergeben sich also folgende Signalverläufe:



Ist diese Taste losgelassen, so ergibt sich folgender Signalverlauf:




```
 1  /*******************************
 2  1.Tastatur-Demonstration
 3  last update 16/02/88
 4  von Frank Kremser und Jörg Koch
 5  © Markt & Technik 1988
 6
 7  *******************************
 8
 9  Diese Demonstration fragt das Hardwareregister, das den Key-Code der
10  zuletzt gedrückten Taste enthält, ab, wandelt den Code in den
11  Raw-Key-Code um und gibt diesen aus.
12
13  *******************************/
14  #include <exec/types.h>              /* Include-Files laden */
15  #include <exec/tasks.h>
16  #include <exec/memory.h>
17  #include <exec/interrupts.h>
18  #include <exec/execbase.h>
19  #include <exec/io.h>
```

```
#include <exec/libraries.h>
#include <exec/devices.h><
#include <exec/ports.h>
#include <exec/lists.h>
#include <exec/nodes.h><
#include <graphics/gfxmacros.h>
#include <graphics/copper.h>
#include <graphics/view.h>
#include #hardware/custom.h>

main()
 {
  SHORT i,taste;
  UBYTE *key;

  key = 0xbfec01;          /* Addresse des Hardwareregisters */
  Forbid();                /* Intuition und Interrupts 'abschalten' */
  Disable();
  for(i=0;i<1000;i++)
   {
    taste = *key;          /* Key-Code lesen */
    taste = 0xff-taste;    /* invertieren */
    taste>>=1;             /* und um ein Bit nach rechts shiften */
    printf("%x",taste);    /* anschließend ausgeben */
   }
  Permit();                /* Intuition und Interrupts 'einschalten' */
  Enable();
 }
```

```
/*******************************

2.Tastatur-Demonstration
last update 16/02/88
von Frank Kremser und Jörg Koch
© Markt & Technik 1988

*******************************

Diese Demonstration liefert das gleiche Ergebnis, wie die 1. Demo.
Der Unterschied besteht lediglich darin, daß eine MC-Routine zur
Tastenabfrage verwendet wird.

Um dieses Programm zu kompilieren, geben Sie
              'execute tastatur2.comp tastatur2'
ein. Dazu muß aber auch die Datei taste.o auf der Disk vorhanden sein.

*******************************/
#include <exec/types.h>            /* Include-Files laden */
#include <exec/tasks.h>
```

```
#include <exec/memory.h>
#include <exec/interrupts.h>
#include <exec/execbase.h>
#include <exec/io.h>
#include <exec/libraries.h>
#include <exec/devices.h>
#include <exec/ports.h>
#include <exec/lists.h>
#include <exec/nodes.h>
#include <graphics/gfxmacros.h>
#include <graphics/copper.h>
#include <graphics/view.h>
#include <hardware/custom.h>

extern SHORT taste(); /* Maschinenroutine kenntlich machen */


main()
 {
  SHORT i,key;

  Forbid();     /* Intuition und Interrupts 'abschalten' */
  Disable();
  for(i=0;i<1000;i++)
   {
    key = taste();      /* Taste abfragen und Raw-Key-Code übergeben */
    key = 0xff-key;     /* Code invertieren */
    key>>=1;    /* und um 1 Bit nach rechts shiften */
    printf("%x",key);/* endgültigen Raw-Key-Code ausgeben */
   }
  Permit();   /* Intuition und Interrupts wieder 'einschalten' */
  Enable();
 }
```

```
;*******************************
;
; MC-Routine zur
; 2.Tastatur-Demonstration
; last update 16/02/88
;von Frank Kremser und Jörg Koch
; © Markt & Technik 1988
;
;*******************************
;
;Diese kurze Routine fragt ein Hardware-Register ab, daß den
;Raw-Key-Code der zuletzt gedrückten Taste enthält.
;
;*******************************

```

```
16              CSECT   text                ;Kennungen für C
17              XDEF    _taste              ;
18
19  _taste      moveq   #0,d0               ;Datenregister löschen
20              move.b  $bfec01,d0          ;Hardwareregister lesen und
21              rts     ;in D0 an C übergeben
22              END
```

10.5: Die Tastaturverbindung zum Amiga

Inzwischen sind drei verschiedene Steckverbindungen zur Amiga-Tastatur bekannt. Die erste, die des Amiga 1000, enthält nur 4 Leitungen. Beim AMIGA 500 wurden noch zusätzliche Funktionen auf der Tastatur integriert, so daß hier auch die meisten Leitungen zur Tastatur gehen. Sie ist jedoch starr eingebaut. Die Steckverbindung zur Amiga 2000-Tastatur wurde der Commodore PC10-Tastatur angepaßt.

Die wichtigsten Leitungen, neben der Spannungsversorgung, sind zwei Datenleitungen. Wie schon im letzten Kapitel erwähnt, dient eine zur Übertragung des Clockimpuls zur Tastatur, »KCLK« genannt, die andere, mit »KDAT« bezeichnet, zur Übertragung der Keyboard-Daten. Die letztere Leitung dient als Ein- und Ausgabe, während die erstgenannte nur als Signal von der Tastatur benutzt wird.

Hier nun die Pinbelegung der Stecker zur Tastatur auf dem jeweiligen Motherboard (Hauptplatine):

Amiga 500

Pin	Signal
1	KCLK – Clock
2	KDAT – Key Daten
3	+5V
4	N.C.
5	GND
6	/RESET
7	Power LED
8	Drive LED

Amiga 1000

Pin	Signal
1	+5V
2	KCLK – Clock
3	KDAT – Key Daten
4	GND

Amiga 2000

Pin	Signal
1	KCLK – Clock
2	KDAT – Key Daten
3	N.C.
4	GND
5	+5V
6	GND
7	GND

10.6: Der Tastencode-Empfänger im Amiga

Als Tastencode-Empfänger wird einer der CIA-8520-Portbausteine des Amiga verwendet. Über ihn werden alle Signale an die Tastatur weitergegeben bzw. von ihr empfangen. Wichtig für das Handling mit der Tastatur ist nur der *CIA-A*-Baustein. Er enthält verschiedene Register, aus denen die empfangenen Daten gelesen oder Signale an die Tastatur gesendet werden können. Hier die Adressen der wichtigsten Register:

`$BFEC01` Dies ist das serielle Datenregister zur Kommunikation mit der Tastatur. Hier können die invertierten, linksrotierten Raw-Key-Codes empfangen werden.

`$BFEE01` Kontrollregister A. Mit diesem Register kann das serielle Datenregister kontrolliert werden.

Die Abfrage der *Portbausteine* übernimmt softwaremäßig die im Kickstart-ROM enthaltene Keyboard-Device, beginnend bei $FE53E8. Sie wertet die Tasten aus und regelt das *Handshake*. Von ihr kann man auch erfahren, wie man direkt auf die CIA zugreift, um eine entsprechende Taste abzufragen.

Folgende Möglichkeiten zur direkten Manipulation der Tastatur bieten sich an:

1. Sendeleitung der Tastatur sperren, d.h. KDAT-Leitung auf logisch low legen, Sendeleitung der Tastatur freigeben und die empfangenen Daten dekodieren.
   Für das Sperren der Sendeleitung, sowie freigeben derselben wird das Kontrollregister A des CIA-A-Portbausteins verwendet. Das Keyboard kann mit dem Befehl

   ```
   ori.b #$40,$BFEE01
   ```

   gesperrt, oder mit

   ```
   andi.b #$BF,$BFEE01
   ```

   freigegeben werden.

2. Das Erkennen der betätigten Taste aus dem seriellen Datenregister der CIA-A ist schon etwas schwieriger, da die Daten invertiert und linksrotiert empfangen werden:

   ```
   move.b $BFEC01,d0 ;Keycode in d0 lesen
   ror.b #,d0 ;1 Bit nach rechts rotieren
   not.b d0 ;d0 invertieren
   ```

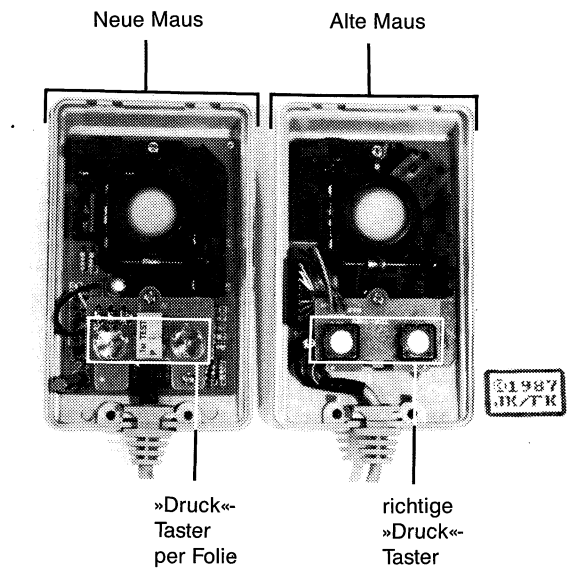
Will man dies in C probieren, so muß noch etwas mehr Aufwand betrieben werden. Hier bieten sich zwei Lösungen an. Einmal ein reines C-Programm, oder eine Kombination zwischen C und Assembler, wie auf den Seiten 289 bis 292 dargestellt.

Kapitel 11

Die Maus

Die Maus ist eine der wichtigsten Eingabeeinheiten des Amiga, oder können Sie sich die Cursorbewegung des Malprogramms DeluxePaint mit der Tastatur vorstellen? Sicherlich nicht, denn diese Methoden waren in der »Steinzeit« der Computertechnologie üblich und sind schon etwas länger her.

Die Maus des Amiga besitzt zwei Tasten. Sie wird auch als *optische Maus* bezeichnet, da hier hier über zwei Lichtschranken die Bewegungen in X- und Y-Richtung abgefragt und durch ein Impuls-Diagramm an den Amiga weitergegeben werden. Auch an der Maus des Amiga hat sich seit dem ersten Amiga viel getan. Die Top-Maus der ersten Stunden, ist inzwischen zu einer Low-cost-Maus mit Folientasten geworden, die so manchen Bediener fast zur Verzweiflung bringen.

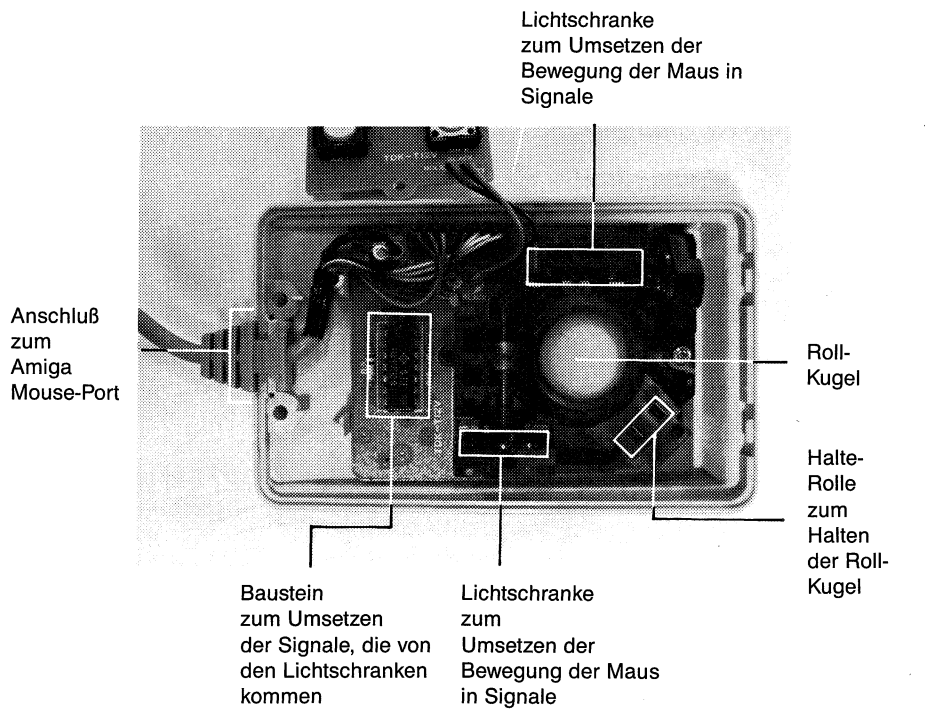


F 11-1: *Die alte und die neue Maus*

```
/*****************************

1. Joyport-Demonstration
last update 16/02/88
von Frank Kremser und Jörg Koch
© Markt & Technik 1988

*****************************

Diese Joyport-Demonstration setzt den 2. Port als Mausport.
Bitte stecken Sie nach dem Start des Programms die Maus in diesen Port.
Keine Sorge, das können Sie auch bei eingeschaltetem Rechner.

*****************************/
#include <exec/types.h>              /* Include-Files laden */
#include <exec/nodes.h>
#include <exec/lists.h>
#include <exec/ports.h>
#include <exec/io.h>
#include <devices/input.h>

struct IOStdReq ior; /* Device-Request-Block */
struct MsgPort mes; /* Messageport */

main() /* HAUPTPROGRAMM */
 {
  UBYTE port = 1; /* Port 1 als Mausport - normal ist Port 0 */

  if (OpenDevice("input.device", 0, &ior, 0) != 0) /* Device öffnen */
    exit(20);

  mes.mp_Node.ln_Type = NT_MSGPORT;        /* Erstelle einen */
  mes.mp_Flags = 0;                        /* Message-Port */
  if ((mes.mp_SigBit = AllocSignal(-1)) <0)
   {
    CloseDevice(&ior);
    exit(20);
   }
  mes.mp_SigTask = (struct Task *) FindTask((char *) NULL);
  NewList(&mes.mp_MsgList);
  ior.io_Message.mn_ReplyPort = &mes;
  ior.io_Command = IND_SETMPORT;       /* Initialisiere die Request-Structure */
  ior.io_pData = &port;                * für das Umschalten auf einen anderen */
  ior.io_Length = 1;                   /* Mouse-Post */
  DoIO(&ior);

  CloseDevice(&ior);                   /* Schließe Device und lösche
                                          Signalbit */
  FreeSignal(mes.mp_SigBit);
 }
```

11.1: Aufbau und Funktionsweise der Maus

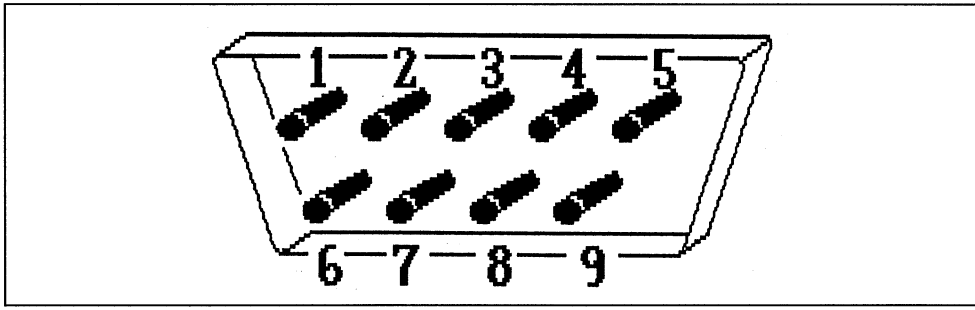
Eines der wichtigsten Elemente der Maus ist die Rollkugel. Sie wird durch ein Plättchenan der unteren Seite der Maus in einer Kuppel gehalten. Liegt die Maus auf, so wird sie durch eine Halterolle an zwei weitere Rollen gepreßt, die jede Bewegung der Kugel mitmachen. Diese zwei Rollen bewegen eine Art Zahnrad, das eine Lichtschranke abwechselnd öffnet und schließt. Diese Signale werden an einen weiteren Baustein gegeben, der die Daten für den Amiga vorbereitet.



F 11.1-1: *Der Aufbau der Amiga-Maus*

11.2: Empfang der Mausdaten

Die Maus wird normalerweise an den *Joyport* 0 angeschlossen. Bei diesem Port verwendet die Maus alle Pins außer Pin 5. Die Pinbelegung an den Joyports zeigt Abbildung Z 11.2-1:



Z 11.2-1: *PIN-Belegung des Joyports*

Hier nun die Signale, die über die einzelnen Pins übertragen werden:

1	V-Impuls	6	linke Maustaste
2	H-Impuls	7	+5V
3	VQ-Impuls	8	GND
4	HQ-Impuls	9	rechte Maustaste
5	nicht benutzt		

Die Signale V-, H-, VQ- und HQ-Impuls werden von einem 2 zu 1 Multiplexer umgesetzt und anschließend Denise zugeführt. Denise setzt die entsprechenden Register nach diesen zwei empfangenen Signalen. Diese Register sind *JOY0DAT* für Port 0 und *JOY1DAT* für Port 1. Vergleiche Bild 16 im Farbteil.

Die Bits 0 bis 7 enthalten den horizontalen Zähler der Maus, während die Bits 8 bis 15 den vertikalen Zähler enthalten. Dabei geben die Bits 7, bzw. 15 jeweils die Richtung an, in die bewegt wurde. Um nun eine genaue Positionsbestimmung durchführen zu können, muß man von dem letzten gelesenen Wert die neuen Werte abziehen. Man erhält dann die Richtung und die Anzahl der Steps, um die die Maus bewegt wurde. Die Maus gibt etwa 79 Steps pro Zentimeter. Hier nun ein paar Beispiele zur Berechnung:

Alter Wert	Neuer	Richtung	Stepdifferenz
100	50	Hoch/Links	50
50	100	Runter/Rechts	–50

Die Maustasten kann man über die Register der CIA-Bausteine abfragen. Ist Bit 6 von $BFE001 gleich 0, so ist die linke Maustaste am Port 0 gedrückt und ist Bit 7 gleich 0, dann die linke Taste von Port 1. Die anderen Tasten lassen sich nur über die *POTGO*-Register abfragen.

Hier ein kurzes Beispiel zur Abfrage der linken Maustaste in Assembler:

```
wait:   btst #,$bfe001 ;Wartet, bis linke Maustaste
        bne  wait      ;gedrückt
```

Kapitel 12

MS-DOS-Erweiterungen

IBM hat durch seine PCs und das MS-DOS einen Standard gesetzt, der heute nicht mehr wegzudenken ist. Obwohl die Technik dieser Computer veraltet ist, sind sie beherrschend in vielen Anwendungsbereichen und es besteht weiterhin große Nachfrage nach diesen Rechnern, da ein Softwareangebot vorhanden ist, das es bisher zu keinem anderen Rechner gibt und auch in absehbarer Zukunft nicht geben wird. So haben es andere, technisch ausgereiftere Rechner, wie beispielsweise der Amiga, sehr schwer gegen einen solchen Koloß anzutreten, da sie zunächst ein recht unterentwickeltes Softwareangebot besitzen. Um dem Amiga große Chancen im Anwendungsbereich geben zu können, mußte ein MS-DOS-Emulator her, der sowohl software- als auch hardwarekompatibel sein sollte. Commodore-Braunschweig entwickelte Anfang 86 einen solchen Emulator, das SideCar. Es kam im Herbst 86 als Zusatz zum Amiga 1000 heraus. Die Fotos 2–4 im Farbteil zeigen das SideCar.

Anfangs hatte man große Schwierigkeiten bei der Entwicklung, so daß das Layout des Interface zum Amiga 1000 sich ständig änderte, was auch heute noch an den vielen aufgetrennten Leiterbahnen und notdürftig gelöteten Brücken erkennbar ist. Auch das BIOS, das anfangs als Testversion ausgeliefert wurde, hinterläßt durchaus keinen vertrauenserweckenden Eindruck. Aber die Bombe eines solchen Beiwagens war eingeschlagen und die Rechnung von Commodore ging auf. Überall sprach man nun von 2 Rechnern in einem. Es ist schon überwältigend, auf der einen Seite im CLI und auf der anderen Seite, in einem kleinen Fenster am unteren Rand mit Wordstar zu arbeiten. Die Fachwelt war begeistert. Auf der CeBit '87 kam dann der ganz große Clou. Mit dem Amiga 2000, der mit der PC-Emulator-Karte und dessen Möglichkeiten das SideCar ablöste, brachte Commodore einen Rechner heraus, der je nach Steckkarte PC/XT oder sogar AT mit 80286 oder 80386 kompatibel sein konnte. Dies war die Revolution auf dem Computermarkt. Ein Computer, den man sich je nach Belieben für seine Zwecke zusammenstellen konnte. Auf der einen Seite der AMIGA mit der kraftvollen MC68000 CPU und Custom-Chips und auf der anderen Seite PC- oder AT-Kompatibilität, was will man mehr.

Commodore hat mit diesem neuen Prinzip einen großen Schritt nach vorne gewagt. Auf jeden Fall will Commodore noch weitere Maßstäbe in diesem Bereich setzen und ihre 2000er Linie weiterführen. Wir sind der Meinung, daß dies eine fantastische Neuerung auf dem Computer-Markt ist. Wenn man sich erst einmal an einen solchen Rechner gewöhnt hat, möchte man seine Fähigkeiten nicht mehr missen.

12.1: Das SideCar

Das *SideCar* stellt eine MS-DOS-Hardware-Emulation für den Amiga 1000 dar. Dort kann es nur angeschlossen werden, wenn dieser auf eine Speicherkapazität von mindestens 512 Kbyte aufgerüstet ist (Farbteil Bild 25).

Beim Anschluß an den Amiga 500 entstehen einige Probleme, da der Expansionsport des Amiga 500 um 180 Grad gedreht wurde. Das Interface, das die Verbindung vom PC- zum Amiga-Teil darstellt, ist das wichtigste Teil des SideCar, da hier die Daten über ein Dual-Ported-RAM zwischen Amiga und SideCar ausgetauscht werden. Weiter enthält dieses Interface einen Adreß- und Datenbus-»Übersetzer«, der die Daten für das *Dual-Ported-RAM* vorbereitet und die Kontrolle übernimmt (Farbteil Bild 26).

Ist das SideCar richtig an den Amiga angeschlossen, so sollte man noch die Dip-Schalter des SideCar überprüfen, bevor man mit dem Starten des Systems beginnt (Farbteil Bild 27).

Hier stehen an der rechten, vorderen Seite, in der Nähe der RAM-Chips, 2 Dip-Schalterleisten zur Verfügung. Die erste, mit 8 Schaltern, bestimmt die Konfiguration des Systems:

Schalter	Funktion
1	OFF = Normales Booten des Systems ON = Systemdiagnose
2	OFF = 8087-Koprozessor nicht installiert ON = 8087 installiert
3 und 4	Größe des Hauptspeichers: 3 4 ON ON = 128 Kbyte RAM OFF ON = 256 Kbyte RAM ON OFF = 512 Kbyte RAM OFF OFF = 640 Kbyte RAM

Schalter	Funktion
5 und 6	Videomodus beim Start: 5 6 ON ON = Kein Video OFF ON = farbig, 40 Zeichen mal 25 Zeilen ON OFF = farbig, 80 Zeichen mal 25 Zeilen OFF OFF = monochrom, 80 Zeichen mal 25 Zeilen
7 und 8	Anzahl der Floppy-Laufwerke 7 8 ON ON = 1 OFF ON = 2 ON OFF = 3 OFF OFF = 4

Der zweite Dip-Schalter bestimmt die Konfiguration des Dual-Ported-RAM (DPR), das wir in den kommenden Kapiteln noch näher betrachten werden:

Schalter	Funktion
1	OFF = Die Adressen B0000–B1FFF sind auf dem Interface belegt, das bedeutet, daß keine externe monochrome Video-Karte verwendet werden darf. ON = Die Adressen B0000–B1FFF sind auf einer externen Video-Karte vorhanden.
2	OFF = Die Adresse B8000–BFFFF für eine Color-Video-Karte sind auf dem Interface belegt, es darf keine externe Video-Karte verwendet werden. ON = Die Adressen B8000–BFFFF sind auf einer externen Color-Video-Karte vorhanden.
3 und 4	Dient zur Verschiebung des Dual-Ported RAM 3 4 Lage des DPR XX ON = A0000–AFFFF ON OFF = D0000–DFFFF OFF OFF = E0000–EFFFF

Wird das SideCar speichermäßig aufgerüstet, so muß neben den Schalterstellungen 3 und 4 des ersten Dip-Schalters auch ein Jumper verändert werden. Dieser befindet sich über dem BIOS-ROM neben den Sockeln für die RAM-Chips. Für eine solche

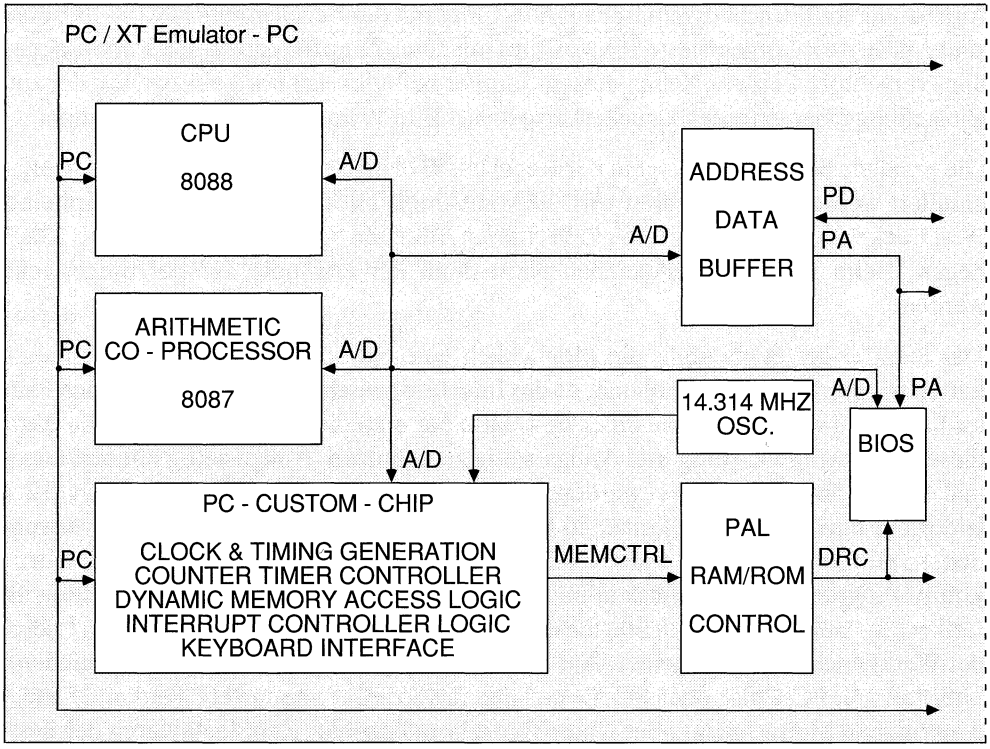
Aufrüstung sollten acht dynamische RAM-Chips mit der Bezeichnung XX256-150, dies sind 256Kbyte x 1 organisierte RAM-Chips mit einer Zugriffszeit von 150 Nano-Sekunden, verwendet werden. Neben diesem Jumper befindet sich noch ein zweiter, der zur Einstellung einer externen Druckerkarte dient. Bild 19 im Farbteil zeigt die Jumper.

Die parallele Schnittstelle liegt in der Regel bei $378 bis $37Fhex. Sie kann vom Amiga emuliert werden. Eingeschaltet wird sie vom Amiga mit dem Utility LPT1. Wird eine PC-Druckerkarte verwendet, so müssen deren Adressen bei LPT2, d.h. $3BC bis $3BF liegen. Beim Einbau einer solchen Karte muß der erwähnte Jumper umgesteckt werden.

Das SideCar bietet ebenfalls die Möglichkeit eine Harddisk zu installieren. Anfangs hatten wir hierbei einige Probleme, da das Interface zwischen Amiga und PC einen Fehler hatte. Mit der Installation einer Festplatte hat man die faszinierende Möglichkeit, diese gleichzeitig PC-seitig und Amiga-seitig zu benutzen. Amiga-seitig können maximal 4 Partitionen eingerichtet werden. PC-seitig erlaubt das mitgelieferte MS-DOS 2.1 leider nur eine Partition mit max. 20 Mbyte. Möchte man größere Harddisks verwenden, so empfielt es sich mit einer anderen MS-DOS-Version zu arbeiten und mit einem Utility-Programm die Harddisk in mehrere Partitionen aufzuteilen. Ist alles richtig installiert, so kann es nach dem Starten des Amiga 1000 mit SideCar und nach dem Laden der Workbench richtig losgehen. Auf der Workbench findet man hierzu verschiedene Utilities. Mit PC-Color oder PC-Mono kann das SideCar »gestartet« werden, je nachdem ob man Monochrom- oder Color-Video installiert hat. Zum Installieren der Festplatte muß nach »BINDRIVERS« noch »DJMOUNT« im CLI eingegeben werden. Mit »ASSIGN« kann dann festgestellt werden, ob dieses »Andocken« gelungen ist. Dazu müssen Sie natürlich vorher im MS-DOS ihre Amiga-Partition(en) mit ADISK eingerichtet haben. Auch das MS-DOS-Laufwerk läßt sich Amiga-seitig benutzen. Hier gab es jedoch anfangs noch einige Schwierigkeiten, die aber inzwischen behoben sind. Schwierigkeiten wird man ebenfalls mit der Belegung der Tastatur haben, da die Amiga 1000-Tastatur nicht IBM-kompatibel ist, sind im PC-Betrieb einige wichtige Tasten nicht auf Anhieb zu finden:

PC-Taste	Amiga-SideCar
Num Lock	rechte Amiga + N
Scroll Lock	rechte Amiga + S
Schift+PRT SC*	rechte Amiga + Shift + P
»+« auf keypad	rechte Amiga + »+«

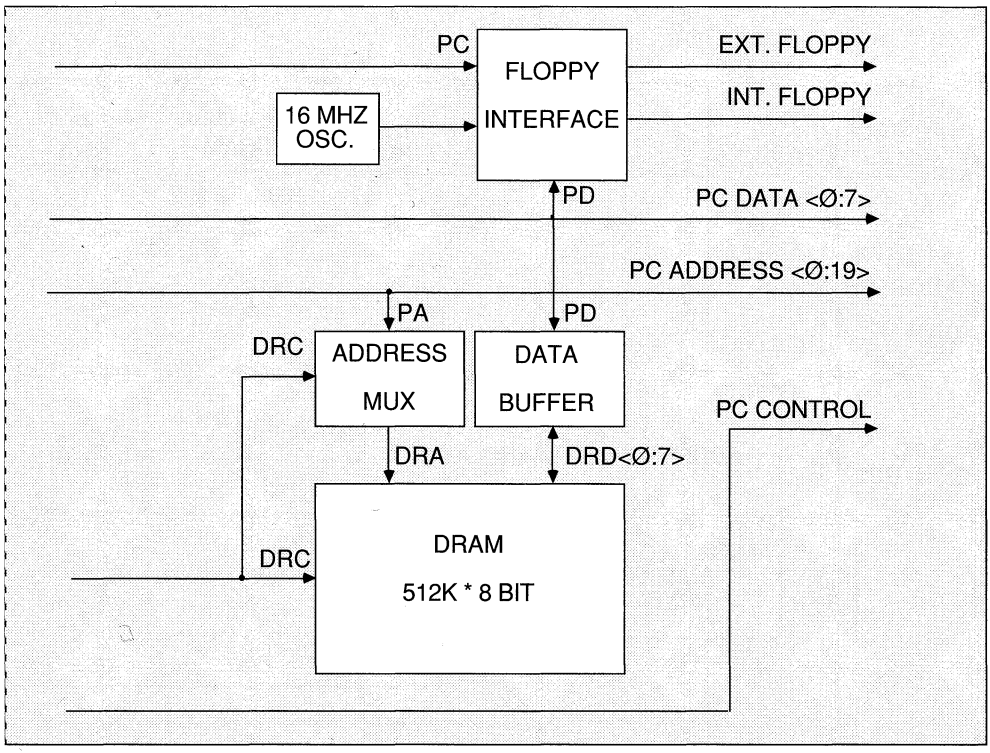
Alles in allem bietet sich dem Anwender mit dem SideCar ein kompaktes Entwicklersystem, mit dem es sich professionell arbeiten läßt.



Z 12.2-1: *Nur vom Schema der PC-Karte läßt sich noch erkennen, daß diese kleine Karte einen vollwertigen PC darstellt (Teil 1)*

12.1.1: Das SideCar am Amiga 500

Das SideCar ist grundsätzlich für den Amiga 1000 entwickelt worden. Beim Anschluß an den Amiga 500 entstehen dabei ein paar Probleme. Das erste Problem, das auftaucht, ist der, beim Amiga 500, um 180 Grad gedrehte Systembus. Wenn man das SideCar nicht gerade um 180 Grad gedreht aufstellen will, empfiehlt es sich, einen Adapter anzuschaffen oder selbst anzufertigen. Das nächste Problem, das auftaucht, ist die Spannungsversorgung. Beim SideCar ist ein Netzanschluß für den Amiga 1000 durchgeschleift worden, damit sichergestellt ist, daß der Amiga immer vor dem SideCar eingeschaltet wird. Möchte man dies auch beim Amiga 500 realisieren, so muß die Zuleitung zum Netzteil des A500 etwas umgebaut werden. Hierbei sei aber zu empfehlen, daß ein solcher Eingriff auch wirklich nur von Kennern des Elektro-Handwerks vorgenommen wird, denn ein Fehler an diesem Teil könnte dazu führen, daß man keinen weiteren mehr macht. Die zweite, vieleicht bessere Möglichkeit, die sich bietet, ist, daß der Amiga 500 separat immer zuerst eingeschaltet wird und erst dann das SideCar. Ist hardwaremäßig alles klar, so braucht man sich nur noch die Workbench zum SideCar einzurichten und schon kann man mit ihm arbeiten.



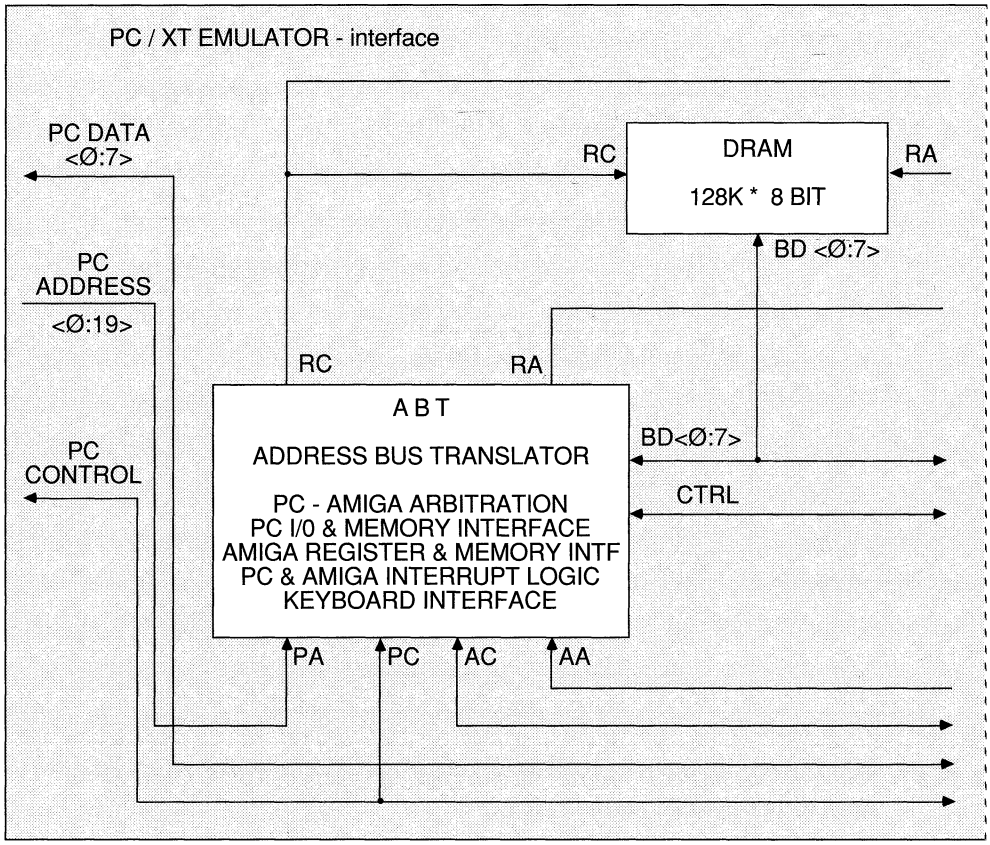
Z 12.2-1: *Nur vom Schema der PC-Karte läßt sich noch erkennen, daß diese kleine Karte einen vollwertigen PC darstellt (Teil 2)*

12.2: Die PC/XT-Karte

Die PC-XT Karte stellt für den Amiga 2000 einen kompletten PC/XT-kompatiblen Personalcomputer dar, der aus 512 Kbyte RAM, einer 8088 CPU mit einer Taktfrequenz von 4.77 MHz, dem notwendigen Timer und DMA-Controller, besteht. Ein freier Sokkel für einen 8087 Math-CO-Prozessor ist ebenfalls vorhanden (Foto 13 im Farbteil).

Ist diese Karte in den Amiga 2000 eingesteckt, so werden alle Slots links der Karte (von vorne gesehen) zu PC-Slots. Diese Slots sind voll kompatibel zu den PC/XT-Slots von IBM. Auf der Emulator-Karte selbst wurde stark aufgeräumt. Eine Parallele zum SideCar läßt sich kaum noch feststellen, da wichtige Teile auf dieser Karte in 3 große Custom-Chips verpackt wurden. Das Schema des PCs ist ansonsten gleich geblieben.

Einer der *Custom-Chips* enthält, wie aus der Schematik erkennbar ist, den DMA- und Interrupt-Controller, sowie den Timer. Die anderen beiden Chips ersetzen eine große Platine, die beim SideCar als Adreßbus- und Datenbus-Translator diente, der die Verbindung zur Amiga-Ebene bewerkstelligte. Die Lage des Dual-Ported-RAM auf der

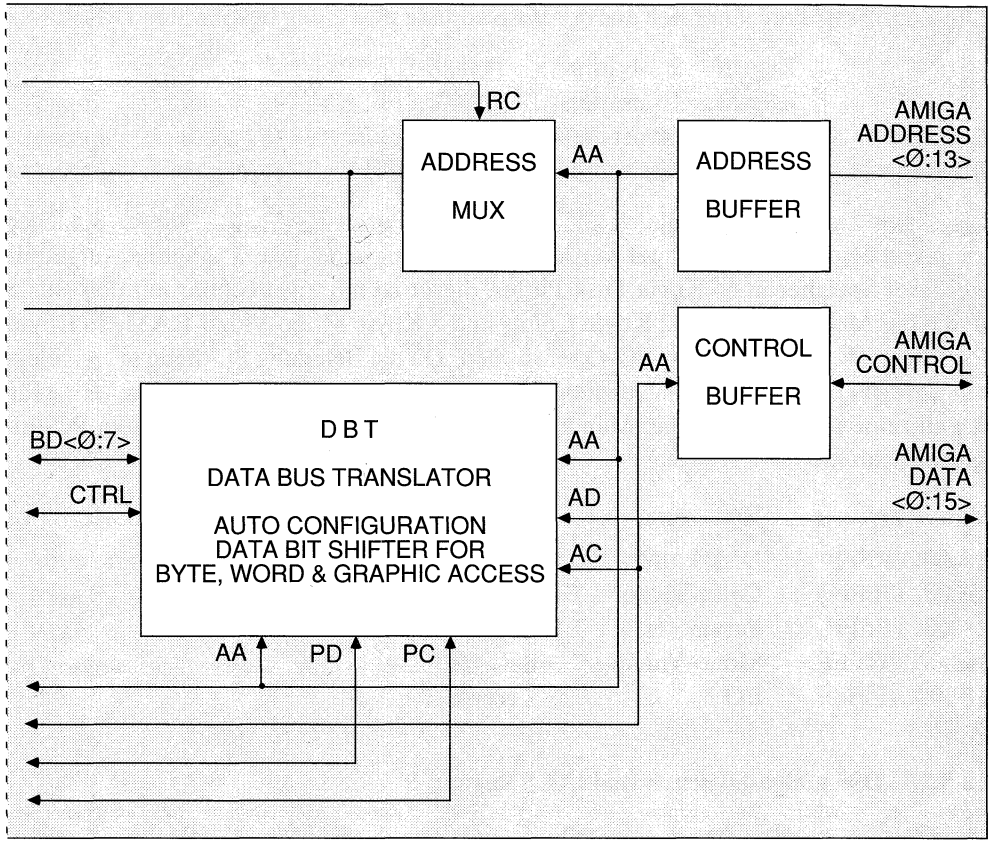


Z 12.2-2: *Das PC/XT-Emulator-Interface in schematischer Darstellung (Teil 1)*

PC-Seite wird bei dieser Karte per Register eingestellt (MODE-Register), was beim SideCar noch per Dip-Schalter von Hand erledigt werden mußte. In dem 128 Kbyte großen Dual-Ported-RAM werden alle Informationen zwischen Amiga und PC hin- und hertransferiert (Farbteil Bild 17).

Schnittstellen, die nach außen führen, wie beispielsweise eine Drucker- oder serielle Schnittstelle, werden vom Amiga emuliert. Diese sind nun an die IBM-Norm angepaßt, so daß keine Spezialkabel nötig sind. Alles im allem stellt diese Karte einen vollwertigen PC-10 dar, der sich ohne Schwierigkeiten voll kompatibel zum IBM-PC/XT verhält. Hier ein Überblick der wichtigsten Daten:

- Prozessor 8088
- Taktfrequenz 4.77 MHz
- Interfaces: 1 internes Floppy 5 1/4" 360/720 Kbyte
  1 internes Floppy 5 1/4" 360/720 Kbyte



Z 12.2-2: *Das PC/XT-Emulator-Interface in schematischer Darstellung (Teil 2)*

MS-DOS formatiert
Parallel-Port (AMIGA-seitig)
- 8087 Co-Prozessor Steckplatz
- benutzt AutoConfig-Funktion des Amiga
- 3 PC/XT-kompatible Steckplätze für Vollformat
- Arbeitsspeicher 512 Kbyte
- Festwertspeicher 16 Kbyte BIOS ROM

12.2.1: Die Speicher- und I/O-Belegung der PC/XT-Karte

Für gewisse Manipulation oder bei direktem Zugriff auf die Speicher- und I/O-Adressen der Amiga- und PC-Seite, ist es von großen Vorteil, die genaue Belegung und Zugriffsart, ob Byte, Word oder Grafik, zu wissen. PC-seitig sind diese Adressen fest eingestellt. Beim Amiga muß hingegen ein gewisser Offset hinzuaddiert werden.

Die Zugriffsart berechnet sich aus der Basisadresse plus einen bestimmten Offset:

Byte – Zugriff = Basisadresse + 00000..1FFFF
Word – Zugriff = Basisadresse + 20000..3FFFF
Graphik – Zugriff = Basisadresse + 40000..5FFFF
I/O Register – Zugriff = Basisadresse + 60000..7FFFF

Insgesamt hat der PC-Emulator 128 Kbyte Dual-Portet-RAM, d.h. dieser Speicher kann von beiden Seiten benutzt werden, sowohl Amiga-, als auch PC-seitig. Aufgeteilt ist dieser Speicher in 64 Kbyte Disk-Puffer, dieser ist nur ansprechbar mit den Bits 5 und 6 des Mode-Register, 32 Kbyte Color- und 8 Kbyte Mono-Video-RAM, 16 Kbyte Parameter- RAM und 8K für I/O-Operationen. Da die Belegung PC-seitig in festgelegten Bereichen verschiebbar ist, enstehen verschiedene Kombinationsmöglichkeiten für die einzelnen RAM-Bereiche. Man kann so die Belegung einmal Amiga- oder PC-seitig betrachten.

Basisadresse	Benutzung	Größe
00000..0FFFF	Disk-Puffer	64 Kbyte
10000..17FFF	Color-Video	32 Kbyte
18000..1BFFF	Parameter	16 Kbyte
1C000..1DFFF	Mono-Video	8 Kbyte
1E000..1FFFF	I/O	8 Kbyte

12.2.1.1: Die Amiga-Speicher- und I/O-Belegung

Zugriffs-art	Amiga-Offset	PC-Adresse	Benutzung	Größe
Byte	00000..0FFFF	A0000..AFFFF	Disk-Puffer	64 Kbyte
Byte	00000..0FFFF	D0000..DFFFF	Disk-Puffer	64 Kbyte
Byte	00000..0FFFF	E0000..EFFFF	Disk-Puffer	64 Kbyte
Byte	10000..17FFF	B8000..BFFFF	Color-Video	32 Kbyte
Byte	18000..1BFFF	F0000..F3FFF	Parameter	16 Kbyte
Byte	1C000..1DFFF	B0000..B1FFF	Mono-Video	8 Kbyte
Byte	1E000..1FFFF	00000..003FF	I/O	1 Kbyte
Word	20000..2FFFF	A0000..AFFFF	Disk-Puffer	64 Kbyte
Word	20000..2FFFF	D0000..DFFFF	Disk-Puffer	64 Kbyte
Word	20000..2FFFF	E0000..EFFFF	Disk-Puffer	64 Kbyte
Word	30000..37FFF	B8000..BFFFF	Color-Video	32 Kbyte
Word	38000..3BFFF	F0000..F3FFF	Parameter	16 Kbyte
Word	3C000..3DFFF	B0000..B1FFF	Mono-Video	8 Kbyte
Word	3E000..3FFFF	00000..003FF	I/O	1 Kbyte

Zugriffs-Art	Amiga-Offset	PC-Adresse	Benutzung	Größe
Graphic	40000..4FFFF	A0000..AFFFF	Disk-Puffer	64 Kbyte
Graphic	40000..4FFFF	D0000..DFFFF	Disk-Puffer	64 Kbyte
Graphic	40000..4FFFF	E0000..EFFFF	Disk-Puffer	64 Kbyte
Graphic	50000..57FFF	B8000..BFFFF	Color-Video	32 Kbyte
Graphic	58000..5BFFF	F0000..F3FFF	Parameter	16 Kbyte
Graphic	5C000..5DFFF	B0000..B1FFF	Mono-Video	8 Kbyte
Graphic	5E000..5FFFF	00000..003FF	I/O	1 Kbyte
	7E000..7FFFF	00000..003FF	I/O	1 Kbyte

12.2.1.2: Die PC-Speicher- und I/O-Belegung

Zugriffs-art	PC-Offset	Amiga-Offset	Benutzung	Größe
Byte	00000..003FF	1E000..1FFFF	I/O	1 Kbyte
Word		3E000..3FFFF		
Graphik		5E000..5FFFF		
		7E000..7FFFF		
Byte	A0000..AFFFF	00000..0FFFF	Disk-Puffer	64 Kbyte
Word		20000..2FFFF		
Graphik		40000..4FFFF		
Byte	B0000..B1FFF	1C000..1DFFF	Mono-Video	8 Kbyte
Word		3C000..3DFFF		
Graphik		5C000..5DFFF		
Byte	B8000..BFFFF	10000..17FFF	Color-Video	32 Kbyte
Word		30000..37FFF		
Graphik		50000..57FFF		
Byte	E0000..EFFFF	00000..0FFFF	Disk-Puffer	64 Kbyte
Word		20000..2FFFF		
Graphik		40000..4FFFF		
Byte	D0000..DFFFF	00000..0FFFF	Disk-Puffer	64 Kbyte
Word		20000..2FFFF		
Graphik		40000..4FFFF		
Byte	F0000..F3FFF	18000..1BFFF	Parameter	16 Kbyte
Word		38000..3BFFF		
Graphik		58000..5BFFF		

12.3: Die AT-Karte

Wo ein PC-Kompatibler ist, da ist der AT-Kompatible nicht weit. Nach der PC-kompatiblen Karte für den Amiga 2000 folgte die AT-kompatible. Sie ist fast genauso aufgebaut wie die PC-Karte. Unterschiede finden sich lediglich in der Formatierung von Disketten, im Prozessorsystem, sowie in der Taktfrequenz. Sollte man sich nicht sicher sein, ob man eine AT- oder PC-kompatible Karte besitzt, kann dies mit dem *MODE-Register* getestet werden. Hier die wichtigsten Daten:

- Prozessor 80286
- Taktfrequenz 8 MHz
- Interfaces: 1 internes Floppy 5 1/4" 1,2 Mbyte
  1 internes Floppy 3 1/2" 720 Kbyte
  MS-DOS formatiert
  Parallel-Port (Amiga-seitig)
- 80287 Co-Prozessor Steckplatz
- benutzt AutoConfig-Funktion des Amiga
- 3 AT-kompatible Steckplätze für Vollformat
- Arbeitsspeicher 512 Kbyte
- Festwertspeicher 16 Kbyte BIOS ROM
- Tastatur XT- bzw. AT-kompatibel durch Emulation des Amiga

12.3.1: Die Speicher- und I/O-Belegung der AT-Karte

Die AT-Karte hat ebenfalls, wie schon die PC-Karte, insgesamt 128 Kbyte Dual-Ported-RAM. Dieser Speicher ist, wie bei der PC-Karte, in 6 4Kbyte Disk-Puffer, dieser ist nur ansprechbar mit den Bits 5 und 6 des Mode-Registers, 32 Kbyte Color- und 8 Kbyte Mono-Video- RAM, 16 Kbyte Parameter-RAM und 8 Kbyte für I/O-Operationen unterteilt. Die Berechnung der Zugriffsart, Byte, Word oder Graphic, sowie die Basisadresse des Interfaces, die das Offset bestimmt, sind im Vergleich zur PC-Karte nahezu identisch. Ein Unterschied besteht lediglich darin, daß bei der AT-Karte das Dual-Ported-RAM auf einen gewissen Bereich nicht eingestellt werden kann, da dieser für den größeren Speicher der Karte verwendet wird.

Da die Adressen hier ebenfalls verschiebbar sind, bieten sich auch hier verschiedene Kombinationsmöglichkeiten für die einzelnen Bereiche. Die Belegung kann auch hier Amiga- oder PC-seitig betrachtet werden. Die Belegung für PC- und AT-Karten ist allerdings nicht völlig identisch, da der AT beispielsweise seinen Disk-Puffer in einem anderen Bereich haben muß.

12.3.1.1: Die Amiga-Speicher- und I/O-Belegung

Zugriffs-art	Amiga-Offset	AT-Adresse	Benutzung	Größe
Byte	00000..0FFFF	A0000..AFFFF	Disk-Puffer	64 Kbyte
Byte	00000..03FFF	Zugriff von AT-Karte nicht möglich		
Byte	04000..0FFFF	D4000..DFFFF	Disk-Puffer	48 Kbyte
Byte	10000..17FFF	B8000..BFFFF	Color-Video	32 Kbyte
Byte	18000..1BFFF	D0000..D3FFF	Parameter	16 Kbyte
Byte	1C000..1DFFF	B0000..B1FFF	Mono-Video	8 Kbyte
Byte	1E000..1FFFF	00000..003FF	I/O	1 Kbyte
Word	20000..2FFFF	A0000..AFFFF	Disk-Puffer	64 Kbyte
Word	20000..23FFF	Zugriff von AT-Karte nicht möglich		
Word	24000..2FFFF	D4000..DFFFF	Disk-Puffer	48 Kbyte
Word	30000..37FFF	B8000..BFFFF	Color-Video	32 Kbyte
Word	38000..3BFFF	D0000..D3FFF	Parameter	16 Kbyte
Word	3C000..3DFFF	B0000..B1FFF	Mono-Video	8 Kbyte
Word	3E000..3FFFF	00000..003FF	I/O	1 Kbyte
Graphic	40000..4FFFF	A0000..AFFFF	Disk-Puffer	64 Kbyte
Graphic	40000..43FFF	Zugriff von AT-Karte nicht möglich		
Graphic	44000..4FFFF	D4000..DFFFF	Disk-Puffer	8 Kbyte
Graphic	50000..57FFF	B8000..BFFFF	Color-Video	32 Kbyte
Graphic	58000..5BFFF	D0000..D3FFF	Parameter	16 Kbyte
Graphic	5C000..5DFFF	B0000..B1FFF	Mono-Video	8 Kbyte
Graphic	5E000..5FFFF	00000..003FF	I/O	1 Kbyte
	7E000..7FFFF	00000..003FF	I/O	1 Kbyte

Die AT-Speicher- und I/O-Belegung

Zugriffs-art	PC-Offset	Amiga-Offset	Benutzung	Größe
Byte	00000..003FF	1E000..1FFFF	I/O	1 Kbyte
Word		3E000..3FFFF		
Graphik		5E000..5FFFF		
		7E000..7FFFF		
Byte	A0000..AFFFF	00000..0FFFF	Disk-Puffer	64 Kbyte
Word		20000..2FFFF		
Graphik		40000..4FFFF		
Byte	B0000..B1FFF	1C000..1DFFF	Mono-Video	8 Kbyte
Word		3C000..3DFFF		
Graphik		5C000..5DFFF		

Zugriffs-art	PC-Offset	Amiga-Offset	Benutzung	Größe
Byte	B8000..BFFFF	10000..17FFF	Color-Video	32 Kbyte
Word		30000..37FFF		
Graphik		50000..57FFF		
Byte	D0000..D3FFF	18000..1BFFF	Parameter	16 Kbyte
Word		38000..3BFFF		
Graphik		48000..5BFFF		
Byte	D4000..DFFFF	04000..0FFFF	Disk-Puffer	64 Kbyte
Word		24000..2FFFF		
Graphik		44000..4FFFF		

12.4: Die PC/AT-I/O-Register

Um beispielsweise eine Umleitung des Drucker-Portes vorzunehmen, ist es nützlich zu wissen, welche Ein- und Ausgabeadressen man Amiga-seitig ansprechen muß, um eine bestimmte PC/AT-IO-Adresse zu bekommen.

Bei der Entwicklung von PC-Karten stehen nur max. 512 von 1024 Ein- und Ausgabe-Adreßleitungen zur Verfügung, wobei nur die Adreßleitungen A0 bis A9 Verwendung finden. Eine wichtige Rolle hierbei spielt die Adreßleitung A9.

A9 schaltet beim Zugriff auf diesen Ein- und Ausgabebereich zwischen Motherboard- und Erweiterungskarten I/O-Bereich um. Ist A9 logisch low, d.h. gleich 0V, so werden nur die ersten 512 I/O-Portadressen auf dem PC-Motherboard aktiviert. Ist A9 logisch high, d.h. +5V, stehen die restlichen 512 Portadressen für Erweiterungskarten zur Verfügung. Die übliche PC/AT-Belegung des I/O-Bereiches:
Ein-/Ausgabeadresse

(I/O in Hex)	A9	Funktion
0000 ... 001F	0	DMA-Controller #1
0020 ... 003F	0	Interrupt-Contoller (Master)#1
0040 ... 005F	0	Zähler/Zeitgeber
0060 ... 006F	0	Systemregister
0070 ... 007F	0	Real-Time-Clock
0080 ... 009F	0	DMA-Seitenregister/NMI Mask
00A0...00BF	0	Interrupt-Controller (Slave) #2
00C0 ... 00DF	0	DMA-Controller #2
00F0	0	Clear 8087/80287 Busy
00F1	0	Reset 8087/80287
00F8 ... 00FF	0	8087/80287 Math Co-Prozessor
0100 ... 01EF	0	nicht spezifiziert

(I/O in Hex)	A9	Funktion
01F0 ... 01F8	0	Festplatten-Controller
0200 ... 0207	1	Game-Port
0278 ... 027F	1	zweiter Drucker
02F8 ... 02FF	1	zweite serielle Schnittstelle
0300 ... 031F	1	Prototypkarte
0329 ... 032F	1	Festplatten-Controller
0378 ... 037F	1	Parallel-Port/Drucker
03B0 ... 03BF	1	Monochromadapter/Drucker
03D0 ... 03DF	1	Farbgrafikkarte
03F0 ... 03F7	1	Floppy-Controller
03F8 ... 03FF	1	serielle Schnittstelle

Hier nun die wichtigsten Register im einzelnen:

PC/AT
I/O Offset/Basisadr.

Adr.	Interface	Amiga	Status	Kommentar
Systemregister:				
060	1E41F	7E41F	W	Keyboard Daten
061	1E05F	1E05F	W	System-Register
062	1E03F	7E03F	W	System-Status
serielle Schnittstelle:				
2F8	1E07D	7E07D	W	COM2 Sende Daten
2F8	1E09D	7E09D	R	COM2 Empfang Daten
2F8	1E09D	7E09D	R	COM2 Reset IRQ3_b
2F9	1E0BD	7E0BD	W	COM2 Interruptkontrolle
2F9	1E0DD	7E0DD	R	COM2 Interruptkontrolle
2FB	1E07F	7E07F	R/W	COM2 Divisor Latch (LSB)
2FB	1E07F	7E07F	R	COM2 Interrupt Ackn.
2F9	1E09F	7E09F	R/W	COM2 Divisor Latch (MSB)
2FA	1E0FF	7E0FF	R	COM2 Interrupt bestätigen
2FA	1E01F	7E01F	W	COM2 dummy Adresse
2FB	1E11F	7E11F	W	COM2 Line-Control
2FB	1E01F	7E01F	R	COM2 dummy Adresse
2FC	1E13F	7E13F	W	COM2 Modem-Control
2FC	1E01F	7E01F	R	COM2 dummy Adresse
2FD	1E15F	7E15F	R	COM2 Line-Status
2FD	1E01F	7E01F	W	COM2 dummy Adresse
2FE	1E17F	7E17F	R	COM2 Modem-Status
2FE	1E01F	7E01F	W	COM2 dummy Adresse
2FF	1E01F	7E01F	R/W	COM2 dummy Adresse

Adr.	Interface	Amiga	Status	Kommentar
Printer:				
378	1E19F	7E19F	R/W	LPT1 Printer Daten
379	1E1BF	7E1BF	R	LPT1 Status
379	1E1BF	7E1BF	R	LPT1 Reset IRQ7
379	1E19F	7E19F	W	LPT1 Interruptkontrolle
				Bit 6 = 1 Interrupt ein
				Bit 6 = 0 Interrupt aus
37A	1E1DF	7E1DF	W	LPT1 Kontrolle
37A	1E19F	7E19F	R	LPT1 Kontrolle
Mono CTR:				
3B0	1E1FF	7E1FF	W	MONO CRT Adr. Indexregister
3B0	1E01F	7E01F	R	MONO Reset IRQ3_a
3B2	1E1FF	7E1FF	W	MONO CRT Adr. Indexregister
3B2	1E01F	7E01F	R	MONO dummy Adresse
3B4	1E1FF	7E1FF	W	MONO CRT Adr. Indexregister
3B4	1E01F	7E01F	R	MONO dummy Adresse
3B6	1E1FF	7E1FF	W	MONO CRT Adr. Indexregister
3B6	1E01F	7E01F	R	MONO dummy Adresse
3B1	Adressen teilen		R/W	MONO CRT Datenregister
3B3	sich hier, nach-		R/W	MONO CRT Datenregister
3B5	dem in das letzte		R/W	MONO CRT Datenregister
3B7	geschrieben Index		R/W	MONO CRT Datenregister
	auf:			
	1E2A1	7E2A1		Index = 00
	1W2A3	7E2A3		Index = 01
	1E2A5	7E2A5		Index = 02
	1E2A7	7E2A7		Index = 03
	1E2A9	7E2A9		Index = 04
	1E2AB	7E2AB		Index = 05
	1E2AD	7E2AD		Index = 06
	1E2AF	7E2AF		Index = 07
	1E2B1	7E2B1		Index = 08
	1E2B3	7E2B3		Index = 09
	1E2B5	7E2B5		Index = 0A
	1E2B7	7E2B7		Index = 0B
	1E2B9	7E2B9		Index = 0C
	1E2BB	7E2BB		Index = 0D
	1E2BD	7E2BD		Index = 0E
	1E2BF	7E2BF		Index = 0F
3B8	1E2FF	7E2FF	W	MONO Kontrollregister

Adr.	Interface	Amiga	Status	Kommentar
3BA	-----	-----	R	MONO Statusregister Bit 0 = H-Sync (18KHz) Bit 3 = V-Sync (50 Hz)
3BA	1E01F	7E01F	W	dummy Adresse
3BB	1E01F	7E01F	R/W	dummy Adresse
3BC	1E01F	7E01F	R/W	dummy Adresse
3BD	1E01F	7E01F	R/W	dummy Adresse
3BE	1E01F	7E01F	R/W	dummy Adresse
3BF	1E01F	7E01F	R/W	dummy Adresse
Color CRT:				
3D0	1E21F	7E21F	W	COLOR CRT Adr. Indexregister
3D0	1E01F	7E01F	R	dummy Adresse
3D2	1E21F	7E21F	W	COLOR CRT Adr. Indexregister
3D2	1E01F	7E01F	R	dummy Adresse
3D4	1E21F	7E21F	W	COLOR CRTAdr. Indexregister
3D4	1E01F	7E01F	R	dummy Adresse
3D6	1E21F	7E21F	W	COLOR CRT Adr. Indexregister
3D7	1E01F	7E01F	R	dummy Adresse
3D1	Adressen teilen		R/W	COLOR CRT Datenregister
3D3	sich hier, nach-		R/W	COLOR CRT Datenregister
3D5	dem in das letzte		R/W	COLOR CRT Datenregister
3D7	geschrieben Index		R/W	COLOR CRT Datenregister
	auf:			
	1E2C1	7E2C1		Index = 00
	1W2C3	7E2C3		Index = 01
	1E2C5	7E2C5		Index = 02
	1E2C7	7E2C7		Index = 03
	1E2C9	7E2C9		Index = 04
	1E2CB	7E2CB		Index = 05
	1E2CD	7E2CD		Index = 06
	1E2CF	7E2CF		Index = 07
	1E2D1	7E2D1		Index = 08
	1E2D3	7E2D3		Index = 09
	1E2D5	7E2D5		Index = 0A
	1E2D7	7E2D7		Index = 0B
	1E2D9	7E2D9		Index = 0C
	1E2DB	7E2DB		Index = 0D
	1E2DD	7E2DD		Index = 0E
	1E2DF	7E2DF		Index = 0F
3D8	1E23F	7E23F	W	COLOR Kontrollregister

Adr.	Interface	Amiga	Status	Kommentar
3D8	1E01F	7E01F	R	dummy Adresse
3D9	1E25F	7E25F	W	COLOR Selectregister
3D9	1E01F	7E01F	R	dummy Adresse
3DA	-----	-----	R	COLOR Statusregister Bit 0 = H-Sync (18 KHz) Bit 3 = V-Sync (50 Hz)
3DA	1E01F	7E01F	W	dummy Adresse
3DD	1E29F	7E29F	W	Display Systemregister
3DD	1E01F	7E01F	R	dummy Adresse
3DE	1E01F	7E01F	R/W	dummy Adresse
3DF	1E01F	7E01F	R/W	dummy Adresse

8 wichtige Register haben wir in dieser Tabelle absichtlich ausgelassen, da sie nur von der Amiga-Seite zugreifbar sind:

	Offset/Basisadr.		
Amiga-Register	Interface	Amiga	Status
Amiga Interrupt Status	1FFF1	7FFF1	R
PC Interrupt Status	1FFF3	7FFF3	R
PC Reset unwirksam machen	1FFF5	7FFF5	R
Mode Register	1FFF7	7FFF7	R/W
Interrupt Mask	1FFF9	7FFF9	R/W
PC Interrupt Control	1FFFB	7FFFB	R/W
Kontrollregister	1FFFD	7FFFD	R/W
Keyboard-Register	1FFFF	7FFFF	R/W

Die Register im einzelnen:

AMIGA Interrupt Status (1FFF1/7FFF1 Read)
Wenn dieses Register gelesen wird, erhält man die augenblicklichen Interrupt-Ereignisse des PCs. Das Ereignis war gültig, wenn das jeweilige Bit gesetzt wurde. Nach dem Lesen dieses Registers werden alle Bits auf 0 gesetzt und das Interrupt-Flag wird negiert. Hier die Interrupt-Bits im einzelnen:

Bit Nr.	Funktion
0	Mono Video RAM
1	Color Video RAM
2	Mono CRT
3	Color CRT
4	Keyboard Register
5	LPT1 Kontrollregister
6	COM2 Dataregister
7	PC System Status Reset

PC Interrupt Status (1FFF3/7FFF3 Read)
Beim Lesen dieses Registers erhält man die augenblicklichen Interrupts des PCs. Da PC-seitig nur vier Interrupt-Requests möglich sind, IRQ1, IRQ3_a, IRQ3_b, IRQ7, wird nur das untere Nibble, d.h. die untern vier Bits benötigt:

Bit Nr.	Funktion
0	IRQ1 (Keyboard Interrupt)
1	IRQ3_a
2	IRQ3_b
3	IRQ7
4–7	werden nicht benötigt und sind immer

High, d.h. +5VNegate PC Reset (1FFF5/7FFF5Read)

Wenn auf diese Adresse zugegriffen wird, wird die PC-Reset-Leitung negiert, was ein Starten der PC-Boot-Procedure bewirkt.

Mode Register (1FFF7/7FFF7 Read/Write)

Mit diesem Register kann der Anwender beim Lesen Systeminformationen erhalten, bzw. beim Schreiben eine neue Systemkonfiguration setzen. Beim SideCar ist dies nicht möglich, da hier die Konfiguration mit Dip-Schaltern auf dem PC-Motherboard eingestellt wird.

Lesezugriff auf das Mode-Register:

Bit Nr.	Name	Funktion
0	SERON	serielle Schnittstelle ein/aus
1	PARON	parallele Schnittstelle ein/aus
2	KEYON	Keyboard-Schnittstelle ein/aus
3	MON	Mono Display ein/aus
4	COLOR	Color Display ein/aus
5	SEL1	bestimmt PC/AT-Speicher-Bank
6	SEL2	bestimmt PC/AT-Speicher-Bank
7	PC/AT	Low = AT-Modus, High = PC-Modus

Schreibzugriff auf das Mode-Register:

Bit Nr.	Name	Funktion
0	SERON	serielle Schnittstelle einschalten
1	PARON	parallele Schnittstelle einschalten
2	KEYON	Keyboard-Schnittstelle einschalten
3	MON	ermöglicht MonoDisplay-Emulation

Bit Nr.	Name	Funktion
4	COLOR	ermöglicht ColorDisplay-Emulation
5	SEL1	bestimmt PC/AT-Speicher-Bank(s.u.)
6	SEL2	bestimmt PC/AT-Speicher-Bank
7	/STOPCLK	HIGH = einschalten der Clock für Video-Wiederholung und Keyboard LOW = ausschalten

SEL1	SEL2	AT-Speicher	PC-Speicher
0	0	----------	----------
0	1	A0000..AFFFF	A0000..AFFFF
1	0	D0000..DFFFF	D4000..DFFFF
1	1	----------	E0000..EFFFF

Interrupt Mask(1FFF9/7FFF9 Read/Write)

Dieses Register dient zum Ausmaskieren von PC-Interrupts. Ausmaskiert wird der jeweilige Interrupt, dessen Bit auf 1 steht.

Bit Nr.	Interrupt
0	MonoVideo RAM
1	ColorVideo RAM
2	Mono CRT
3	Color CRT
4	Keyboard-Register
5	LPT1 Kontrollregister
6	COM2 Data-Register
7	PC System Status Reset

PC Interrupt Control (1FFFB/7FFFB Read/Write)
Dieses Register dient zur Kontrolle der PC-Interrupts. Ein PC-Interrupt kann durch Setzen einer 1 in das jeweilige Interrupt-Bit gültig gemacht werden.

Bit Nr.	Interrupt
0	KBSTART
1	IRQ3 a
2	IRQ3 b
3	IRQ7

Kontroll-Register(1FFFD/7FFFDRead/Write)
Mit diesem Register kann der Anwender allgemeine Kontrollfunktionen ausüben. Die jeweilige Kontrollfunktion wird ausgeführt, wenn eine 0 in das jeweilige Bit geschrieben wird.

Bit Nr.	Funktion
0	Ermöglicht den generellen Interrupt zum Amiga
1	Verhindert den generellen Interrupt zum Amiga
2	Behauptet den PC-Reset
3	Negiert alle PC-Interrupts, ausgenommen den Keyboard-Interrupt
4	Setzt Printer BUSY (379 Hex bit 7) zurück. Gesetzt werdenkanner durch das Schreiben einer 1 in das Bit0 der I/O-Adresse $37A Hex von der PC-Seite aus.

Keyboard Register (1FFFF/7FFFFRead/Write)
Dieses Register dient zum Einstellen der Keyboard- Emulation. Hierzu muß ein Charakter in dieses Register geschrieben werden und in das PC-Interrupt-Kontroll- Register Bit 0 KBSTARTeine 1.

Kapitel 13

RAM-Erweiterungen

Der Amiga ist leider zu einer Zeit entwickelt worden, zu der RAMs (Speicherbausteine) noch recht teuer waren. Man ging von einer Grundkonfiguration von 256 Kbyte, die auf 512 Kbyte erweiterbar war, aus. Für professionelle Anwendung stellte sich der maximal verfügbare Speicher als zu klein heraus. Multitasking, faszinierende Grafiken und ein fetziger Sound brauchen nunmal sehr viel Speicher. So wurden die neuen Amigas mit mehr Speicher ausgestattet (Amiga 500 mit 512 Kbyte, Amiga 2000 mit 1Mbyte), der entsprechend erweiterbar ist.

Da mehr Speicher das A und O für Freaks, Programmierer und Hobbybastler ist, sollte dieses Kapitel nicht unter den Tisch fallen. Schon zu oft erschien die allgemein bekannte »Guru Meditation«, die den fehlenden Speicher anzeigte. Für Amiga 500 und Amiga 1000 sind Erweiterungen im unteren Bereich (256-Kbyte und 768-Kbyte-Erweiterungen für A1000 und 512 Kbyte für A500) sehr einfach realisierbar, da hier alle wichtigen Signale vom Motherboard bereitgestellt werden.

Höhere RAM-Aufrüstungen sind sehr komplex, da hier eine aufwendige Dekodier- oder Refresh-Logik benötigt wird. Für *dynamische RAMs* gibt es inzwischen RAM-Controller-ICs, die die nötigen Refresh-Signale erzeugen, sie sind aber im Vergleich zu einer »selbstgestrickten« Refresh-Logik sehr teuer und schwer erhältlich. Auch hier haben wir in diesem »RAM-Kapitel« ein Beispiel beigefügt. Zuguterletzt seien noch die *statischen RAMs* erwähnt. Hier sei aber gleich gesagt, diese Chips sind sehr teuer (32K8 ca. 20 DM das Stück!!!), so daß recht leicht bei der Anschaffung einer solchen RAM-Aufrüstung Konflikte mit dem Geldbeutel auftreten können.

13.1: Statisch oder dynamisch?

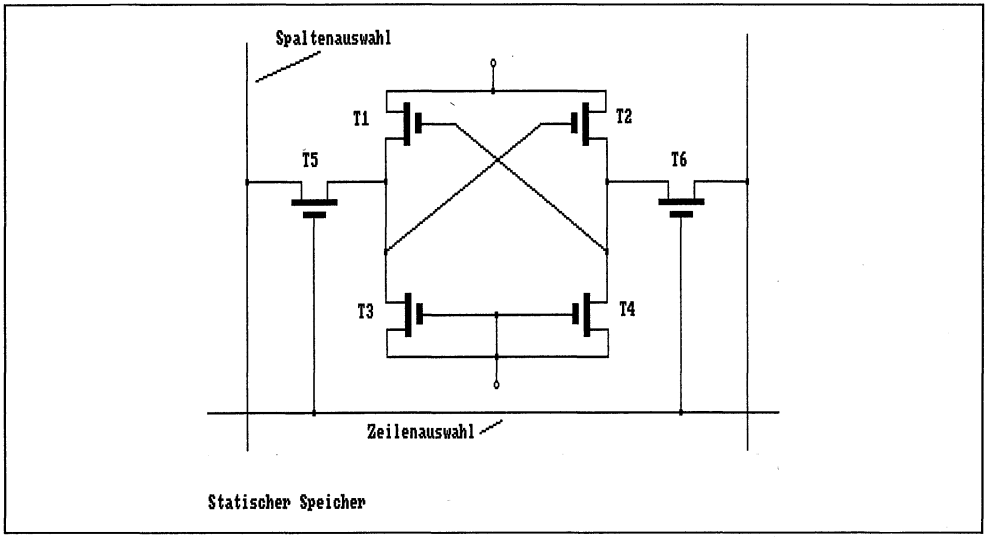
Die Frage, ob statische oder dynamische RAMs verwendet werden sollen, ist nicht nur eine Frage des Geldes. Es ist jedoch nicht zu leugnen, daß statische RAMs um ein Vielfaches teurer sind als dynamische. Oft entscheidet das Einsatzgebiet über die Verwendung verschiedener RAM-Typen. Wird eine Speichererweiterung mit RAMs benötigt, die leicht gegen *EPROMs* austauschbar sind, so werden sicherlich statische RAMs ein-

gesetzt. Wird hingegen viel Speicher auf kleinem Raum benötigt, finden dynamische RAMs Verwendung.

Um nun eine RAM-Aufrüstung für einen Amiga anfertigen zu können, muß zunächst ein Grundwissen über den Aufbau statischer und dynamischer RAM-Chips vorhanden sein, da sie unterschiedliche »Ansprech-Logiken« benötigen.

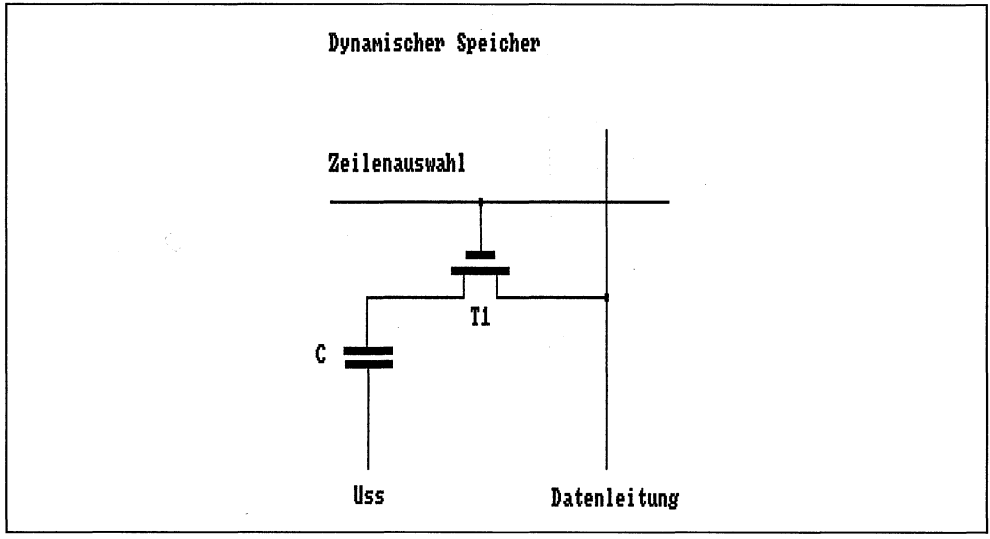
Die bedeutendsten Speicherprinzipien, die sich durchgesetzt haben, sind, wie bisher schon erwähnt, statische und dynamische RAMs. In den Anfangszeiten des Computerzeitalters gab es noch mehr Speicherprinzipien, so z.B. Ferritkern-Speicher. Der Platzbedarf war jedoch im Vergleich zu heutigen enorm groß und die Fabrikationskosten um ein Vielfaches teurer als heute.

Eine statische Speicherzelle besteht im Vergleich zur dynamischen aus einer bistabilen Kippstufe, der man zwei Zustände zuordnen kann (0 und 1). Diese wird durch zwei rückgekoppelte Inverter nachgebildet. Als Lastwiderstände werden aus Platzgründen zwei Transistoren verwendet. Die Zeilenauswahl findet wiederum über zwei Transistoren statt. Diese können über eine Zeilenauswahlleitung leitend geschaltet werden. Anhand des Potentialunterschiedes kann man dann erkennen, ob eine Information anliegt (0 oder 1). Beim Beschreiben dieser Zelle wird über diese Zeilenauswahlleitung die gewünschte Information zugeführt, die danach einen entsprechenden logischen Zustand annimmt (0 oder 1). Abbildung Z 13.1-1 zeigt eine solche Speicherzelle im prinzipiellen Aufbau. Der Nachteil bei dieser Speicherzelle liegt darin, daß für den Aufbau viele Transistoren benötigt werden und daß jeweils über einen Zweig der Kippstufen ständig Strom fließt, was den Leistungsbedarf statischer RAMs stark in die Höhe treibt. Vorteil ist der, daß der gespeicherte Zustand so lange erhalten bleibt, bis die Speicherzelle spannungslos ist. Es wird keine externe Auffrischung der Zelle benötigt. Geschwindigkeitsverluste im Vergleich zu dynamischen bestehen kaum, zumal die neueste Generation statischer RAMs durch Einsatz des Materials Galliumarsenid bei Verwendung der Molekularstrahl-Epitaxie-Verarbeitung extrem schnell ist.



Z 13.1-1: *Dieses Bild zeigt eine statische Speicherzelle im prinzipiellen Aufbau. Es werden sehr viele Bauelemente zum Speichern einer Information benötigt*

Dynamische Speicherzellen sind sehr einfach aufgebaut. Hier werden im Vergleich zu statischen RAMs sehr wenige Bauteile benötigt, was den Preis und den Platzbedarf stark reduziert. Im Prinzip bestehen sie aus einer einzelnen Transistorzelle. Bei dieser Zelle wird die Information in einem Kondensator gespeichert und je nach Polarität in einem Potentialsprung auf der Datenleitung dargestellt (0 oder 1), wenn der zugehörige Transistor über eine Zeilenauswahlleitung leitend geschaltet ist. Abbildung Z 13.1-2 zeigt diesen Aufbau. Nachteil ist jedoch, daß die Information (0 oder 1), die hier durch die Kondensatorladung dargestellt wird, aufgrund der unvermeidbaren Leckströme periodisch regeneriert werden muß. Diese Regenerierung, auch bezeichnet als Refreshing, erfolgt im RAM-Chip durch Lesezyklen auf den Refresh-Adressen. Extern wird also eine Schaltung benötigt, die einen Spalten- und einen Zeilenrefresh erzeugt, damit die gespeicherte Information nicht verloren geht. Der große Vorteil liegt jedoch im sehr einfachen Aufbau einer Speicherzelle, wodurch eine sehr hohe Speicherkapazität auf kleinstem Raum verwirklicht werden kann. Derzeit steht die Entwicklung bei 4-Mega-Bit-Chips und 16-Mega-Bit-Chips sind schon als Laborversionen zu bestaunen. Zusätzlich zu den nötigen *Refresh*-Zyklen müssen die zugeführten Adreßleitungen gemultiplext werden, da zum Ansprechen der Speicherzellen jeweils Zeilenadreßbits und Spaltenadreßbits benötigt werden.

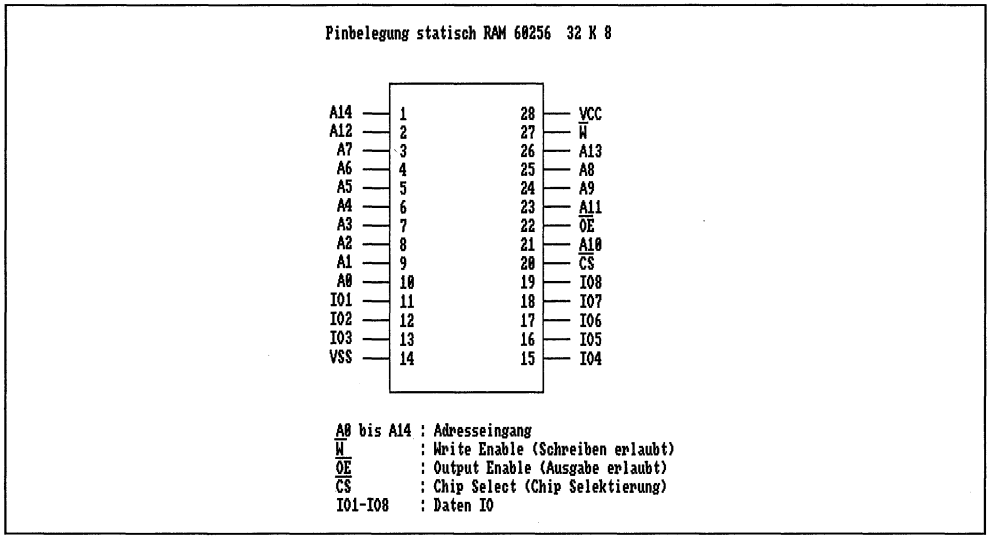


Z 13.1.2: *Dieses Bild zeigt den Aufbau einer dynamischen Speicherzelle. Der einfache Aufbau ermöglicht eine hohe Speicherdichte auf kleinstem Raum*

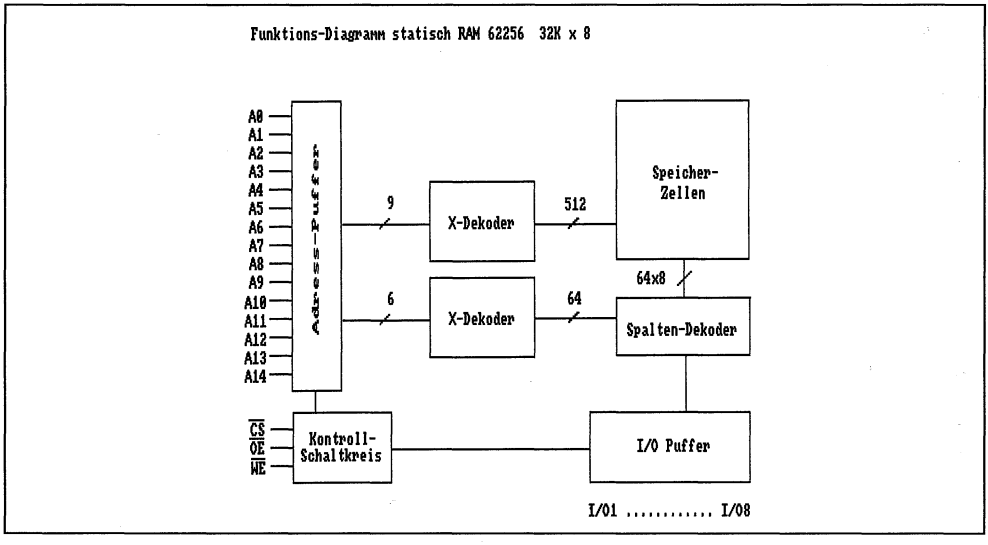
13.2: Statische RAMs am Amiga

Statische RAMs benötigen aufgrund ihres internen Aufbaus keine *Refresh*-Logik. Ein *Multiplexen* der Adreßleitungen enfällt. Die jeweiligen Speicherzellen können hier direkt angesprochen werden. Dies erleichtert die externe Beschaltung ungemein. Zum Ansprechen dieser RAMs wird somit nur eine Chip-Select- und Write/Output-Enable-Logik benötigt.

Zum Anschluß an den Amiga eignen sich im Prinzip alle statischen RAMs, die eine Zugriffszeit von höchstens 150 ns besitzen. Besser geeignet sind allerdings Chips mit 120 ns. Die Zugriffszeit berechnet sich aus dem Systemtakt von 7.14 Mhz und der Formel F (die Frequenz) = 1 / t (Zeit). Die Zeit lautet also t = 140 ns. Speicher mit einer Zugriffszeit von 150 ns sind also nur bedingt geeignet, da hier für eine korrekte Funktion nicht hundertprozentig garantiert werden kann. Von Vorteil ist ebenfalls, wenn statische RAMs xxKBit mal 8, also beispielsweise 32 K 8, organisiert sind, da hier der Anschluß erheblich vereinfacht wird. Ein einzelnes RAM-IC kann so ein komplettes Byte speichern, während ein Byte bei dynamischen Rams auf 8 Bausteine verteilt ist. Für unsere statische RAM-Aufrüstung haben wir uns für 32 Kbit mal 8 RAMs entschieden. Im Fachhandel sind sie unter der Bezeichnung 20256 bzw. 60256 erhältlich. Sie besitzen 15 Adreß-,8 Daten- und 3 Kontrollanschlüsse. Hier eine Pinbelegung und ein Funktionsdiagramm dieser RAM-Bausteine:



Z 13.2-1: *Die Pinbelegung der 32 K 8 statischen RAMs*



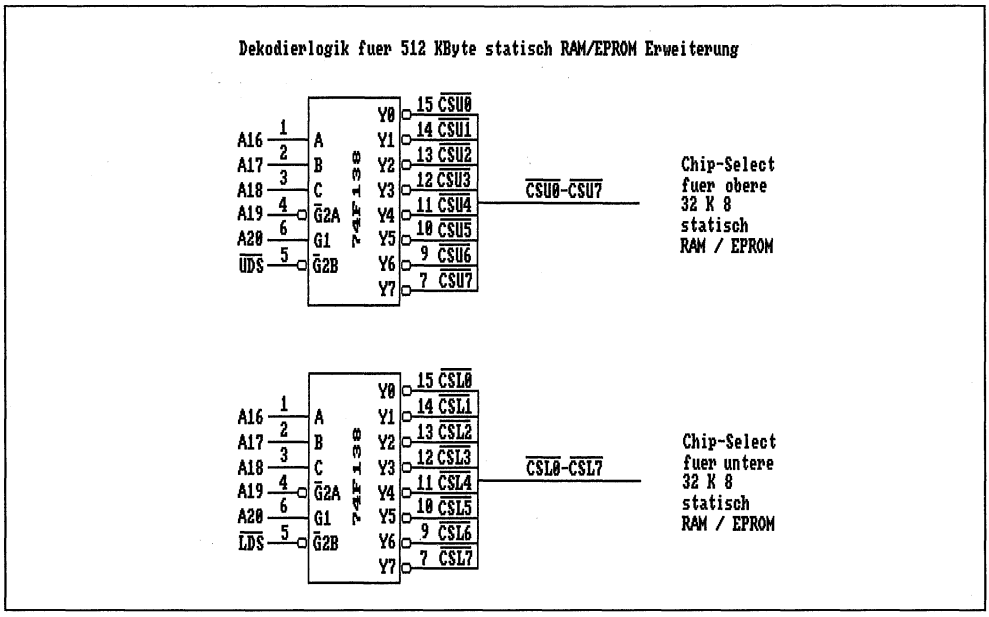
Z 13.2-2: *Dieses Bild zeigt den internen Aufbau und die Pinbelegung der 32 K x 8 statischen RAMs. Es werden 15 Adreß-, 3 Kontroll- und 8 Datenleitungen benötigt*

Die 15 Adreßleitungen werden intern zu Reihen- und Spaltenadreßleitungen dekodiert, so daß alle Speicherzellen zu 8 Bit ausgelesen werden können. Als Kontrolle dienen 3 Signale Chip-Select, Output-Enable und Write-Enable. Chip-Select dient dazu, den Speicherbaustein »einzuschalten«, es ist low-aktiv. Output-Enable wird benötigt,

um die gewünschten Informationen aus dem statischen RAM auslesen zu können, hierzu muß dieser Anschluß low-aktiv sein. Soll eine oder mehrere Speicherzellen beschrieben werden, so muß der Anschluß Write-Enable low-aktiv sein.

Möchten wir eine 512 Kbyte statische RAM-Aufrüstung entwickeln, so werden 8 RAM-Chips für die oberen 256 Kbyte und 8 weitere für die unteren 256 Kbyte benötigt. Es werden dann insgesamt 16 RAM- Chips, sowie 16 Chip-Select-, 16 Output-Enable- und 16 Write-Enable-Leitungen benötigt. Die Chip-Select-Leitungen ergeben sich aus einer Dekodierung der Adreßleitungen, damit die RAM-Chips bei einer bestimmten Adresse angesprochen werden. Output- und Write-Enable erhält man, indem man die Chip-Select-Leitungen mit dem R/W-Signal des Mikroprozessors verknüpft. Die benötigten Adreßleitungen können direkt an die Adreßleitungsanschlüsse des RAM-ICs angeschlossen und von Chip zu Chip durchgeschleift werden. Die Datenleitungen müssen hingegen zunächst auf einen Bustreiber geführt werden, der die Richtung der Daten steuert, womit ein möglicher Busfehler (zusammentreffen mehrer Daten von verschiedenen Baugruppen) vermieden wird. Nach diesem Bustreiber können die Datenleitungen von Chip zu Chip durchgeschleift werden. Für unsere Schaltung benötigen wir 2 Bustreiber, je einen für die oberen und einen für die unteren 8 Daten-Bits. Sie werden mit der höchsten benötigten Adreßleitung, dem R/W-, sowie dem LDS- und UDS-Signal gesteuert.

Ausgelegt haben wir unsere Schaltung für den Bereich ab 1 Mbyte. Wem dies nicht recht ist, kann die höchste Adreßleitung in diesem Fall A20, ändern. Die Adressendekodierung der RAM-Chips übernehmen zwei 3-Bit-Binärdekoder mit einer Enable-Schaltung. Einer der Dekoder bestimmt das Chip-Select der oberen RAM-Bank, der andere die der unteren. Gesteuert werden die Dekoder durch /LDS für die untere, /UDS für die obere, sowie den Adreßleitungen A19 und A20.

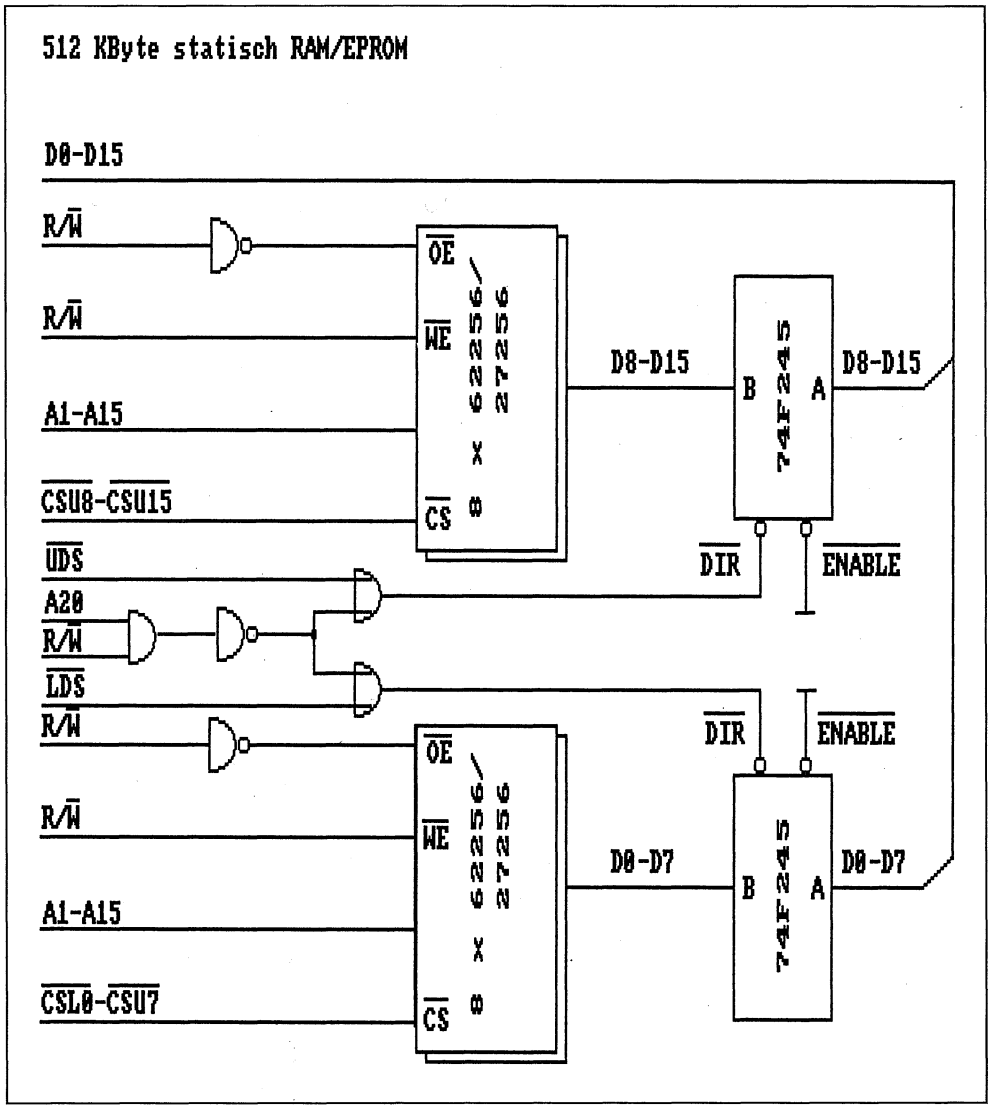


Z 13.2-3: *Die Dekodier-Schaltung für 512 Kbyte statische RAM/EPROM*

Die Ausgänge des Dekoders sind low-aktiv. Der 3 zu 8 Demultiplexer wird durch /LDS für die unteren Daten, /UDS für die oberen Daten, sowie A19 und A20 gesteuert. Dabei enstehen folgende Chip-Select-Signale bei entsprechender Speicheradresse:

Chip-Select	Adresse dezimal	Chip-Select	Adresse dezimal
CS1	1 048 576–1 114 112	CS5	1 310 720–1 376 256
CS2	1 114 112–1 179 648	CS6	1 376 256–1 441 792
CS3	1 179 648–1 245 184	CS7	1 441 792–1 507 328
CS4	1 245 184–1 310 720	CS8	1 507 328–1 572 864

Die so erhaltenen 64 Kbyte Bank-Chip-Select-Signale werden durch /LDS bzw. /UDS nochmals aufgeteilt. Somit stehen 16 Signale zur Verfügung, 8 für die oberen 32 Kbyte-Chips, 8 weitere für die unteren 32 Kbyte-Chips, sie müssen jeweils einzeln an die jeweiligen RAM-ICs geführt werden, so daß jedes IC sein Chip-Select-Signal erhält. Je einmal wird für den RAM-Chip das /OE (output enable) und /WE (write enable) benötigt, es kann also nach einer Aufbereitung durchgeschleift werden. Aufbereitet muß nur das Lesesignal, da hier nur bei Low-Signal eine Umschaltung auf Lesen stattfindet. Hier genügt ein Invertieren des R/W-Signals.

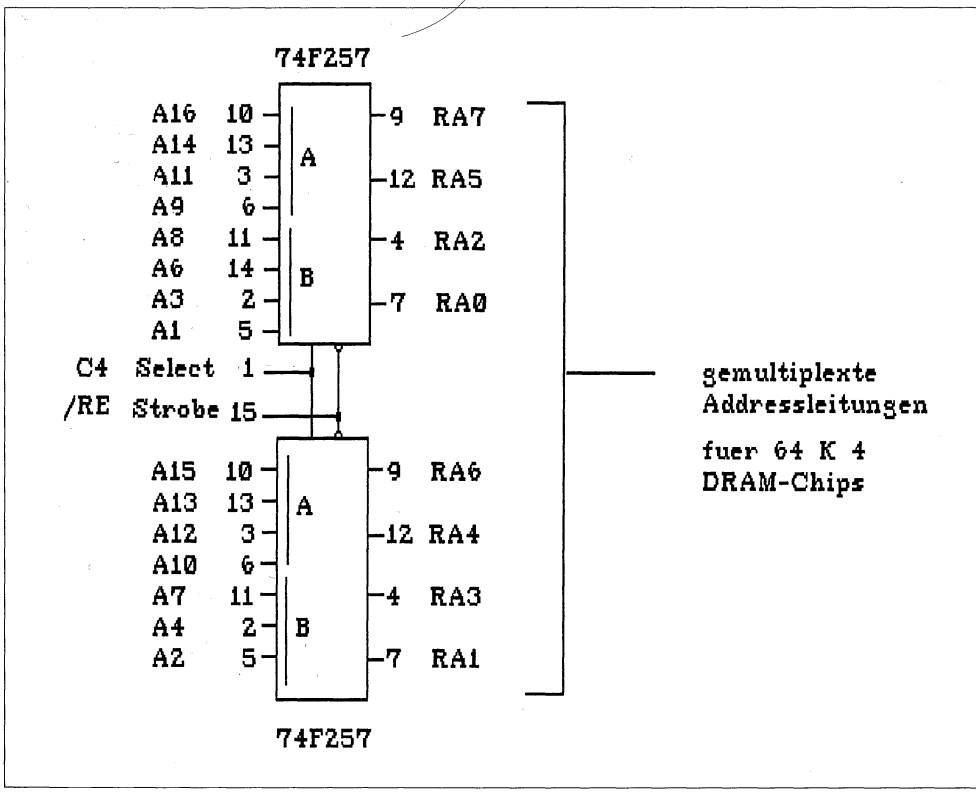


Z 13.2-4: *Der Hauptteil der 512 Kbyte statisch RAM/EPROM Erweiterung*

13.3: Dynamische RAMs am Amiga

Der Bau dynamischer Speicherkarten für den Amiga ist etwas problematischer, da hier die *Refresh*-Signale /CAS und /RAS erzeugt und die Adreßleitungen gemultiplext werden müssen. Der schaltungstechnische Aufwand ist dabei größer als bei statischen RAM-Erweiterungen. Es empfiehlt sich hier, die Ansteuerungs-Logik in ein *PAL* zu brennen. Dies spart Platz und läßt Erweiterung sehr einfach zu.

Die Adreßleitungen werden meistens, bei kleinen RAM-Erweiterungen mit 2 zu 1 Datenselektoren gemultiplext. So auch bei den A1000- und A2000-Modellen. Verwendung finden bei einem Multiplexen der Adreßleitungen des Speicherbereiches $0 bis $1FFFFF zwei 74F157. Zusätzlich besitzt dieser Chip noch eine Steuerschaltung, mit der das Multiplexen gesteuert werden kann. Selektiert wird der Eingang A bei Low-Pegel, bei High-Pegel der Eingang B des Multiplexers. Zum Selektieren wird der /C4-Takt des Systems verwendet, der sich nach der Version, ob PAL oder NTSC, richtet. »Enabelt« wird der Multiplexer mit einem Signal, das sich aus der Abfrage externer Signale, ob DMA-Zugriff oder nicht, und der Speichergrenze, hier $0 bis $1FFFFF, zusammensetzt:



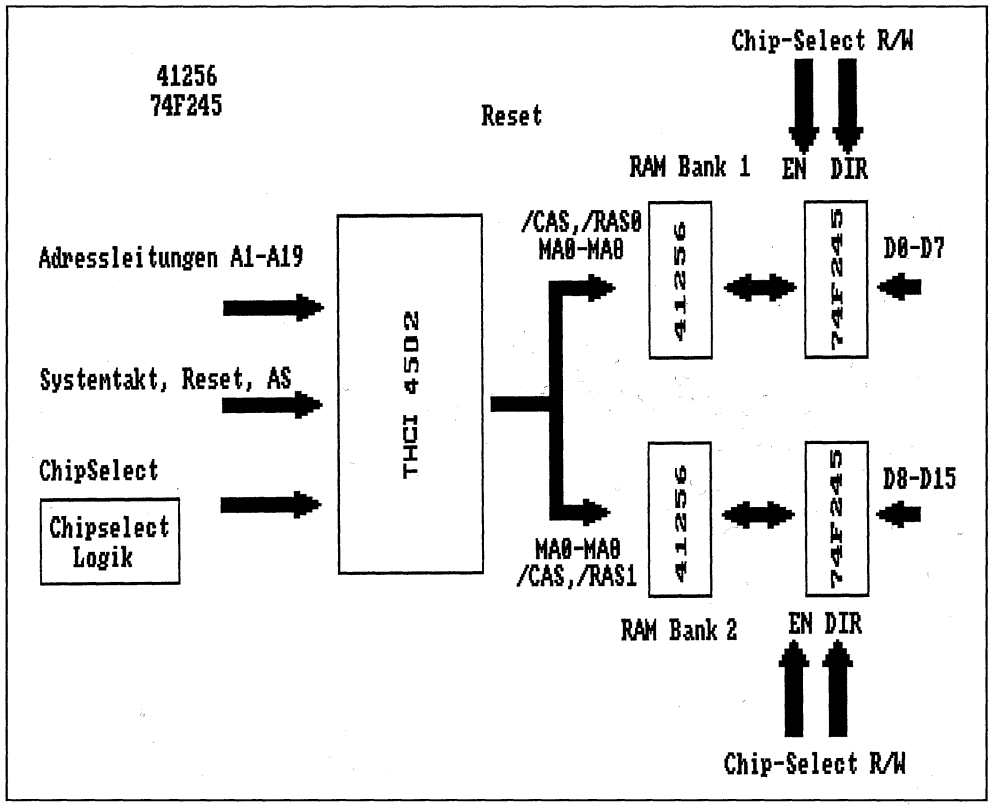
Z 13.3-1: *Das Multiplexen von Daten*

Das Erzeugen der Refresh-Signale /CAS und /RAS ist etwas aufwendiger und von dem jeweiligem RAM-Typ abhängig. Meistens wird so verfahren, daß ein Signal fest durch einen Frequenzteiler erzeugt wird, das andere durch Verknüpfen der möglichen Adresse und verschiedener anderer Signale. Das letztere Signal wird dann in zwei Signale, eines für die oberen Datenbits und eines für die unteren Datenbits aufgeteilt. Somit entstehen zwei RAM-Bänke. Diese Dekodierung und Signalgenerierung beim A1000 und A2000 übernehmen zwei PALs. Das eine PAL, PALEN genannt, übernimmt die Enable-Schaltungen, das andere, PALCAS, ist für die Signal-Generierung zuständig. Beim A500 und B2000 wird die Signal-Generierung durch die FatAgnus übernommen. Signale zur Erweiterung des RAM-Bereiches sind bei den Amigas, außer dem B2000, der die max. 1 Mbyte schon besitzt, vorhanden. Man benötigt hier nur die RAM-Chips und das Know-how der Verdratung. Eine weitere, recht einfache, aber nicht billige Lösung, ist die Verwendung eines DRAM-Controller-Chips.

13.3.1: Mehr DRAM per Kontroll-Chip

Relativ einfach und ohne viel Aufwand sind RAM-Erweiterungen mit RAM-Kontroll-Chips zu bauen. Der RAM-Kontroller, z.B. der THCT4502 für 512 Kbyte, erledigt hierbei alle wichtigen Aufgaben, die für die Generierung der Refresh-Signale und zum Multiplexen der Adreßleitungen benötigt werden.

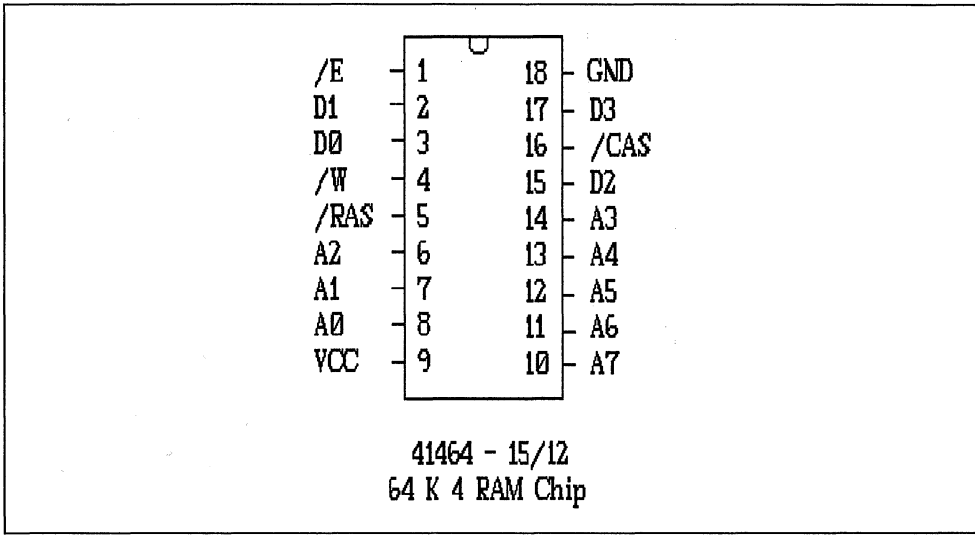
Eingeschaltet wird dieser Kontroller durch das höchste Adreßbit, welches die Startadresse des RAM-Bereichs bestimmt. Extern muß eine Steuerung zum Schalten der *Datenbus-Treiber* in eine obere und untere RAM-Bank hinzugefügt werden. Weiterhin wird ein DTACK-Generator benötigt, der dem MC68000-Prozessor mitteilt, daß die Daten auf dem Bus verfügbar sind, bzw. von dem RAM-Controller assimiliert wurden. Für den Bau einer DRAM-Aufrüstung ist jedoch nach unserer Meinung ein diskreter Aufbau mit Steuerungs-PAL lohnenswerter und flexibler, denn ein solcher *DRAM-Kontroller* ist ersteinmal sehr schwer in einem Elektronik-Laden zu erhalten, und eine aufgebaute Platine benötigt sehr viel Platz.



Z 13.3.1-1: *Der Aufbau einer DRAM-Erweiterung mit dem THCT4502 im Blockdiagramm*

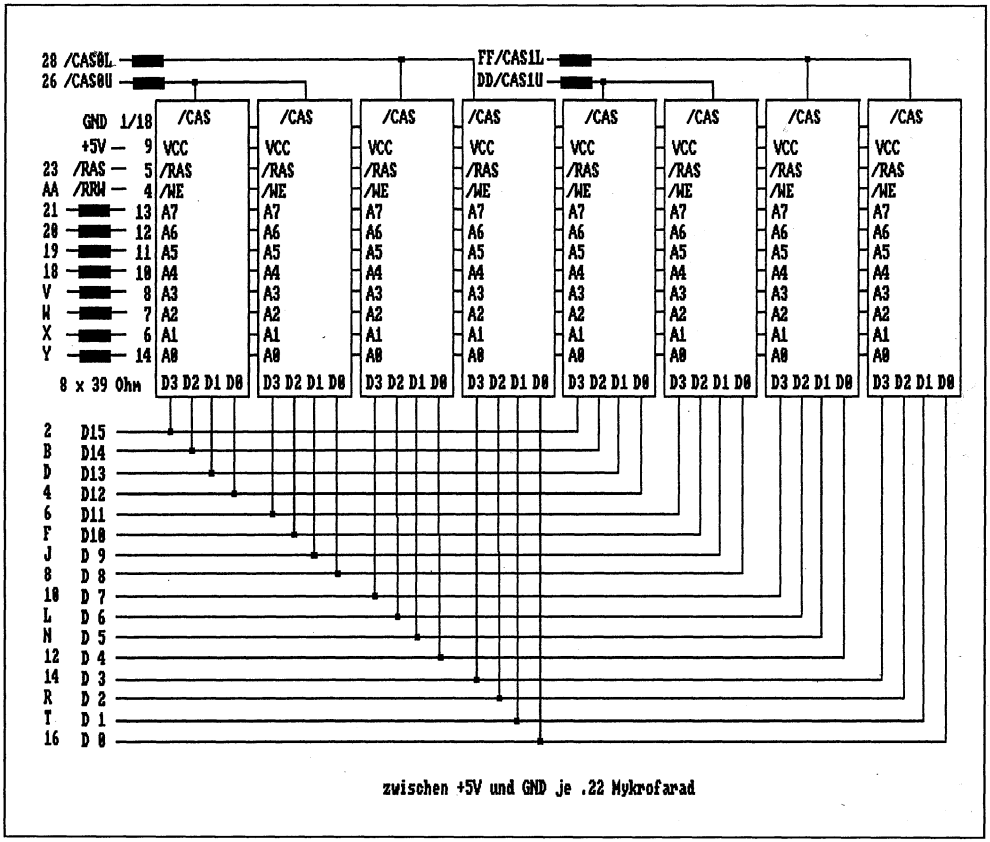
13.3.2: Die 256-Kbyte-RAM-Erweiterung des A1000

Im Gegensatz zu größeren RAM-Erweiterungen, ab dem Bereich von 1 Mbyte, sind Erweiterungen im unteren Bereich leicht zu entwickeln. So wird der Amiga 1000 in einer Grundausstattung von 256 Kbyte geliefert. Er ist intern maximal bis auf 1 Mbyte erweiterbar. Durch ein RAM-Modul ist er sehr einfach um weitere 256 Kbyte erweiterbar, mit dem erst der Betrieb des SideCar und auch ein komfortables Arbeiten möglich wird. Alle benötigten Signale, gemultiplexte Adreßleitungen, Refresh-Signale, sind an den Modul-Port herausgeführt, wodurch der Selbstbau eines solchen RAM-Moduls auch durch einen Laien sehr einfach nachvollzogen werden kann. Benötigt werden hierzu acht 64 K 4 DRAM-Chips mit einer max. Zugriffszeit von 150 ns, sowie vierzehn 39 Ohm Widerstände. Abbildung Z 13.3.2-1 zeigt die Pinbelegung eines 64 K 4 DRAM-Chip.



Z 13.3.2-1: *Die Pinbelegung eines 64 K 4 DRAM-Chip*

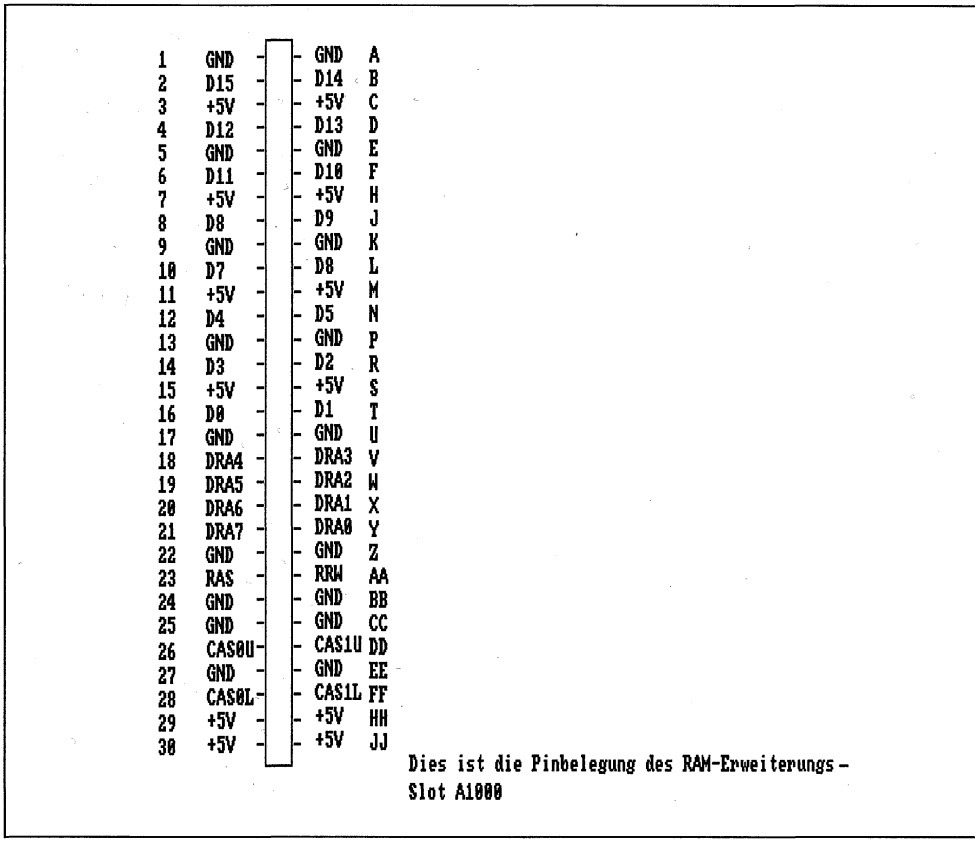
Die Widerstände werden in Reihe geschaltet und dienen so als Strombegrenzer für die Refresh-, Steuer-, und gemultiplexten Adreßsignale. Die gemultiplexten Adreßsignale werden nach den Widerständen einfach von RAM zu RAM durchgeschleift. Ebenfalls geschieht dies mit dem /RAS-, Lese- und Schreibsignal. Die /CAS-Signale hingegen werden auf die obere und untere RAM-Bank verteilt. Zwischen dem +5V- und dem Masse-Anschluß des DRAM-Chips sollte jeweils noch ein 22 F Kondensator liegen. Er dient zum Abfangen von Spannungsimpulsen bzw. Spannungseinbrüchen. Nun braucht nur noch die Schaltung zusammengelötet zu werden. Der RAM-Modul-Stecker wird nach dem Entfernen einer Abdeckung in der Mitte der Frontplatte sichtbar. Fertig ist der 1/2-Meg-Amiga (siehe auch Bild 15 im Farbteil).



Z 13.3.2-2: *Das Schema einer 256-Kbyte-Erweiterung für den RAM-Modul-Port des Amiga (Teil 1)*

13.3.3: 1-Meg-Amiga 1000

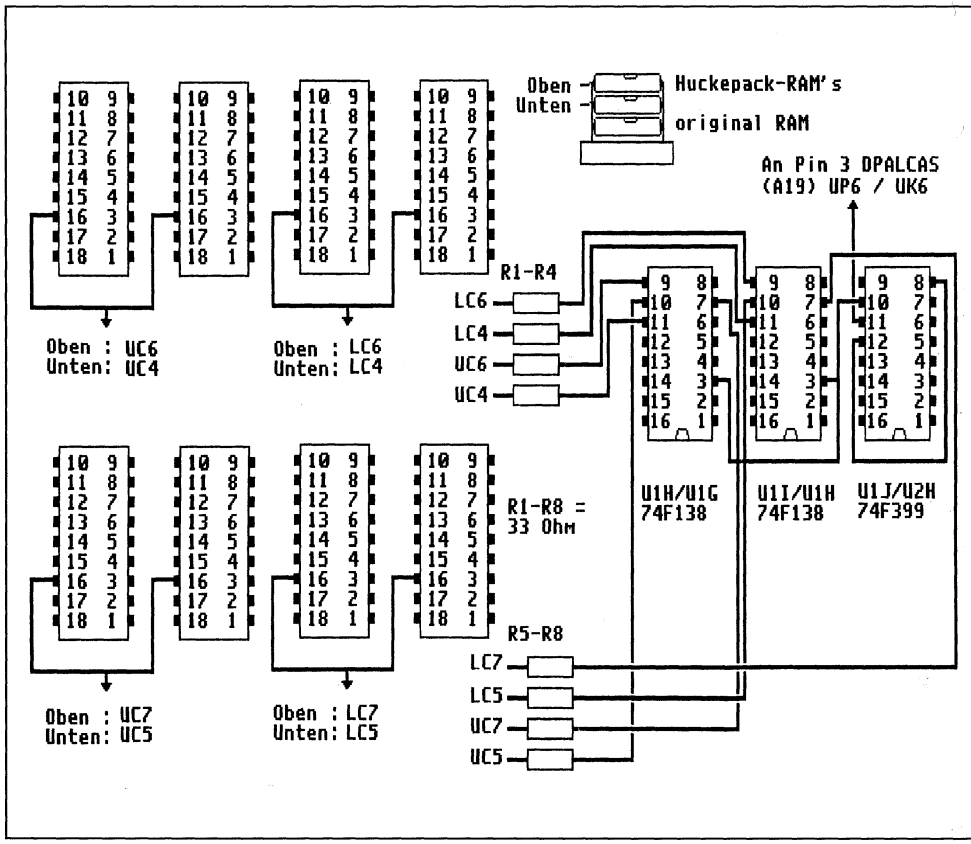
Der Amiga 1000 besitzt die Möglichkeit, sehr einfach und ohne großen technischem Aufwand, das Grundgerät von 512 Kbyte maximalem Ausbau (256 Kbyte intern, 256 Kbyte RAM-Modul) nochmals um 512 Kbyte zu erweitern. Hierzu sind ein paar kleine technische Tricks notwendig. Ein direkter Eingriff in das Gerät läßt sich dabei leider nicht vermeiden. Dies bedeutet natürlich, daß alle Garantieansprüche erlöschen. Dieser Eingriff sollte auch nur von Erfahrenen durchgeführt werden, die sich Ihrer Sache sicher sind und genau wissen, was sie Ihrer Freundin antun. Spätestens dann, wenn bei einem Guru ein Systemabsturz passiert, sollte die RAM-Aufrüstung entfernt und nochmals neu aufgebaut werden. Benötigt für die Aufrüstung werden sechszehn 64 K 4-DRAM-Chips mit einer Zugriffszeit von höchstens 150 ns und acht 33-Ohm-Widerstände, sowie möglichst eine ruhige Hand. Vor dem Eingriff sollte das Netzkabel entfernt werden. Nach dem Abschrauben der Abdeckungen und dem Entfernen des



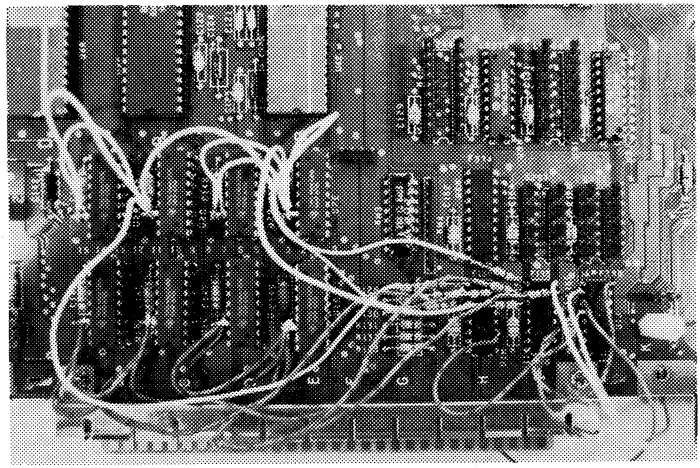
Z 13.3.2-2: *Das Schema einer 256-Kbyte-Erweiterung für den RAM-Modul-Port des Amiga (Teil 2)*

Piggy-Pack (bei älteren Modellen) sind unterhalb der Custom-Chips die RAM-Chips mit der Beschriftung 41464-15 sichtbar. Auf diese RAM-Chips werden je zwei RAM-Chips im Huckepack-Verfahren aufgelötet. Dabei muß der Pin 16 /CAS abgebogen werden. Zudem müssen noch 3 Chips in ihrer Beschaltung modifiziert werden, damit der Amiga die erweiterten RAM-Chips mit einem entsprechendem /CAS-Signal versorgen kann. Hierbei handelt es sich um die ICs mit der Kennung, bei neueren A1000er U1G, U1H und U2H, bei Amigas mit Piggy-Pack U1H, U1I und U1J. U2H bzw. U1J ist ein vier 2 zu 1 Datenselektor 74F399, welcher die Aufgabe hat, die höchsten Adreß-Bits der RAM-Chips, die für das /CAS-Signal benötigt werden, zwischen Custom-Chip und CPU zu multiplexen. Dieses IC muß mit einer weiteren Adreßleitung, A19, beschaltet werden. Die Adreßleitung A19 kann von dem PAL DPALCAS an Pin 3, welches sich auf dem Piggy-Pack oder am Laufwerk hinten links bei neueren Modellen befindet, abgegriffen werden. Diese Adreßleitung wird an Pin 11 des 74F399 gelötet.

Zuvor muß dieser Pin 11 und der benachbarte Pin 10 von dem +5V Anschluß abgetrennt werden, damit der Pegel dieses Signals nicht immer auf +5V gehalten wird. Da die Custom-Chips nur die unteren 512 Kbyte ansprechen können, muß der Pin 12 des 74F399 auf Masse gelegt werden. An Pin 10 dieses ICs liegt nach diesem Umbau die gemultiplexte Adreßleitung MA19 an, die von 74F138 weiterverwendet wird (MA 19 erhalten Sie von Pin 3 des ICs DPAL CAS UP6 bzw. UK6). Diese 74F138, auf der Platine mit U1G und U1H bzw. U1H und U1I bezeichnet, sind zwei 3 Bit Binärdekoder, die hier zur entgültigen /CAS-Generierung benötigt werden. Bei diesen ICs muß der Pin 3 von Masse abgetrennt werden, da hier die gemultiplexte Adreßleitung MA19, die von dem Pin 10 des 74F399 kommt, angeschlossen wird. Durch diese dritte angeschlossene Adreßleitung ergeben sich an den Ausgängen der 74F138 nun 4 weitere /CAS-Signale pro Dekoder-IC. Diese so gewonnenen /CAS-Signale werden jeweils über einen 33-Ohm-Widerstand an dem abgebogenen Pin 16 von zwei benachbarten huckepack aufgelöteten RAM-ICs angelötet. Die Abbildung Z 13.3.3-1 zeigt den Umbau nochmals in bildlicher Form.



Z 13.3.3-1: *Aus einem 512 K-AmigaMIGA wird der 1-Mega-Amiga*



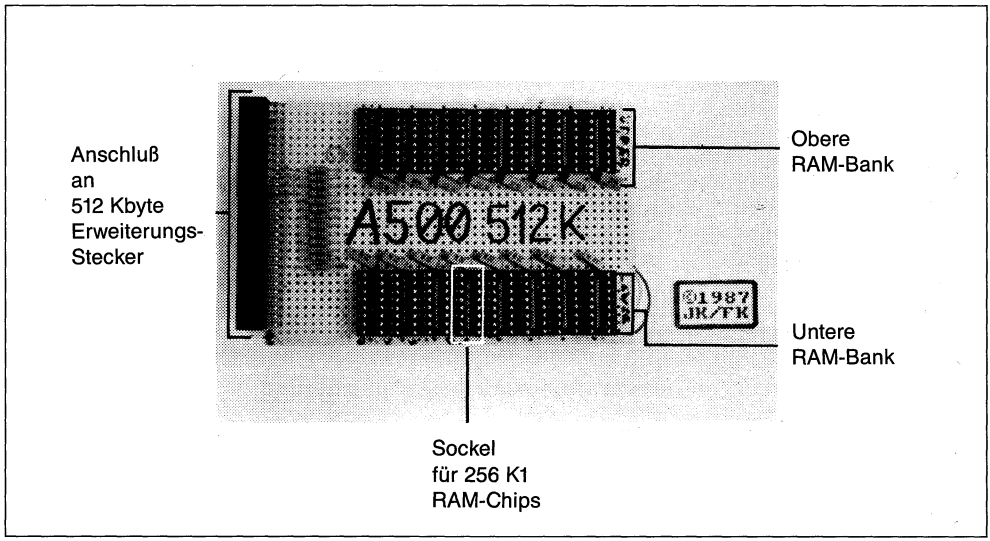
F13.3.3-1: *So sollte der Umbau zum Mega-Amiga ungefähr aussehen*

Abgeschaltet werden kann diese RAM-Aufrüstung durch Einlöten eines Wechselschalters an Pin 3 einer der Dekoder 74F138. Wird dieser Pin auf Masse gelegt, ist die RAM-Aufrüstung nicht aktiv. Dabei muß aber die gemultiplexte Adreßleitung MA19 vom 74F399 abgetrennt werden. Ist der Einbau vollzogen, sollte ersteinmal ein Funktionstest vollzogen werden. Besitzer des alten 1000er sollten hierbei nicht das Einsetzen des Piggy-Pack vergessen. Gibt Ihr Rechner beim Booten der Kickstart 1.2 ein Farbspiel von sich, z.B. ein Umschalten der Bildschirmfarbe auf grün, so sollte eine gründliche Kontrolle der eingebauten RAM-Aufrüstung stattfinden. Fährt der Rechner normal hoch, so kann der zusätzliche Speicher mit dem Befehl »AddMem $080000,$100000« eingebunden werden. Durch Veränderung der Adreßleitung A19 kann die Startadresse der RAM-Aufrüstung verändert werden. A20 bestimmt z.B. den Start ab $100000 bis $180000. Diese Einbindung des Speichers hat jedoch den Nachteil, daß Programme, die ihre Daten im Speichertyp Chip-Mem benötigen (untere 512 Kbyte, auf die die Custom-Chips nur zugreifen können), nicht 100prozentig laufen, da bei nicht genauer Deklarierung des Speichertyps die Daten im Fast-RAM (RAM-Speicher über 512 Kbyte) abgelegt werden, welches meistens zum Absturz des Rechners führt.

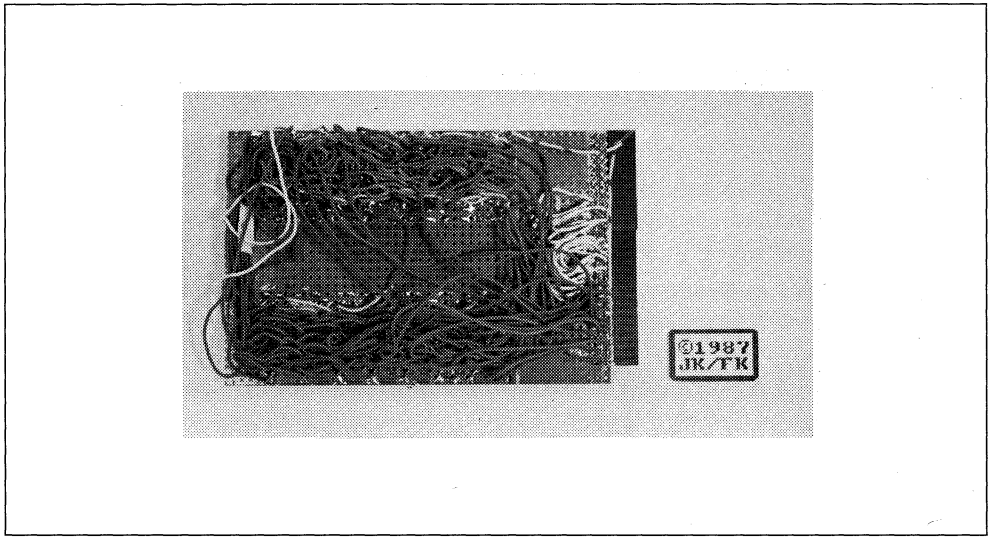
13.3.4: Die 512-Kbyte-RAM-Erweiterung des Amiga 500

Weniger problematisch ist die Aufüstung des Amiga 500 auf 1 Mbyte. Er besitzt einen kleinen RAM-Erweiterungsport, der von unten, nach dem Entfernen einer kleinen Klappe sichtbar ist. Hier sind ebenfalls alle wichtigen Signale, die für eine RAM-Aufrüstung benötigt werden, herausgeführt. Es werden nur 16 RAM-Chips benötigt. Die Entwicklung einer Steckkarte ist hier ebenso möglich, wie das bekannte Huckepack-

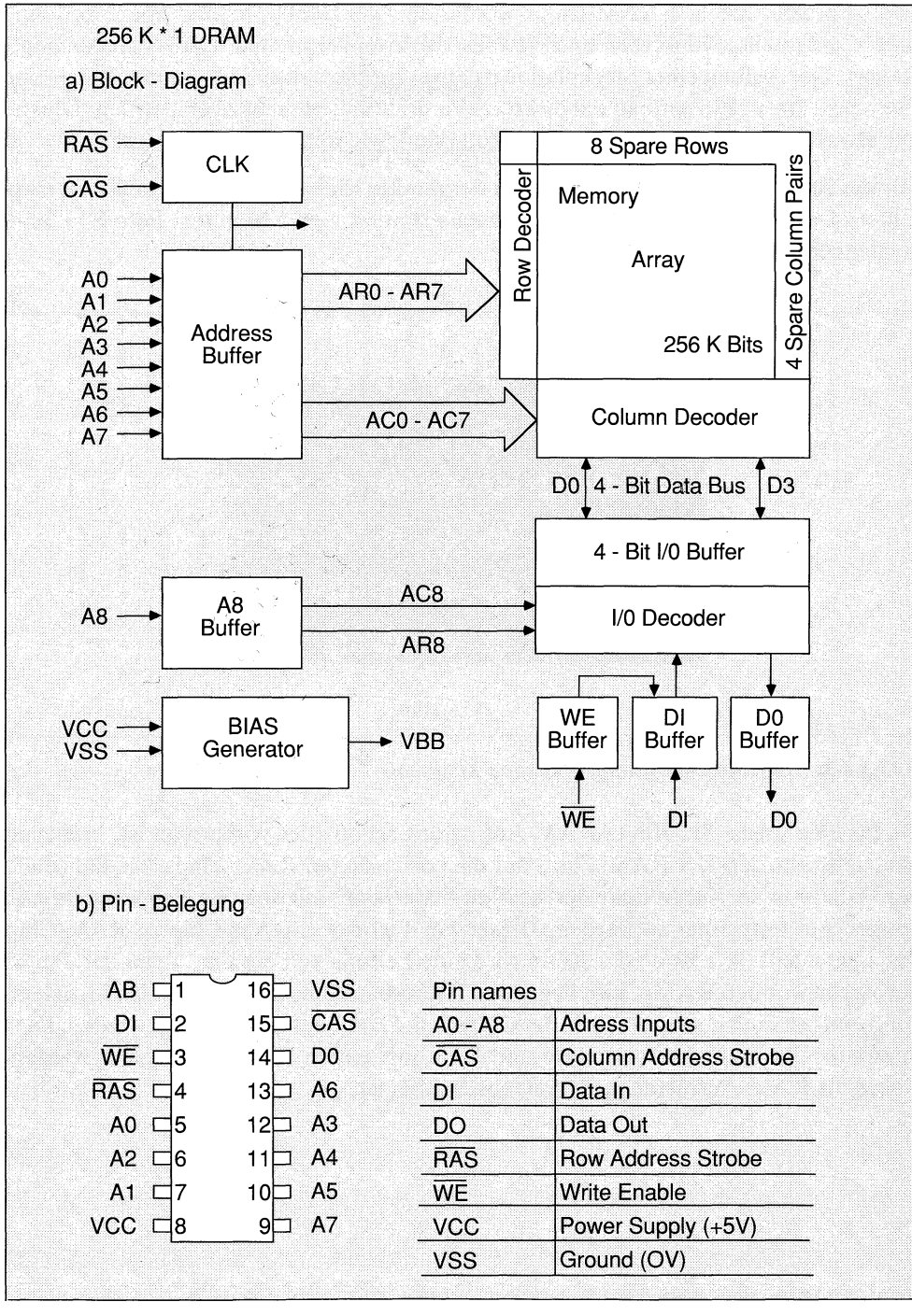
Verfahren der internen RAM-Aufrüstung beim A1000. Im Gegensatz zum 1000er werden beim 500er 256-K 1-RAM-Chips verwendet. Dadurch wurde auch die ganze Signal- Generierung für die RAMs umgestrickt, die PAL's vom 1000er wurden überflüssig und durch FatAgnus und Garry ersetzt.



F 13.3.4-1: *Von oben gesehen: ein normaler Aufbau einer Testplatine*



F 13.3.4.2: *Wir werden uns doch wohl nicht verlötet haben?*

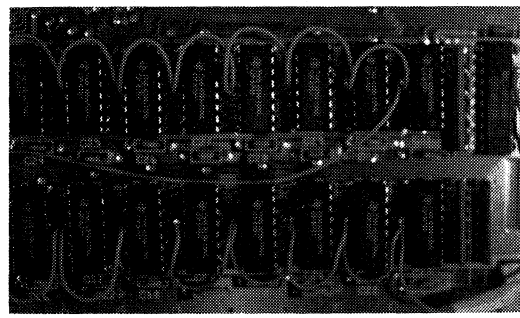


Pin names	
A0 - A8	Adress Inputs
CAS	Column Address Strobe
DI	Data In
DO	Data Out
RAS	Row Address Strobe
WE	Write Enable
VCC	Power Supply (+5V)
VSS	Ground (OV)

Z 13.3.4-1: *Das Blockdiagramm und die Pinbelegung eines 256-K 1-RAM-Chips*

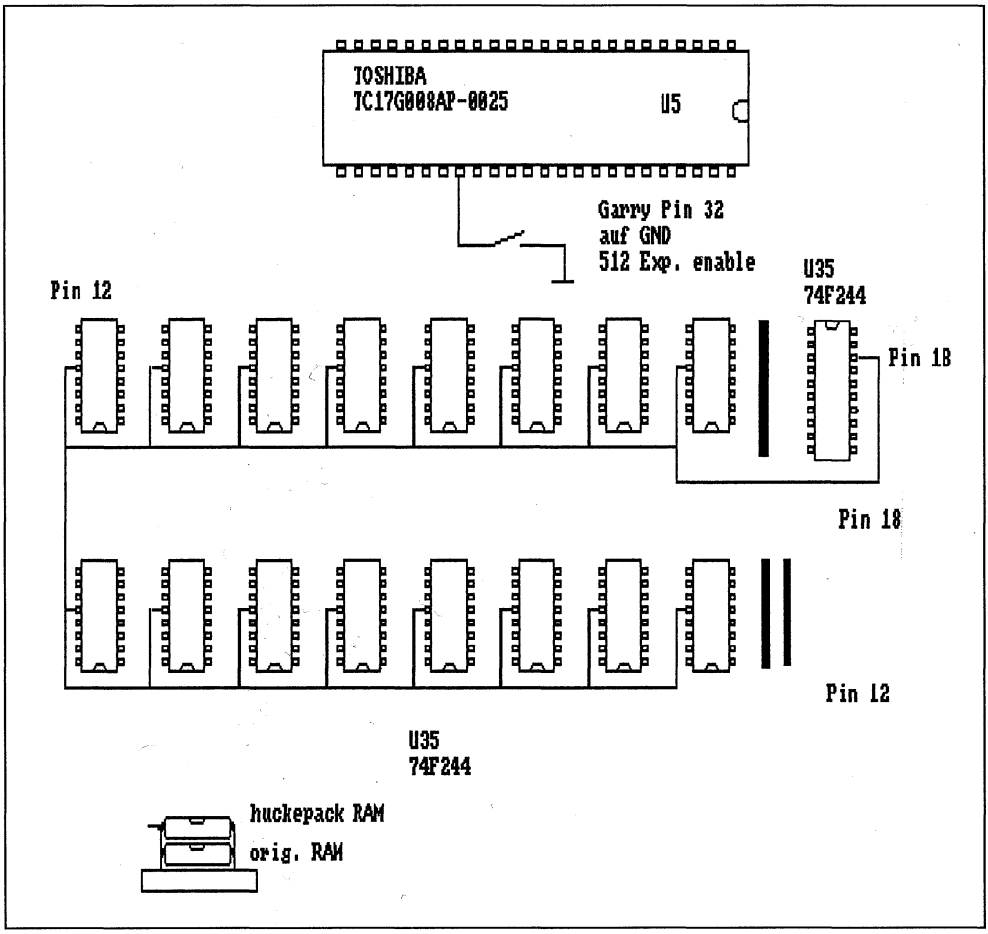
Die sechszehn 256-K 1-RAM-Chips werden in zwei Bänke, je eine obere und eine untere aufgeteilt. Sie werden auch jeweils von zwei getrennten /CAS-Signalen angesteuert. Der Aufbau einer Steckplatine ist etwas problematisch. Es empfiehlt sich eine Platine zu ätzen. Ein Aufbau mit diskreter Verdrahtung ist nicht zu empfehlen. Dies ist nur etwas für Freaks, die total von der Rolle sind, wie folgende Fotos zeigen:

Für die Schnellen unter den Bastlern ist der direkte Einbau in den Amiga 500 zu empfehlen. Der Eingriff ist sehr einfach und dauert nur knapp 5 Minuten. Foto F 13.3.4-3 zeigt einen solchen Umbau.



F 13.3.4-3: *512-Kbyte-Aufrüstung in knapp 5 Minuten*

Da für eine solche 512-Kbyte-RAM-Aufrüstung schon alles vorbereitet ist, brauchen nur sechszehn 256-K 1-RAM-Chips auf die vorhandenen RAM-Chips, die unterhalb der Tastatur in der linken Ecke des A500 zu finden sind, mit dem Pin 4 abgebogen, aufgesetzt und festgelötet zu werden. Dieser Pin 4 wird von RAM-Chip zu RAM-Chip durchgeschleift. Am Ende der Kette wird eine Leitung von diesem Pin an den Pin 18 des, rechts vor den RAM-Chips liegenden Treiberbausteins U35, einen 74F255, gelegt. Zu guterletzt muß noch der Pin 32 des Chips U 5, Garry, auf Masse gelegt werden. Dies macht die RAM-Aufrüstung aktiv und kann mit einem Schalter versehen werden, womit die RAM-Aufrüstung jederzeit abschaltbar ist.



Z 13.3.4-2: *Der schematische Aufbau der 512-Kbyte-RAM-Aufrüstung*

Kapitel 14

Die Monitore des Amiga

Neben dem *Monitor* der ersten Stunden, dem A 1081, sind inzwischen weitere Monitore, wie der A1084 und der A 2024 auf dem Markt. Während der A 2024 ein Monochrom-Monitor ist, zählen die anderen beiden zu der Klasse der Spitzen-Farbmonitore.

Die Farbmonitore sind für alle Computer der Commodore-Serie geeignet. Sie lassen sich sowohl an einen PC, als auch an einen Amiga anschließen. Ebenfalls lassen sie sich als TV-Monitor oder als Sichtgerät für Videorekorder einsetzen.

In der Praxis überzeugen sie durch ihre hervorragende Farbwiedergabe und die gestochen scharfe Zeilendarstellung. Der Bildschirm hat eine Diagonale von 14 Zoll, die Video-Bandbreite liegt bei einem FBAS-Signal bei 4.5 MHz, bei RGB um die 12 MHz. Im Vergleich zum A 1081 ist der A 1084 entspiegelt. Die Auflösung der Monitore beträgt 600 Zeilen in der Bildmitte, bei einer *Bildfrequenz* von 50/60 Hz und einer *Zeilenfrequenz* von 15.625 Hz. Zudem besitzen die Monitore verschiedene Regler zum Einstellen eines optimalen Bildes. Auf der Vorderseite befinden sich Regler für Helligkeit, Kontrast, Farbsättigung und Bildschärfe, sowie ein Umschalter für RGB-linear bzw. Videoeingang. Auf der Rückseite sind Regler für vertikale Höhe, horizontale Weite und vertikale Mitteneinstelllung, sowie ein Umschalter für den Betrieb mit einem Videorekorder vorhanden. Der Audio-Ausgang der Monitore hat eine Ausgangsleistung von 1 Watt, was durchaus ausreichend ist. Auf der Rückseite befinden sich alle Anschlüsse, die die Monitore mit der Umwelt verbinden. Ein SCART-Anschluß ist ebenso vorhanden, wie ein Audio- und ein RGB-TTL-Eingang.

Neu auf dem Markt ist der monochrome Monitor A 2024. Dies ist ein hochauflösender Monitor für den Amiga 500, Amiga 1000 und Amiga 2000, soweit sie mit mind. 1 Mbyte RAM ausgestattet sind. Er bietet zwar nur vier Graustufen (schwarz, dunkelgrau, hellgrau und weiß), besitzt dafür aber eine Auflösung auf dem 15 Zoll-Bildschirm von 704 x 256, 704 x 512 und 1008 x 1024 Punkte, die nur im Zusammenhang mit einer speziellen Workbench erreichbar ist.

Ganz neu auf dem Markt ist noch der Farbmonitor 2040, der ein hervorragendes Bild auch im *Interlace*-Modus liefert. Leider stehen dazu noch keine detaillierten Informationen zur Verfügung.

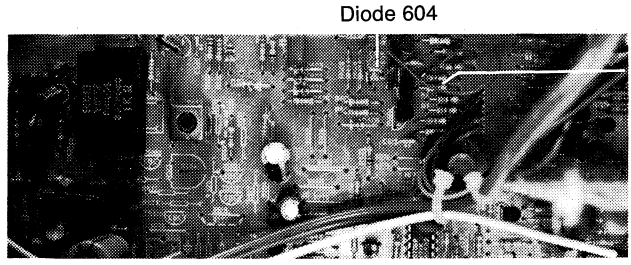
14.1: Verbesserungsmöglichkeiten des A1081/A1084

Verbesserungen, die auch wirklich durchführbar sind, sind bei einem solchen Monitor nicht leicht zu finden. Eine lohnenswerte Verbesserung wäre sicher, den Monitor 1081 zum Stereo-Monitor umzubauen, oder die, beim Umschalten von RGB- zu FBAS-Betrieb auftretenden Störungen abzustellen. Diese Änderungen, so meinen wir, sind meistens überflüssig, da ein Stereo-Sound-Betrieb über die 200 Watt Stereoanlage lohnenswerter ist und das Umschalten von RGB-Amiga auf »FBAS-Ronnys Pop Show« durch unser low-cost Genlock Interface, das wir später vorführen, überflüssig wird. Interessanter ist es, beim Umschalten auf die Gründarstellung mal blau zu sehen.

14.1.1: Ein Grün-Monitor sieht Blau

Mit unserer kleinen Schaltung kann man ganz leicht bei einem A 1081-Monitor den Grünbetrieb auf Blaubetrieb ändern. Bei dieser nun folgenden Umbauanleitung sollte man sehr sorgfältig umgehen, denn ein paar KV an der Bildröhre werfen sogar den stärksten Hacker um.

Der Umbau ist ganz einfach, wenn man von der oben erwähnten Gefahr absieht. Drei Transistoren vor der RGB-Endstufe bereiten das RGB-Signal des Decoders TDA3505 auf. Es handelt sich hierbei um die Transistoren TS606 bis TS604 (für die Farben Blau, Grün, Rot). Am Emiter der Transistoren TS606 (Farbe Blau) und TS604 (Farbe Rot) ist unter anderem eine Diode D604 (Farbe Blau) und D605 (Farbe Rot) angeschlossen. Diese Dioden leiten beim Einschalten des Grün-Schalters des A 1081-Monitors ca. +12V auf den Emiter des jeweiligen Transistors, der deshalb nicht mehr durchschalten kann und somit sein Ausgangswert gleich null ist, d.h. der Blau- und der Rot-Wert werden herausgefiltert. Der Transistor für Grün hingegen kann noch durchsteuern. Es wird so nur die Farbe Grün angezeigt. Wird nun die Diode D604 (Farbe Blau) von dem Emiter-Anschluß des Transistors TS606 getrennt und auf den Emiter des Transistors TS605 gelegt, so entsteht beim Umschalten von RGB auf Grün-Betrieb diesmal kein grünes, sondern ein blaues Bild, da nur der Transistor für Blau durchsteuern kann. Der Emiter des Transistors für die Farbe Grün liegt ebenfalls am Widerstand R612 und R615. Foto F 14.1.1-1 zeigt diesen Umbau.



F 14.1.1: *Aus Grün wird Blau*

Selbstverständlich kann man auch auf Rot-Betrieb umstellen, oder so, daß nur der Grün-, Blau- oder Rot-Wert ausgefiltert wird, das überlassen wir ganz Ihnen.

Kapitel 15

PAL für den Amiga

Amiga-Besitzer, die ihren Rechner gerne an einen Fernseher anschließen möchten, stehen vorerst vor einem großen Problem. Bei alten Amiga 1000ern kann zum einen das Farbbild-Austast- und Synchronsignal *(FBAS*-Signal) in NTSC-Norm vorliegen, bei den A2000ern ist die Encoder-Logik für dieses Signal komplett entfallen und bei den 500ern und B2000ern ist diese Logik durch einen Video-Hybrid-Baustein ersetzt worden, weshalb kein Farbsignal vorhanden ist. Besitzt jedoch der Farbfernseher eine Scart-Buchse, so enfällt das Problem eines Modulators. Der RGB-Port kann direkt an einen solchen Fernseher angeschlossen werden. Hierbei sollte aber bei einer Verbindung die Scart-Stecker-Belegung des Fernsehers und des AMIGA-RGB-Ports genau verglichen werden. Das so erhaltene Bild vom AMIGA ist fast vergleichbar mit dem des *RGB*-Monitors.

Ob PAL- oder NTSC-Norm am *Composite-Video*-Anschluß des Amiga 1000 anliegt, ist sehr einfach zu testen. Hierzu muß eine Verbindung zwischen dem Composite-Ausgang des 1000er und dem Composite-Eingang des Amiga 1081-Monitor geschaffen werden. Ist das Bild beim Umschalten auf CVBS schwarz-weiß, ist der Encoder- MC1377 in NTSC-Norm geschaltet. Dies ist zusätzlich daran zu erkennen, daß sich in seiner Nähe ein R57 bzw. R46 befindet (der MC1377 befindet sich in der Nähe des Composite-Video-Anschlusses). Zeigt der Composite-Videoanschluß des Amiga 1000 ein schwarz-weißes Bild, so ist ebenfalls zu prüfen, ob der Custom-Chip AGNUS (der Mittlere von den dreien, gekennzeichnet auf dem Motherboard mit »A«) in PAL-Norm vorliegt, da von ihm wichtige Synchronisationssignale geliefert werden. Ist dieser Chip mit 8361 gekennzeichnet, so liegt er in der *NTSC*-Version vor. Soll der Chip PAL-Signale liefern, so muß er gegen den AGNUS 8367 ausgetauscht werden (PAL-Version) (Farbteil Bild 30).

Nun, warum ist der Unterschied zwischen PAL und NTSC so wichtig. PAL und NTSC sind Verfahren zur Modulation eines Farbträgers mit jeweils drei Farbsignalen. Das NTSC-Vefahren ist schon 1953 von einem Ausschuß von Technikern und Ingenieuren aus der Industrie der USA entwickelt worden (NTSC = National Television System Commitee). Das NTSC-Verfahren zum Farbfernsehempfang wird überwiegend in den USA und Japan eingesetzt. Das PAL-Verfahren (Phase Alternation Line = Zeilenfre-

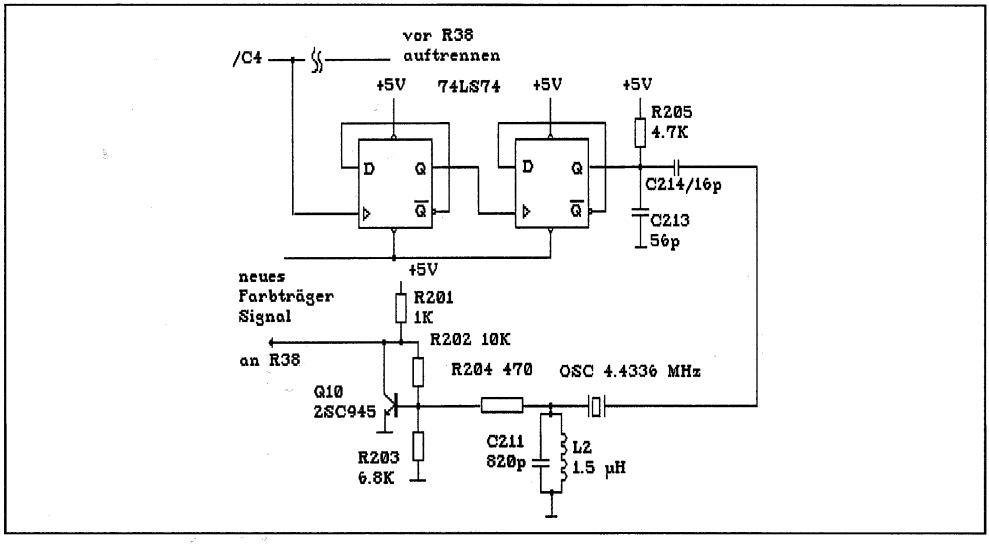
quenter Phasenwechsel) ist eine Weiterentwicklung des NTSC-Verfahren. Es findet überwiegend in den europäischen Ländern Verwendung. Ein wesentlicher Unterschied zwischen dem PAL- und dem NTSC-Verfahren liegt darin, daß bei PAL von Zeile zu Zeile die Modulationsrichtung des Farbsignals von positiver nach negativer Modulation umgeschaltet wird. So können Unterschiede in der Farbinformation durch Bilden eines Mittelwertes ausgeglichen werden, was beim NTSC-Verfahren nicht möglich ist. Trotz dieser Weiterentwicklung kann mit einem PAL-Farbfernsehempfänger ein NTSC-Signal empfangen werden. Das sichtbare Bild enthält aber keine Farb-, sondern nur Grauwerte. Dies liegt an den normbedingten Abstimmfrequenzen. Der Farbträger liegt bei dem PAL-Verfahren auf der Frequenz von 4.43361875 MHz, bei NTSC jedoch auf 4.4296875 MHz, da hier ein Halbzeilen-Offset-Verfahren benutzt wird.

Soll der Amiga 1000 nun von NTSC auf PAL umgerüstet werden, so muß man den RGB-Encoder MC1377, der das Composite-Videosignal liefert, mit einem neuen Farbträger speisen. Beim Amiga 500 ist die Sache etwas schwieriger, da hier der Video-Hybrid-Baustein alle wichtigen Funktionen übernimmt (Hybrid = Mischschaltung aus verschiedenen Technologien). Das Farbträgersignal wird bei dieser Schaltung nicht ausgewertet, so daß das Videosignal nur in schwarz-weiß vorliegt (wem dies genügt, kann auch ein Schwarz-weiß-Signal modulieren). Möchte man ein FBAS-Signal in PAL-Norm erhalten, so muß eine komplette Encoder-Schaltung des MC1377 an den RGB-Port angefügt werden. Beim Amiga A2000 fehlt die ganze Logik zum Anschluß eines normalen Monitors. Hier muß die komplette Encoder-Schaltung in einer Steckkarte für den Video-Slot aufgebaut werden. Beim B2000 ist wie beim A500 zu verfahren. (vergleiche Bild 29 im Farbteil).

Ist der PAL-Composite-Videoausgang vorhanden, muß dieser noch aufbereitet werden, wenn man den Amiga an einen Farbfernseher anschließen möchte. Damit ist gemeint, daß dieser Ausgang mit einem Trägersignal Amplituden-moduliert wird.

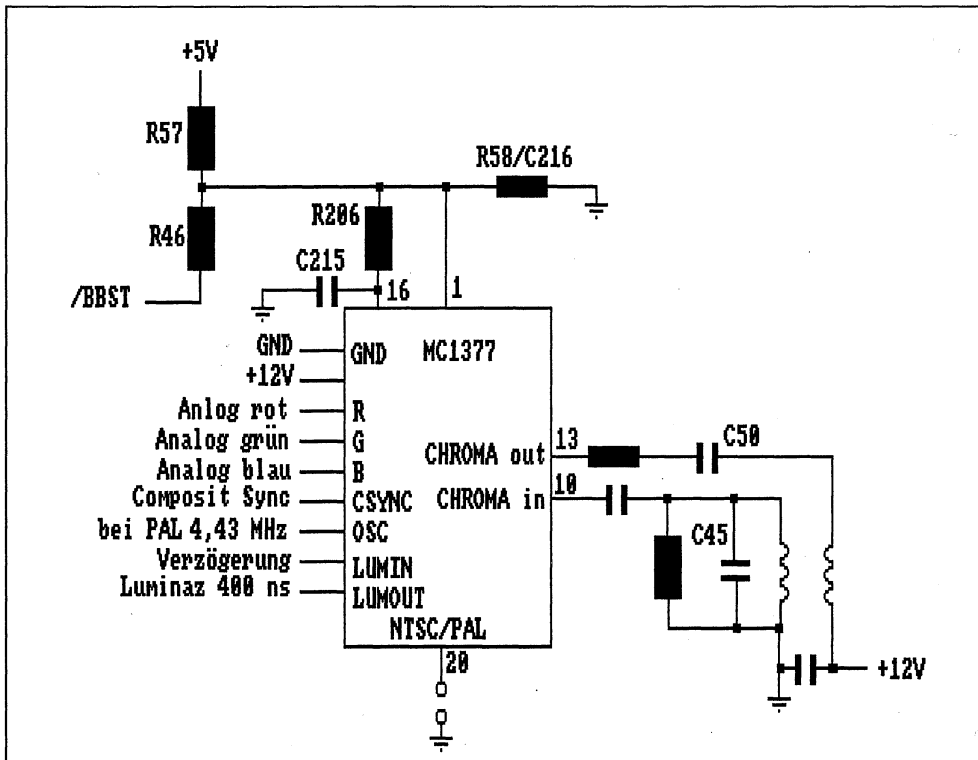
15.1: Amiga 1000 – Aus NTSC wird PAL

Amiga 1000-Rechner, die seit Ende '86 ausgeliefert werden, enthalten den kompletten Umbau für einen PAL-Amiga. Hier ist sowohl der RGB-Encoder, als auch AGNUS der PAL-Norm angepaßt. 1000er, die noch Mitte '86 erhältlich waren, enthielten nur die PAL-Agnus. Möchte man jedoch den Composite-Videoanschluß auf PAL-Norm umstricken, müssen einige Bauteile, die die Beschaltung des RGB-Encoders betreffen, umgelötet bzw. weggelassen werden. Zudem muß man eine Schaltungserweiterung für das Farbsignal hinzufügen. Hierzu muß an R38 die Leitung zum Taktgenerator aufgetrennt werden (der andere Anschluß geht an C51). Die Taktschaltung von 4.43 MHz wird über 2 Daten-FlipFlops mit Preset und Clear (sie dienen als Frequenzteiler) »Enable« geschaltet. Der anschließende Transistor Q10 dient als Schalter. Abbildung Z 15.1-1 zeigt den Aufbau der Taktschaltung.



Z 15.1-1: *Ein 4.433 MHz-Quarz sorgt für das bei PAL benötigte Farbsignal*

Ist dieser Umbau vollzogen, braucht man nur noch 7 Bauteile, die den RGB-Encoder MC1377 beschalten, umzulöten. Sehr wichtig bei diesem Encoder ist Pin 20. Hier wird »eingestellt«, ob ein Composite-PAL- oder NTSC-Signal produziert werden soll. Abbildung Z 15.1-2 zeigt den kompletten Umbau.



	R46	R57	R58/C216	C45	C50	R206	C215
NTSC	1.69K	3.92K	3.92K	47pF	100pF	----	----
PAL	----	----	.001µF	33pF	56pF	56.2K	.1µF

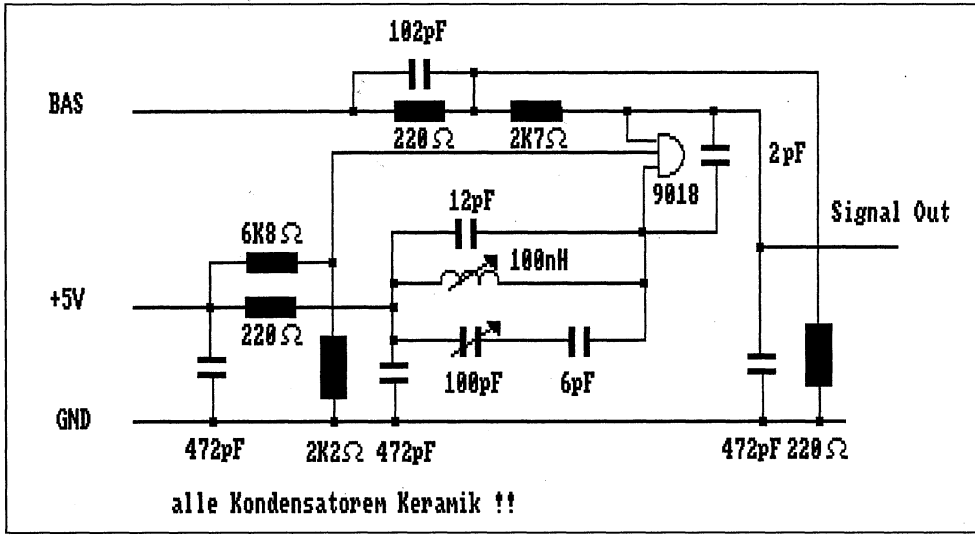
Pin 20 offen = NTSC GND = PAL

Z 15.1-2: *Aus NTSC wird PAL. Ca. 7 Bauelemente müssen umgelötet werden. Dann bekennt der Composite-Video-Ausgang PAL- Farbe*

Auf richtige Funktion kann man die Beschaltung mit dem Composite-Eingang des Amiga-Monitors 1081 testen. Zeigt der Amiga beim Umschalten auf *CVBS* Farbe, so verlief die Operation einwandfrei. Andernfalls Schaltung nochmals überprüfen.

15.2: PAL-Modulator für den Amiga

Ein PAL-Modulator hat die Aufgabe, ein FBAS/BAS-Signal auf eine gewisse Frequenz zu modulieren. Üblicherweise wird hier der AV-Kanal verwendet, der auch für Videorekorder benutzt wird. Die Modulation, die hier verwendet wird, wird auch als eine Amplituten-Modulation bezeichnet. Eine Amplituten-Modulation erhält man, indem z.B. ein Transistor mit einem Trägersignal ausgesteuert wird. Die Signalschwingung (Composite-Video) verändert die Basis-Vorspannung. Somit wird auch der Verstärkungsfaktor B des Transistors im Rhythmus der Signalschwingung verändert. Das Resultat ist eine Frequenz, deren Amplitude sich im Takt des Signals verändert. Mit diesem HF-Signal kann nun der Empfänger eines Fernsehers gespeist werden. Die Bildqualität ist aber leider nicht so überragend, da der Amiga mit einer sehr hohen Auflösung arbeitet. Dadurch werden feine Konturen verzerrt wiedergegeben. Die folgende Schaltung stellt einen einfachen Modulator dar. Durch die Spule in dem LC-Schwingkreis kann die Bandbreite des zu empfangenden Kanals eingestellt werden.



Z 15.2-1: *Eine sehr einfache Schaltung zum Modulieren eines BAS/FBAS-Signals*

Kapitel 16

Bastelanregungen

In diesem 16. Kapitel soll der Bastler unter den Amiga-Freaks voll auf seine Kosten kommen. Video- und Sound-Faszination stehen bei unseren ausgewählten Schaltungen im Vordergrund. Eine der interessantesten Schaltungen, die wir für dieses Buch entwikkelt haben, ist das low cost-Genlock-Video-Interface. Im Anfangsstadium der Entwicklungsarbeiten glaubten wir selbst nicht so recht daran, daß die Teile, die wir für ca. 20 DM gekauft hatten, ein relativ gutes Genlock-Video-Interface abgeben würden. Das Ergebnis war von daher erstaunlich. Als wir am Ende der Manuskriptarbeiten zu diesem Buch einen Digitalisierer testeten, kam uns die Idee, aus dem Genlock noch einen Digitizer zu bauen. Leider konnten wir unsere Idee aus Zeitgründen nicht ganz verwirklichen. Oftmals reicht jedoch einem Bastler ein guter Hinweis, um eine große Sache zu verwirklichen.

16.1: Der Amiga-Sound-Digitizer

Den Sound-Digitizer, der in diesem Kapitel vorgestellt wird, ist der Sound-Digitizer aus der ehemaligen 68000er von Markt&Technik. Mit wenigen Bauteilen wird hier eine sehr effektvolle Schaltung verwirklicht. Das Herz der Schaltung ist ein 7574. Dies ist ein 8-Bit-AD-Wandler mit einer min. Wandlungszeit von 15 Mikrosekunden. Diese 15 Mikrosekunden werden extern durch ein RC-Glied bestimmt. R4 und C5 sind maßgebend für die Wandlungszeit und können bei Timingproblemen verändert werden. Der AD-Wandler ist ständig aktiv, da die Chip-Selekt-Leitung direkt auf Masse gelegt ist. Gesteuert wird der Wandlungsvorgang mit dem Read/Write-Signal /RD. Es ist an den Parallel-Port an Pin 13 SEL angeschlossen. Geht SEL nach 0 Volt, so wird die Digitialisierung gestartet und der AD-Wandler signalisiert durch Low-Schalten des Signals BUSY, daß er mit dem Digitialisieren begonnen hat. Ist der Digitialisierungsvorgang beendet, wird BUSY wieder auf +5V gelegt. Dies bedeutet, daß die 8-Bit-Daten an den Datenregistern vorhanden sind. Der Wert kann nun vom Amiga ausgelesen werden. Die Eingangsempfindlichkeit des AD-Wandlers kann mit dem Poti P1 eingestellt werden. Man sollte darauf achten, daß die angeschlossene Spannung am Eingang dieses Sound-Digitizers nie größer als 5 Vpp wird, da sonst die Schaltung zerstört wird. Zum

Anschluß eignet sich z.B. der Line-Out-Ausgang oder die Köpfhörer-Buchse des Kassettenrekorders. Vor dem Eingang des AD-Wandlers befindet sich noch ein Integrierglied, mit dem sich die Dauer der zugeführten Impulse vergrößern läßt. Die –5V werden bei dieser Schaltung durch ein MAX 7660 erzeugt. Beachten sollte man beim Anschluß des Digitizers, ob es sich um einen A1000 oder A500/2000 handelt. Die Leitung von Pin 14 muß beim Anschluß an einen Amiga 500/2000 auf Pin 18 und die Leitung von Pin 23 auf Pin 14 gelegt werden. Nähere Informationen zur parallelen Schnittstelle finden Sie im Kapitel »Die parallele Schnittstelle«.

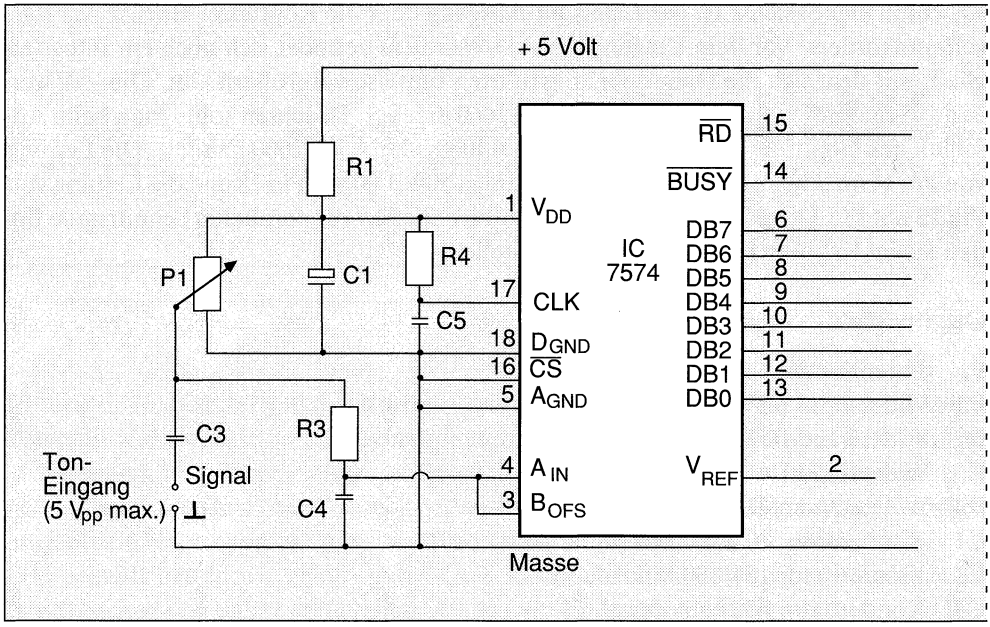
Die Stückliste:

P1 Potentiometer 10 KOhm linear
R1 Widerstand 1/4 Watt 1 Ohm
R2 Widerstand 1/4 Watt 1 Ohm
R3 Widerstand 1/4 Watt 910 Ohm
R4 Widerstand 1/4 Watt 180 KOhm
C1 Kondensator 10 V 100 Mikrofarad
C2 Kondensator 10 V 100 Mikrofarad
C3 Kondensator 680 Nanofarad
C4 Kondensator 22 Nanofarad
C5 Kondensator 120 Picofarad
C6 Kondensator 10 V 10 Mikrofarad
C7 Kondensator 10 V 10 Mikrofarad
IC AD 7574
Hersteller: Analog Devices 089/570050
IC ICL7660 Spannungswandler
1 25poliger Sub-D-Stecker oder Kupplung, je nach Amiga

Das Ansprechen des Sound-Digitizers ist ebenso einfach wie sein Aufbau. Am besten eignet sich für ein Steuerprogramm eine Maschinen-Routine, da hier gewährleistet ist, daß mit der max. Geschwindigkeit gearbeitet wird. Das Beispielprogramm schreibt direkt in den Speicher des Amiga und spielt die Daten sofort ab. Dies geschieht mit einer kleinen Verzögerung. Verwendet werden können auch handelsübliche Abspielprogramme. Das Programm finden Sie im Kapitel zur parallelen Schnittstelle.

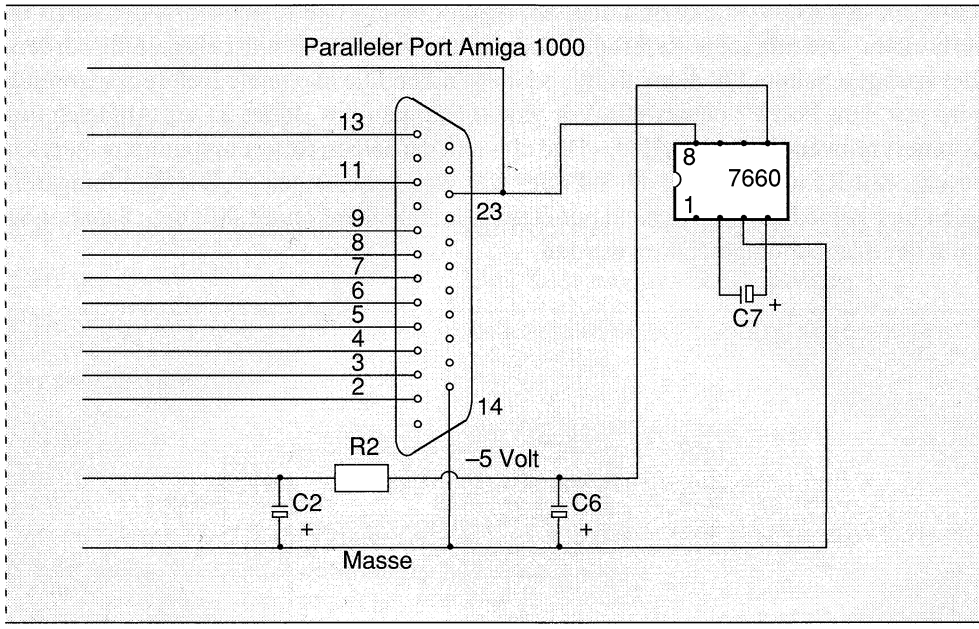
16.2: Der Amiga-Scanner

Es gibt verschiedene Möglichkeiten ein Bild zu digitialisieren. Die einfachste und billigste Variante ist die eines Scanners. Der Scanner, den wir hier vorstellen, ist ebenso einfach aufgebaut wie der Sound-Digitizer im vorhergehenden Kapitel. Er besteht aus einem Abtastkopf, der auf den Druckkopf des Druckers aufgesetzt wird, dieser wird per Software schrittweise bewegt und tastet so helle oder dunkle Punkte ab. Der Wert

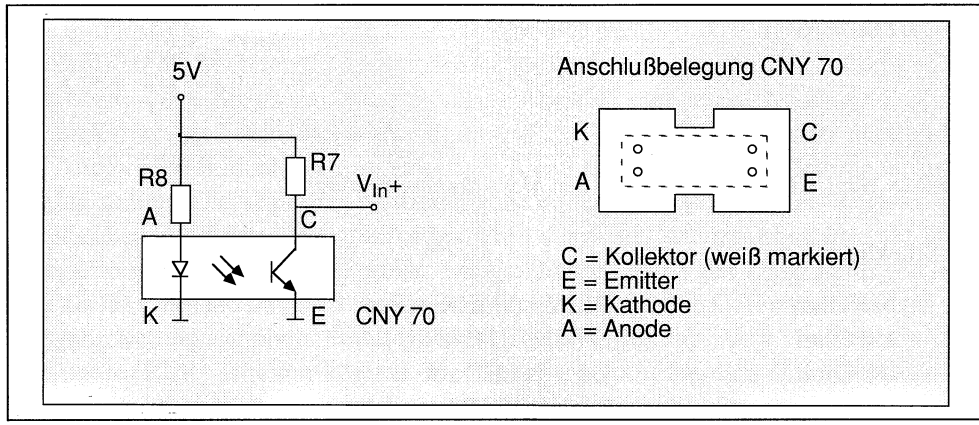


Z 16.1-1: *Die Schaltung des M&T-Sound-Ddigitizers (Teil 1)*

den wir von dem Abtastkopf erhalten, wird auf einen AD-Wandler gegeben, der uns 2^8 verschiedene Grauwerte liefern kann. Ausgelesen wird der AD-Wandler über den GamePort 0 und 1, da der Parallel-Port für den Drucker benötigt wird. Als Abtast-Kopf findet ein Optoreflexkoppler CNY70 Verwendung. Er wird in einem kleinen Röhrchen auf einer Platine anstelle des Druckkopfs auf dem Schlitten montiert. Der Sensor liefert bei einem dunklen Punkt die höchste Spannung. Er muß wie folgt beschaltet werden:



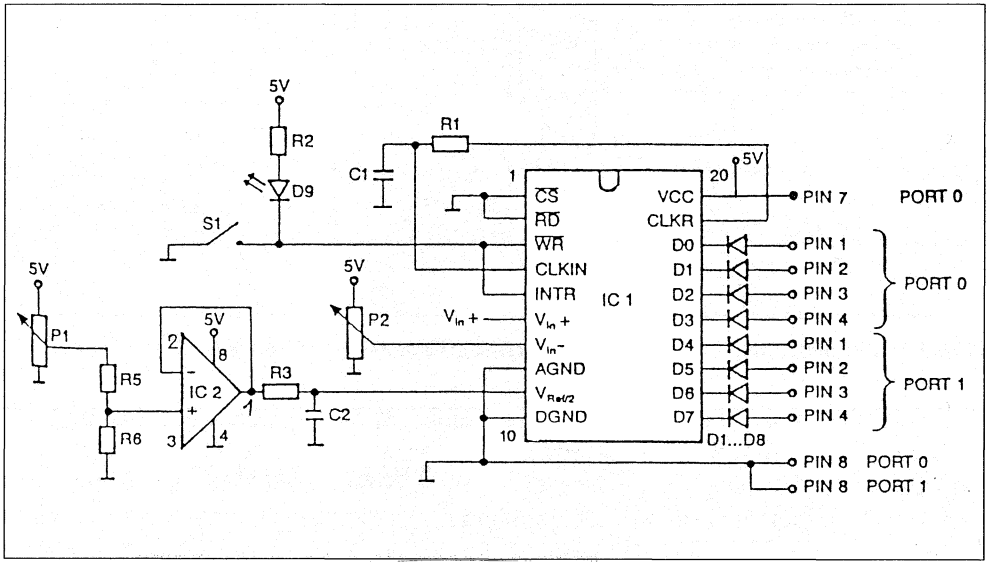
Z 16.1-1: *Die Schaltung des M&T-Sound-Digitizers (Teil 2)*



Z 16.2-1: *Die Beschaltung des Sensors*

Die zwei Versorgungsleitungen und die Datenleitung werden mit einer flexiblen Leitung zur Hauptplatine des Scanners geführt. Auf dieser Platine befindet sich der AD-Wandler. Ausgewählt wurde der Typ ADC 0804. Er läßt sich sehr einfach beschalten. Somit ist es möglich, die Schaltung auf nur wenige Bauteile zu begrenzen. Die Daten-

leitungen des AD-Wandler sind über Schutzdioden mit den Game-Ports des AMIGA verbunden. Die minimale Referenzspannung kann mit dem Poti P1 eingestellt werden. Der nachgeschaltete OP dient als Impedanzwandler. Die maximale Referenzspannung wird mit dem Poti P2 eingestellt. Vor jedem Scannen des Bildes ist ein Abgleich des Scanners notwendig, da nicht jedes Bild ein- und denselben Referenzspannungsbereich besitzt. Mit R1 und C1 wird der Takt des AD-Wandlers festgelegt. Die Wandlungszeit spielt hier keine Rolle, da es nicht auf Geschwindigkeit ankommt. Mit dem Schalter S1 kann der Scanner eingeschaltet werden.



Z 16.2-2: *Wenige Bauteile werden für den M&T-Scanner benötigt*

Die Stückliste:

R1	10 KOhm	P1,P2	50 KOhm Poti
R2	150 Ohm	D1–D8	1N4148
R3	180 Ohm	D9	LED rot
R5	15 KOhm	C1	150 Pikofarad
R6	33 KOhm	C2	0.1 Mikrofarad
R7	22 KOhm	IC1	ADC0804
R8	150 Ohm	IC2	LM 358

Die gescannten Daten können über die Register des Game-Port ausgelesen werden. Der Drucker kann mit folgenden Steuercodes gesteuert werden (Werte für den Epson FX80):

Kopf bewegen / Drucker initialisieren:

ESC,A,CHR$(1)	Line spacing auf 1/72 Zoll.
ESC,D,CHR$(64),CHR$(0)	Horizontaler Tabulator an die Stelle 64 setzen.
ESC,‹	Eine Zeile unidirektional drucken.
CHR$(10)	Zeilenvorschub auf 1/72 Zoll setzen.

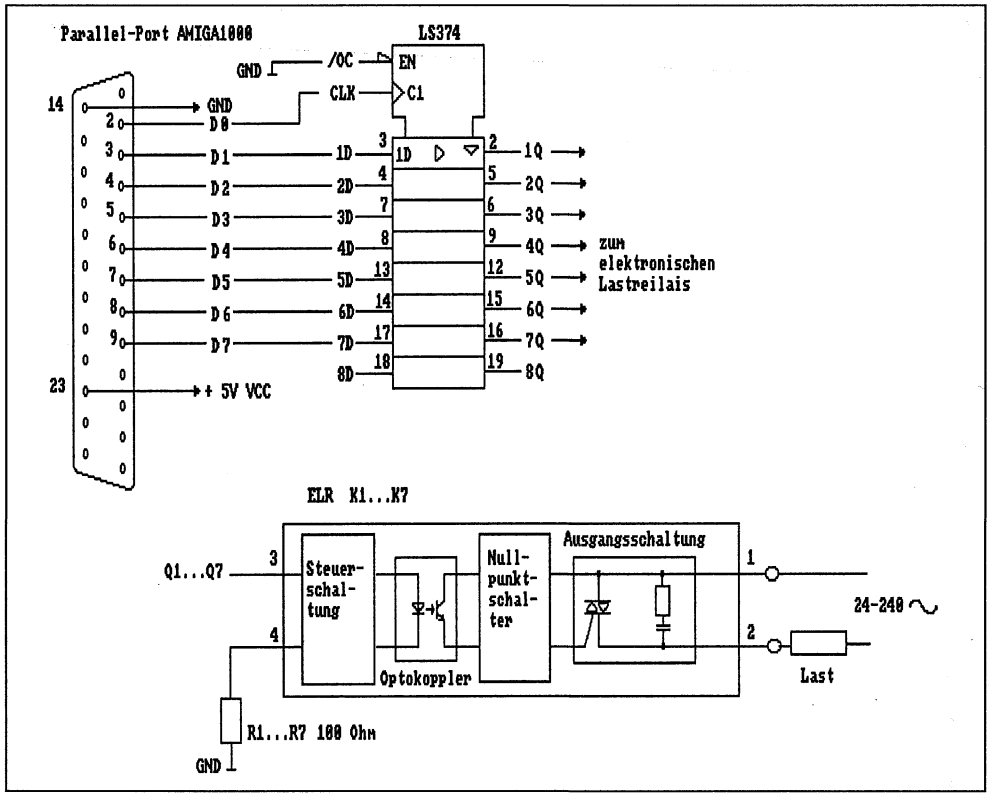
Scannen starten:

CHR$(9)	Zur Tab.-Position fahren vorbereiten.
ESC,K,CHR$(1).CHR$(0),CHR$(0)	Druckerkopf losfahren.

Tips zum Umgang mit der Printer-Device und die Anwendung der Steuercodes finden Sie im Amiga-Programmierhandbuch M&T-Nr. 90491.

16.3: Der Amiga als Schalter

Oftmals möchte man mit dem Computer auch externe Geräte schalten. Hierzu wird eine einfache Optokopplerkarte für den Computer benötigt. Das direkte Schalten von Transistoren als Schalter, ist nicht empfehlenswert. Zwischen externen Geräten und dem Computer sollte immer eine Trennung vorhanden sein, so daß bei einem Fehlerfall nie der Computer beschädigt wird. Für eine solche Anwendung bietet der Parallel-Port des Amiga wieder ideale Eigenschaften, da hier 8 Daten- und Steuerleitungen zur Verfügung stehen. Die Schaltung ist wieder ganz einfach. Ein 8-Bit-Daten-Register empfängt sieben der 8-Bit-Daten des Amiga. Pin 1 des 8-Bit Datenregisters wird auf Masse gelegt, somit ist der Baustein immer »eingeschaltet«. Bit 7 der Datenleitungen dient hier als Takt. Bei jedem positiven Flankenwechsel werden die Daten vom Eingang zum Ausgang weitergeschoben. Somit stehen 6 Bit zur Verfügung, die je ein elektronisches Lastrelais (ELR) schalten können. Der Schaltspannungsbereich dieser Relais reicht von 24 bis 280 Volt. Je nach Ausführung können 2 A bis 40 A geschaltet werden. Die Steuerleistung ist gering, da mit einer Steuerspannung von 3 bis 30 Volt gearbeitet werden kann. Somit sind sie TTL-kompatibel. Der Type B1 (Best.: V23100-S0032-B105) von Siemens kann z.B. 5 Ampere schalten. Dies entspricht ungefähr einer 1000-Watt-Glühbirne.



Z 16.3-1: *Mit dieser kleinen Schaltung kann der Amiga sehr einfach für Steueraufgaben eingesetzt werden*

Das softwaremäßige Setzen der Ausgänge ist denkbar einfach. In dem zu schreibenden Byte werden in den unteren 6 Bit die Bits gesetzt, dessen Ausgang eingeschaltet werden soll. Das höchste Bit, Bit 7, bleibt noch auf logisch low. Der Schreibvorgang wird nochmals wiederholt, diesmal mit gesetztem Bit 7:

```
move.b #%01010101,$BFE101
move.b #%11010101,$BFE101
```

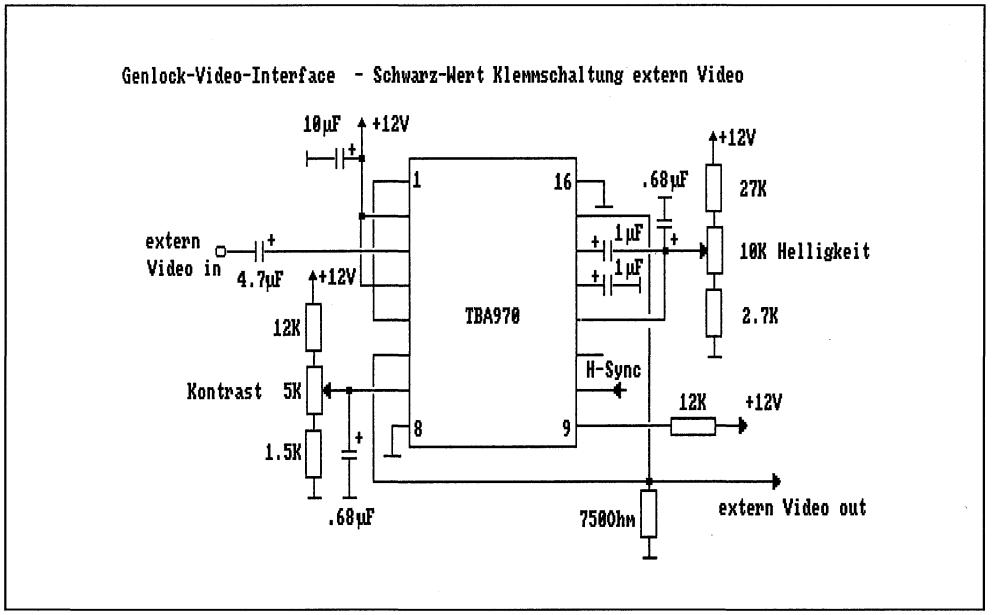
16.4: Das low-cost-Genlock-Interface

Ein Genlock-Interface ist ein Bildmischer-Interface. Eine externe Videoquelle wird mit dem Bild des Amiga vermischt, wobei das Amiga-Bild in den Vordergund oder in den Hintergrund »gelegt« werden kann. Professionelle Genlocks splitten das externe Videosignal auf, wobei man die einzelnen Farben Rot, Blau, Grün in analogen Werten erhält. Die wichtigsten Signale, die man bei einem solchen Zerlegen des Videosignals erhält, sind die vertikalen und horizontalen Synchronimpulse. Diese Impulse signalisieren den Start des Bildaufbaus, sowie den Start jeder einzelnen Zeile. Der vertikale Syn-

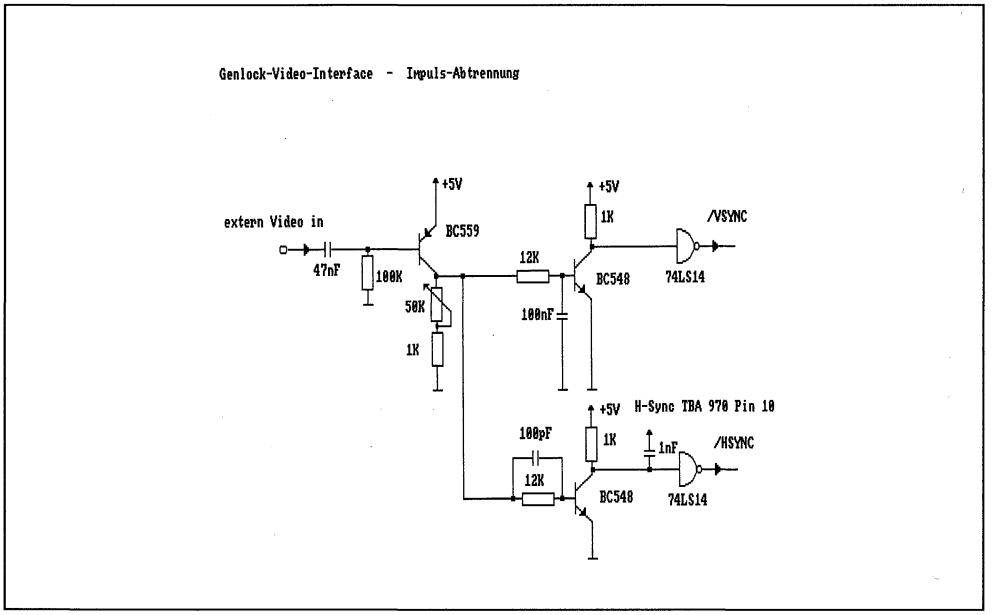
chronimpuls hat eine Frequenz von 50 Hz, was einem kompletten Bildaufbau von 20 Millisekunden entspricht. Der horizontale Synchronimpuls hat eine Dauer von 64 Mikrosekunden. Dies entspricht einer Zeilenfrequenz von 15625 Hz. Diese Synchronimpulse werden über den RGB-Stecker dem Amiga zugeführt. Die /VSYNC- und /HSYNC-Signale des RGB-Steckers gehen direkt zum Custom-Chip AGNUS. Ist bei diesem Chip im Register BPL1CON0 das Genlock-Video-Bit gesetzt, so versucht AGNUS seinen Bildaufbau mit den externen Signalen zu synchronisieren. Dieses Bit wird automatisch beim Booten des Amiga gesetzt, wenn externe Synchronsignale anliegen. Der Synchronisiervorgang dauert ca. 2 bis 4 Sekunden. Nachdem der Bildaufbau des Amiga mit dem externen Bild synchron ist, kann das Mischen der Bilder beginnen. Am einfachsten ist das Zusammenlegen der einzelnen Farben. Hierbei ergeben sich aber Mischwerte, die unerwünscht sind. Eine Schaltung muß her, die die Hintergrundfarbe des Amiga, z.B. schwarz, gegen das externe Signal austauscht. Dies kann über schnelle CMOS-Schalter erzielt werden. Der Amiga stellt hierfür ein Signal /ZD am RGB-Port zur Verfügung, mit dem ein solcher Schalter gesteuert werden kann. /ZD signalisiert die Verwendung des BitPlane-Registers 0 (Hintergrundfarbe) des Amiga. Durch Abfrage der digitalen RGB-Werte ist auch das Herausfiltern von anderen Farben (BitPlanes) möglich.

Ein Aufsplitten in RGB-Werte ist sehr aufwendig. In unserer Schaltung verwenden wir deshalb die komplette Bildinformation, d.h. das Videosignal wird zwischen Amiga und externer Quelle, je nach Farbe des Amiga, umgeschaltet. Es ergibt sich dabei fast die gleiche Qualität wie bei einem Umschalten der RGB-Werte. Der Nachteil, der sich bei unserer Version bemerkbar macht, zeigt sich beim Mischen von Farb- und Schwarzweißbildern. Wird ein Farb- und Schwarzweißbild gemischt, können die Farbsignale nicht einwandfrei ausgewertet werden. Das Resultat ist ein Schwarzweißbild. Da der Amiga 500 nur ein BAS- und kein FBAS-Signal enhält, ist das resultierende Bild immer schwarzweiß. Für Amiga 1000-Anwender möchten wir hier auf das Kapitel »PAL für den Amiga« hinweisen, da der Bildmischer nur bei einem PAL-FBAS/BAS- Signal funktioniert.

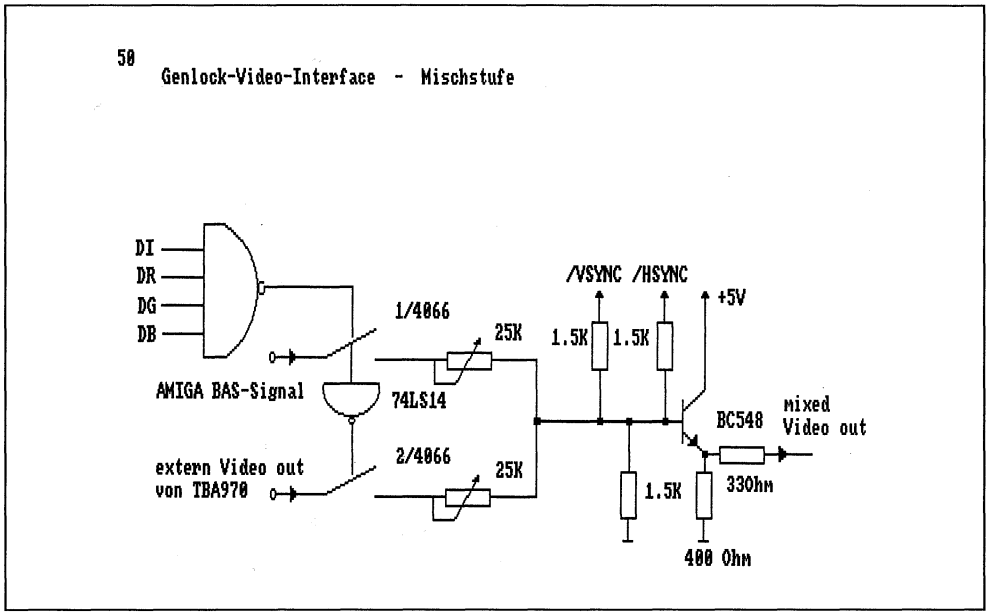
Aufgrund dieses Verfahrens ist unser Bildmischer sehr einfach aufgebaut. Die Synchronimpulse werden mit einer Transistorabtrennstufe, die aus einem Entstörfilter, sowie einem Hochpaß- und Tiefpaßfilter zum Herausfiltern der Synchron-Impulse besteht, herausgefiltert. Diese werden nochmals invertiert und dem Amiga zugeführt. Mit einer Schwarzwertklemmschaltung wird das externe Signal auf ein Pegelniveau angehoben, mit der die weitere Verarbeitung vereinfacht wird. Zudem läßt sich mit diesem Baustein der Kontrast und die Helligkeit nachregeln. Das so erhaltene Signal wird auf einen CMOS-Schalter gegeben, der erst bei der Hintergrundfarbe des Amiga das Signal durchschaltet. Das Amiga-BAS-Signal wird ebenfalls auf einen solchen Schalter gegeben, der nur dann durchschaltet, wenn nicht die Hintergrundfarbe des Amiga vorliegt. Die beiden Signale werden nach diesem Schalter addiert und auf die Transistor-Endstufe gegeben. Fertig ist das Genlock-Interface.



Z 16.4-1: *Genlock-Video-Interface – Impuls-Abtrennung (Teil 1)*



Z 16.4-1: *Genlock-Video-Interface – Impuls-Abtrennung (Teil 2)*



Z 16.4-1: *Genlock-Video-Interface – Impuls-Abtrennung (Teil 3)*

Stückliste:

ICs:
1 TBA 970 (Fairchild)
1 74 LS 14 Inverter/Schmitttrigger
1 4016/4066 CMOS-Analog-Schalter
1 74 LS 20 NAND 4 Eingänge

Transistoren:
3 BC 548
1 BC 559

Widerstände:
1 33 Ohm
1 400 Ohm (390 Ohm)
1 750 Ohm
1 1 KOhm
3 1.5 KOhm
1 2.7 KOhm
4 12 KOhm
1 27 KOhm
1 100 KOhm

Poti:
1 5 KOhm
1 10 KOhm
2 25 KOhm
1 50 KOhm

Kondensatoren:
2 .68 Mikrofarad /Tantal 20V
2 1 Mikrofarad /Tantal 20V
1 4.7 Mikrofarad /Tantal 20V
1 10 Mikrofarad /Tantal 20V
1 100 Pikofarad
1 1 Nanofarad
1 47 Nanofarad
1 100 Nanofarad

Das externe Videosignal kann von einem Videorekorder mit FBAS-Ausgang geliefert werden. Sollte der Amiga beim Booten nicht hochfahren, so ist die Sync-Abtrennstufe nicht richtig abgestimmt. Mit dem 50 KOhm-Poti am Eingangs-Transistor der Abtrennstufe ist die Schwellwertspannung für die folgenden Transistoren einzustellen. Bei jedem Verändern des Potis ist neu zu booten. Sollte das 50 KOhm-Poti schon auf Anschlag 0 Ohm stehen und der Amiga bootet nicht, so ist der in Reihe geschaltete 1 K-Widerstand zu entfernen. Die genaueste Einstellung kann man hier mit einem Oszilloskop machen, somit entfällt das lange Probieren.

Mit den 2 Potis am TBA970 läßt sich der Kontrast und die Helligkeit des externen Videosignals einstellen.

Die zwei 25 KOhm-Potis in der Mischstufe dienen zum Ein- und Ausblenden des jeweiligen Signals.

Die digitalen Farbwerte dienen hier zum Steuern der CMOS-Schalter. Dies kann auch das Signal /ZD (Hintergrundfarbe null) übernehmen.

16.5: Die Verwandlung: Aus Genlock wird ein Digitizer

In diesem Kapitel wollen wir eine Anregung geben, wie man aus dem oben angeführten Genlock-Interface einen Digitizer baut.

Da bei dem Genlock-Interface alle wichtigen Signale zur Verfügung stehen, ist der Umbau zum Video-Digitizer sehr einfach. Das vertikale Synchronsignal dient zur Erkennung, wann das Bild beginnt, das horizontale, wann die jeweilige Zeile beginnt. Somit kann unser Video-Digitzer mit dem vertikalen Signal gestartet werden. Nach diesem Signal folgen 312.5 Zeilen, ein Halbbild des Videosignals. Nach dem nächsten vertikalen Impuls werden die nächsten 312.5 Zeilen um eine Zeile versetzt gezeichnet, so daß sich ein komplettes Bild aus 625 Zeilen zusammensetzt. Ein Halbbild ist für das Digitialisieren vollkommend ausreichend.

Mit dem horizontalen Impuls wird signalisiert, daß nun die Bildinformation kommt. Ein horizontaler Impuls dauert 64 Mikrosekunden, wobei ca. 57 Mikrosekunden für die Bildinformation zur Verfügung steht. Bei einer horizontalen Auflösung von 320 Pixels haben wir dann für jeden Punkt ca. 57 Mikrosekunden / 320 Pixels = 178 Nanosekunden Zeit, um einen Punkt zu digitialisieren und einzulesen. Ein Buszyklus dauert aber schon ca. 279 Nanosekunden. Dies bedeutet, entweder weniger Punkte einzulesen, oder eine Zeile mehrmals zu digitialisieren. Die letztere Möglichkeit ist sicherlich die beste, hat aber den Nachteil, daß das Bild sich nicht bewegen darf, um Verzerrungen zu vermeiden. Für einen Punkt hat man so 2 mal 20 Millisekunden Zeit, ihn zu verarbeiten (da nur ein Halbbild ausgewertet wird). Für eine gesamte Zeile von 320 Pixels wird so max. ca. 1/100 Sekunde benötigt. Ein ganzes Bild benötigt somit, bei 200 Pixel vertikal, gerade ein paar Sekunden. Die eingelesenen Daten brauchen dann nur noch in das BitPlane-Format umgesetzt zu werden, um mehrere Grauabstufungen zu erhalten.

Kapitel 17

Die Janus-Library

Das *Janus-Library* ist das neueste Library, das an das Amiga-System »geheftet« wurde. Es dient als Brücke, beim Amiga 2000 und auch beim 500er bzw. 1000er mit SideCar, zur Kommunikation bzw. dem Datenaustausch zwischen dem PC- oder AT-kompatiblen Teil und dem Amiga-Hauptteil. Für C-Programmierer ist ein Link-File mit Namen »Janus.Library« auf der Workbench enthalten. Ein einfaches Beispiel, das zeigt, wie man Daten zwischen dem Amiga und dem PC austauschen kann, folgt nun:

```
 1  /*****************************
 2
 3  Janus-Demonstration
 4  last update 16/02/88
 5  von Frank Kremser und Jörg Koch
 6  © Markt & Technik 1988
 7
 8  *****************************
 9
10  Diese Demonstration sendet eine Datei namens Amigafile zum PC.
11  Dabei darf die Datei maximal 65535 Byte lang sein und das Amiga-
12  Programm, also dieses, muß zuerst gestartet werden.
13  Anschließend muß das zugehörige PC-Programm gestartet werden.
14  Auf dieser Diskette befindet sich ein Turbo-Pascal-Programm, das
15  aber zuerst zum PC gebracht werden muß. Dazu 'PCDisk' von der Work-
16  bench starten und anschließend im PC-Modus 'AREAD df0:pcjanus janus.pas'
17  starten.
18  Nun befindet sich das Programm auf dem PC. Es muß dann allerdings noch
19  mit Turbo-Pascal kompiliert werden. Das PC-Programm empfängt lediglich
20  die Daten, speichert sie aber nicht ab, was aber keine großen Änder-
21  rungen nötig machen würde.
22  Nach diesem Schema könnten beispielsweise auch die Positionsdaten der
23  Maus übertragen werden, so daß man mittels der Amiga-Maus auch eine
24  PC-Maus emulieren könnte, wenn noch der geeignete Treiber dazu vor-
25  handen ist. Eine lohnende Aufgabe!
26
26  *****************************/
```

```
28  #include<exec/types.h>                /* Include-Files laden */
29  #include<exec/memory.h>
30  #include<exec/ports.h>
31  #include<exec/devices.h>
32  #include<exec/io.h>
33  #include<exec/libraries.h>
34  #include<devices/trackdisk.h>
35  #include<libraries/dos.h>
36  #include<libraries/dosextens.h>
37
38  #extern struct MsgPort *CreatePort(); /* MsgPort extern deklarieren */
39  extern struct FileHandle *Open();
40
41  struct IOExtTD *request;
42  struct MsgPort *port;
43  struct FileHandle *fileh;
44
45  UBYTE *buffer;
46
47  main()
48   {
49    APTR mempos;
50    UWORD *lang,i,len;
51    UBYTE *data;
52                                       /* Pufferspeicher bereitstellen */
53    buffer = (UBYTE *)AllocMem(65535,MEMF_CLEAR);
54
55    mempos = AllocAbs(4,0x202000);     /* Addresse zur Datenübertragung */
56                                       /* freihalten (4 Byte) */
57    lang = data = 0x202000;            /* Variablen werden auf diese */
58                                       /* Addresse gelegt */
59    fileh = Open("Amigafile",MODE_OLDFILE); /* Datei wird geöffnet */
60    len = Read(fileh,buffer,65535);    /* und in den Puffer gelesen */
61    Close(fileh);                      /* Datei wird geschlossen */
62    *lang = len;                       /* Dateilänge wird übertragen */
63    while(*lang != 0);                 /* Warten, bis Übertragung
64    bestätigt */
65
66    for(i = 0; i<len; i++)
67   {
68    *data = *(buffer + i); /* Byte übertragen */
69    while(*data != 255-(*(buffer+i)));/* Warten auf Bestätigung */
70   }
71
72  *data = *(buffer + len - 1);       /* Weiteres Byte übertragen, um */
73                                     /* PC-Programm zu beenden */
74  FreeMem(buffer,65535);             /* Datenpuffer löschen */
```

17.1: AllocJanusMem

Syntax: `Ptr = AllocJanusMem(Size,Type);`

Funktion: Stellt Speicher für den Datenaustausch mit dem PC-Teil zur Verfügung.

Parameter: Size – Größe des benötigten Speicherbereiches
Type – Speichertyp

Ergebnis: Ptr – Zeiger auf den Speicherbereich

Datentyp: Long Size – D0
LongType – D1
Zeiger Ptr – D0

Sonstiges: Type kann mit folgenden Makros gesetzt werden:
MEMF_PARAMETER
MEMF_BUFFER
sowie
MEM_BYTEACCESS
MEM_WORDACCESS
MEM_GRAPHICACCESS
MEM_IOACCESS

17.2: CheckJanusInt

Syntax: `Status = CheckJanusInt(Jintnum);`

Funktion: Liest ein Statusbyte für einen PC-Interrupt aus dem DualPorted-RAM.

Parameter: Jintnum – Interruptnummer

Ergebnis: Status – gelesenes Statusbyte

Datentyp: Byte Jintnum – D0
Byte Status – D0

17.3: FreeJanusMem

Syntax: `FreeJanusMem(Ptr,Size);`

Funktion: Gibt den belegten Speicherbereich für den Datenaustausch wieder frei.

Parameter: Ptr – Zeiger auf den Speicherbereich
Size – Größe des Speicherbereiches

Datentyp: Zeiger Ptr – A1
Long Size – D0

Sonstiges: Falls ein Speicherbereich freigesetzt werden soll, der schon frei ist, stürzt das System ab.

17.4: GetJanusStart

Syntax: `Ptr = GetJanusStart ();`

Funktion: Gibt die Anfangsadresse des DualPorted-RAM's zurück.

Ergebnis: Ptr – Zeiger auf das DualPorted-RAM

Datentyp: Zeiger Ptr – D0

17.5: GetParamOffset

Syntax: `Offset = GetParamOffset (Jintnum);`

Funktion: Gibt den Offsetwert für den bezeichneten Interrupt zurück.

Parameter: Jintnum-Interrupt, dessen Offset zurückgegeben werden soll.

Ergebnis: Offset – Offsetadresse des Interrupts

Datentyp: Byte Jintnum – D0
Word Offset – D0

17.6: JBCopy

Syntax: `JBCopy (Source,Destination,Length);`

Funktion: Kopiert einen Speicherbereich.

Parameter: Source – Anfangsadresse des zu kopierenden Speicherbereiches
Destination – Anfangsadresse des Bereiches, in den kopiert werden soll
Length – Länge des Bereiches, der kopiert werden soll.

Datentyp: Zeiger Source – A0
Zeiger Destination – A1
Long Length – D0

17.7: JanusLock

Syntax: `JanusLock (Ptr);`

Funktion: Ermittelt ein »Lock«, das für andere Janus-Library-Befehle benötigt wird.

Parameter: Ptr – nicht belegter Zeiger. Dieser Zeiger zeigt nach Aufruf dieser Funktion auf die Lock-Structure.

Datentyp: Zeiger Ptr – A0

17.8: JanusMemBase

Syntax: `Ptr = JanusMemBase (Type);`

Funktion: Ermittelt die Startadresse eines bestimmten Speicherblockes des Dual-Ported-RAMs.

Parameter: Type – Speichertyp, dessen Startadresse ermittelt werden soll.

Ergebnis: Ptr – Zeiger auf den Start des gesuchten Speichers.

Datentyp: LONG Type – D0
Zeiger Ptr – D0

Sonstiges: Die möglichen Speichertypen ersehen Sie aus Kapitel 17.1.

17.9: JanusMemToOffset

Syntax: `Offset = JanusMemToOffset (Ptr);`

Funktion: Wenn »Ptr« auf einen Speicherbereich des Dual-Ported-RAMs zeigt, so wird der Offset ermittelt, also die Adresse, die zur Basisadresse des angesprochenen Bereiches hinzuaddiert werden muß.

Parameter: Ptr – Zeiger auf eine Speicherstelle, zu der der Offset ermittelt werden soll.

Ergebnis: Offset – Offset zur angegebenen Adresse.

Datentyp: Zeiger Ptr – D0
Zeiger Offset – D0

Sonstiges: Als Offset wird die relative Adresse genannt, zu der man die Basisadresse des anzusprechenden Speicherbereiches addieren muß, um die absolute Adresse zu erhalten.

17.10: JanusMemType

Syntax: `Type = JanusMemType (Ptr);`

Funktion: Ermittelt den Speichertyp zu einer angegebenen Adresse.

Parameter: Ptr – Zeiger auf die Speicherstelle, deren Speichertyp ermittelt werden soll.

Ergebnis: Type – Ermittelter Speichertyp.

Datentyp: Zeiger Ptr – D0
LONG Type – D0

17.11: JanusUnLock

Syntax: `JanusUnLock(Ptr);`

Funktion: Löscht ein »Lock«, auf das »Ptr« zeigt.

Parameter: Ptr – Zeiger auf die zu löschende Lock-Structure.

Datentyp: Zeiger Ptr – A0

Referenz: siehe auch JanusLock

17.12: SendJanusInt

Syntax: `SendJanusInt(Jintnum);`

Funktion: Erzeugt einen Interrupt mit der Nummer jintnum auf der PC-Seite.

Parameter: Jintnum – Nummer des Interrupts, der erzeugt werden soll.

Datentyp: LONG Jintnum – D0

Referenz: siehe auch CheckJanusInt

17.13: SetJanusEnable

Syntax: `OldEnable = SetJanusEnable(Jintnum, Newvalue);`

Funktion: Setzt einen Interrupt ein oder aus.

Parameter: Jintnum – Interrupt, der ein- oder ausgeschaltet werden soll.
Newvalue – Ist dieser Wert gleich 0, wird der Interrupt ausgeschaltet, bei 1 wird er eingeschaltet.

Ergebnis: OldEnable – Alter Zustand des Interrupts. Es gilt das Gleiche, wie bei Newvalue.

Datentyp: LONG Jintnum – D0
Byte Newvalue – D1

Byte OldEnable – D0

17.14: SetJanusHandler

Syntax: `OldHandler = SetJanusHandler(Jintnum, Intserver);`

Funktion: Setzt eine neue Interruptroutine für einen bestimmten Interrupt.

Parameter: Jintnum – Interruptnummer, zu dem eine Routine gesetzt werden soll.

Intserver – Zeiger auf die Routine, die als Interruptroutine gesetzt werden soll.

Ergebnis: OldHandler – Zeiger auf die alte Interruptroutine.

Datentyp: LONG Jintnum – D0
Zeiger Intserver – A1

Zeiger OldHandler – D0

17.15: SetJanusRequest

Syntax: `OldRequest = SetJanusRequest (Jintnum, Newvalue);`

Funktion: Setzt oder löscht einen »Interruptrequest«.

Parameter: Jintnum – Nummer des Interrupts, dessen Request modifiziert werden soll.

Newvalue – Wird hier eine 0 übergeben, so wird der Request gelöscht, bei einer 1 wird er gesetzt.

Ergebnis: OldRequest – Alter Zustand des Interruptrequests.

Datentyp: LONG Jintnum – D0
Byte Newvalue – D1

Byte OldRequest – D0

17.16: SetParamOffset

Syntax: `OldOffset = SetParamOffset (Jintnum, Offset);`

Funktion: Diese Funktion setzt einen neuen Offset für den betroffenen Interrupt.

Parameter: Jintnum – Interruptnummer, für den ein neuer Offset gesetzt werden soll.
Offset – Zu setzender Offset.

Ergebnis: OldOffset – Alter Offset des betroffenen Interrupts.

Datentyp: LONG Jintnum – D0
WORD Offset – D1

WORD OldOffset – D0

Kapitel 18

Das Expansion-Library

Das Expansion-Library bietet die Möglichkeit, ohne großen Aufwand Änderungen an der Systemkonfiguration vorzunehmen. Um die Funktionen auch von der Maschinenebene aufrufen zu können, sind unter dem Punkt Datentyp auch die Register angegeben, in denen die Parameter angegeben werden.

Eine Anwendung der Expansion-Library ist das Ändern der Systemdaten eines Floppy-Laufwerkes. Hierzu nun ein Beispielprogramm, daß dies handhabt.

```
 1  /*******************************
 2  Demonstration zur
 3  Expansion-Library
 4  last update 16/02/88
 5  von Frank Kremser und Jörg Koch
 6  © Markt & Technik 1988
 7
 8  *******************************
 9
10  Diese Demonstration ändert die Parameter des 2. Laufwerkes so ab, daß
11  keine normale Diskette mehr gelesen werden kann.
12  Eine normal formatierte Diskette kann allerdings in diesem Format
13  beschrieben und anschließend auch gelesen werden.
14  Die uns bekannten Formatierprogramme unterstützen allerdings nicht
15  solche Parameteränderungen.
16  Trickreiche Programmierer können auf diese Weise unerhört einfach
17  einen Kopierschutz erstellen.
18
19  *******************************/
20
21  #include <exec/types.h                          /* Include-Files laden */
22  #include <exec/memory.h
23  #include <exec/ports.h
24  #include <exec/devices.h
25  #include <exec/io.h
26  #include <exec/libraries.h
27  #include <devices/trackdisk.h
```

```
28 #include <libraries/dos.h
29 #include <libraries/dosextens.h
30 #include <libraries/expansion.h
31
32 char dosName[] = "dfl:"; /* Dos-Name des ersten Laufwerkes */
33 char execName[] = "trackdisk.device"; /* Device-Name */
34
35 ULONG parmPkt[] = /* Parameter */
36  {
37   0,                  /* Frei */
38   0,                  /* Frei */
39   1,                  /* Geraete-Nummer */
40   0,                  /* OpenDevice-Flags */
41   11,                 /* table upper bound */
42   512>>2,             /* Anzahl der Langworte im Block */
43   0,                  /* Sektor-Anfang */
44   2,                  /* Anzahl der Schreib-/Lesekoepfe */
45   1,                  /* Sektoren pro log. Block */
46   9, /* Norm:        11 */ /* Sektoren pro Track */
47   2,                  /* 2 Bootblocks */
48   0,                  /* unbenutzt */
49   0,                  /* Interleave-Faktor */
50   0,                  /* erster Cylinder */
51   35, /* Norm:       79 */ /* letzter Cylinder */
52   5                   /* Anzahl der Buffer */
53  };
54
55 struct DeviceNode *node, *MakeDosNode();
56 struct ExpansionBase *ExpansionBase;
57 main() /* HAUPTPROGRAMM */
58  {
59   parmPkt[0] = (ULONG)dosName; /* Zusatzinformationen eintragen */
60   parmPkt[1] = (ULONG)execName; /* Library öffnen */
61
62   ExpansionBase = OpenLibrary("expansion.library",0);
63   AddDosNode(0,1,MakeDosNode(parmPkt)); /* Neue Parameter setzen */
64
65   CloseLibrary(ExpansionBase); /* Library schließen */
66  }
```

18.1: AddDosNode

Syntax: `ok = AddDodNode(BootPri,Flags,DeviceNode);`

Funktion: Diese Funktion fügt ein Disk-Laufwerk in das System ein.

Parameter: BootPri – Boot-Priotität
Flags – Wird hier 1 angegeben, so wird ein Handlerprozeß sofort begonnen.
DeviceNode – Dies ist eine normale Device-Node-Structure, die dem Laufwerk zugewiesen wird.

Ergebnis: Ok – ist ungleich 0, wenn alles in Ordnung war

Datentyp: BYTE BootPri – D0
BIT Flags – D1
Node DeviceNode – A0

BYTE Ok – D0

Referenz: siehe auch MakeDosNode

18.2: MakeDosNode

Syntax: `DeviceNode = MakeDosNode(ParameterPkt);`

Funktion: Diese Funktion erstellt eine DOS-Node-Structure für ein bestimmtes Laufwerk. Die dazu benötigten Daten müssen mit übergeben werden.

Parameter: ParameterPkt – Ist ein Longword-Array, das alle benötigten Daten enthält.

Ergebnis: DeviceNode – Zeiger auf die initialisierte DOS-Node-Structure für das Laufwerk.

Datentyp: Array ParameterPkt – A0
Pointer DeviceNode – D0

Sonstiges: Ein Beispiel für die Verwendung dieses Befehls und besonders der Parameter-Liste finden Sie oben.

Referenz: siehe auch AddDosNode

18.3: AddConfigDev

Syntax: `AddConfigDev(ConfigDev);`

Funktion: Fügt eine neue ConfigDev-Structure in das System ein.

Parameter: ConfigDev – eine initialisierte Structure.

Datentyp: Pointer ConfigDev – A0

Sonstiges: Diese Structure wird in eine Liste von Configuration-Devices des Systems eingebunden.

Referenz: siehe auch RemConfigDev

18.4: AllocBoardMem

Syntax: `StartSlot = AllocBoardMem(SlotSpec);`

Funktion: Stellt Speicherbereich aus einer Erweiterung zur Verfügung.

Parameter: SlotSpec – Speichergröße aus der Konfiguration der Karte (siehe Autokonfiguration).

Ergebnis: StartSlot – Die Slotnummer, in der sich die Karte befindet.

Datentyp: Byte SlotSpec – D0
Byte StartSlot – D0

Sonstiges: Diese Funktion kehrt mit der Slotnummer der Karte zurück, auf deren Speicher zugegriffen werden darf. Mittels dem EC_ MEMADDR-Makro kann diese Nummer in eine Speicheradresse gewandelt werden.

Referenz: siehe auch AllocExpansionMem
FreeExpansionMem
FreeBoardMem

18.5: AllocConfigDev

Syntax: `ConfigDev = AllocConfigDev();`

Funktion: Stellt eine ConfigDev-Structure zur Verfügung.

Ergebnis: ConfigDev – Zeiger auf eine ConfigDev-Structure.

Datentyp: Zeiger ConfigDev – D0

Sonstiges: Die Structure, deren Startadresse zurückgegeben wird, ist noch nicht initialisiert.

Referenz: siehe auch FreeConfigMem

18.6: AllocExpansionMem

Syntax: `StartSlot = AllocExpansionMem(NumSlots,SlotOffset);`

Funktion: Diese Funktion stellt Speicher auf einer Erweiterungskarte zur Verfügung.

Parameter: NumSlots – Die Anzahl der benötigten Slots
SlotOffset – Begrenzungsoffset für StartSlot

Ergebnis: StartSlot – Die Slotnummer des belegten Slots

Datentyp: Byte NumSlots – D0
Byte SlotOffset – D1

ByteStartSlot – D0

Sonstiges: Diese Funktion berücksichtigt alle Regeln, die für Erweiterungskarten in Hinsicht auf Speicherbelegungen gelten (siehe Autokonfiguration).

Referenz: siehe auch FreeExpansionMem
AllocBoardMem
FreeBoardMem

18.7: ConfigBoard

Syntax: `Error = ConfigBoard(Board,ConfigDev);`

Funktion: Erweiterungskartenkonfigurierung

Parameter: Board – Die Adresse, an der das Erweiterungsboard liegt.
ConfigDev – eine initialisierte ConfigDev-Structure.

Ergebnis: Error – ist ungleich 0, wenn ein Fehler aufgetreten ist.

Datentyp: Zeiger Board – A0
Zeiger ConfigDev – A1

Byte Error – D0

Sonstiges: Diese Funktion wird nur beim Booten des Systems benötigt. Zu dieser Zeit liegen alle Erweiterungskarten an der gleichen Stelle im Speicherbereich, doch wegen der Kompatibilität mit späteren Systemen muß diese Adresse als Parameter angegeben werden.

Referenz: siehe auch FreeConfigDev

18.8: ConfigChain

Syntax: `Error = ConfigChain(BaseAddr);`

Funktion: Zentrale Konfigurationsroutine

Parameter: BaseAddr – Zeiger auf die Basisadresse des Expansionsbereiches.

Ergebnis: Error – Ist ungleich 0, falls ein Fehler aufgetreten ist.

Datentyp: Zeiger BaseAddr – A0

Byte Error – D0

Sonstiges: Dies ist die zentrale Routine der Autokonfiguration. Diese Routine ruft alle anderen Funktionen auf, die zur Konfiguration benötigt werden und bindet alle gefundenen Karten in das System ein.

Referenz: siehe auch FreeConfigDev

18.9: FindConfigDev

Syntax: `ConfigDev = FindConfigDev(OldConfigDev,Manufacturer, Product);`

Funktion: Sucht nach einer bestimmten ConfigDev-Structure.

Parameter: OldConfigDev – Zu suchende Structure
Manufacturer – Herstellercode
Product – Produktcode

Ergebnis: ConfigDev – gefundene Structure

Datentyp: Zeiger OldConfigDev – A0
Long Manufacturer – D0
Long Product – D1

Zeiger ConfigDev – D0

Sonstiges: Diese Funktion sucht im Prinzip in allen eingebundenen Erweiterungskarten nach den Eintragungen, die als Parameter angegeben werden und gibt eventuell die gefundene ConfigDev-Structure zurück.

18.10: FreeBoardMem

Syntax: `FreeBoardMem(StartSlot,SlotSpec);`

Funktion: Diese Funktion gibt einen mit AllocBoardMem reservierten Speicherbereich auf einer Erweiterungskarte wieder frei.

Parameter: StartSlot – Slotnummer der betroffenden Karte
SlotSpec – Speichergröße als Index (siehe Autokonf.)

Datentyp: Byte StartSlot – D0
Byte SlotSpec – D1

Sonstiges: Falls ein Speicherbereich freigegeben werden soll, der bereits frei ist, stürzt das System ab.

Referenz: siehe auch AllocExpansionMem
FreeExpansionMem
AllocBoardMem

18.11: FreeConfigDev

Syntax: `FreeConfigDev(ConfigDev);`

Funktion: Löscht eine ConfigDev-Structure

Parameter: ConfigDev – Zeiger auf eine ConfigDev-Structure

Datentyp: Zeiger ConfigDev – A0

Sonstiges: FreeConfigDev stellt den für eine ConfigDev-Structure reservierten Speicherbereich wieder zur Verfügung.

Referenz: siehe auch AllocConfigDev

18.12: FreeExpansionMem

Syntax: FreeExpansionMem(StartSlot,NumSlots);

Funktion: Diese Funktion stellt den von Erweiterungskarten belegten Speicherplatz wieder zur Verfügung.

Parameter: StartSlot – Nummer des Slots, ab dem freigestellt werden soll.
NumSlots– Anzahl der Slots, die freigestellt werden sollen.

Datentyp: StartSlot – D0
NumSlots– D1

Sonstiges: Soll ein Speicherbereich zur Verfügung gestellt werden, der bereits frei ist, so stürzt das System ab.

Referenz: siehe auch AllocExpansionMem
AllocBoardMem
FreeBoardMem

18.13: GetCurrentBinding

Syntax: `Actual = GetCurrentBinding(CurrentBinding,Size);`

Funktion: Liest den Konfigurationsbereich eines Erweiterungsboards.

Parameter: CurrentBinding – Zeiger auf eine CurrentBinding-Structure
Size – Größe der Benutzer-Binddriver-Structure

Ergebnis: Actual – Die wirkliche Größe der CurrentBinding-Structure

Datentyp: Zeiger CurrentBinding – A0
Byte Size – D0

Actual – D0

Referenz: siehe auch SetCurrentBinding

18.14: ObtainConfigBinding

Syntax: `ObtainConfigBinding();`

Funktion: Bindet die Driver der Erweiterungskarten zu den ConfigDev-Structures.

Referenz: siehe auch ReleaseConfigBinding

18.15: ReadExpansionByte

Syntax: `Byte = ReadExpansionByte(Board,Offset);`

Funktion: Liest ein Konfigurationsbyte aus der Erweiterungskarte.

Parameter: Board – Zeiger auf den Erweiterungskartenbereich
Offset – Zu lesendes Konfigurationsbyte

Ergebnis: Byte –gelesenes Konfigurationsbyte

Datentyp: Zeiger Board – A0
Long Offset – D0

Byte Byte – D0

Sonstiges: Das Konfigurationsbyte setzt sich aus zwei Nibbles zusammen. Diese Nibbles sind im Kapitel 6 ausführlich beschrieben.

Referenz: siehe auch WriteExpansionByte
ReadExpansionRom

18.16: ReadExpansionRom

Syntax: `Error = ReadExpansionRom(Board,ConfigDev);`

Funktion: Liest einen Bereich aus dem ROM der Erweiterungskarte in den Bereich cd_ROM der ConfigDev-Structure.

Parameter: Board – Zeiger auf den Beginn des Erweiterungskarten-Dekodierbereiches.
ConfigDev – Zeiger auf eine ConfigDev-Structure

Ergebnis: Error – Ist ungleich 0, wenn ein Fehler aufgetreten ist.

Datentyp: Zeiger Board – A0
Zeiger ConfigDev – A1

Byte Error – D0

Sonstiges: Bis auf die ersten zwei Nibbles werden alle Nibbles invertiert.

Referenz: siehe auch ReadExpansionByte
WriteExpansionByte

18.17: ReleaseConfigBinding

Syntax: `ReleaseConfigBinding();`

Funktion: Erlaubt anderen Funktionen das Einbinden von Erweiterungskarten.

Referenz: siehe auch ObtainConfigBinding

18.18: RemConfigDev

Syntax: `RemConfigDev(ConfigDev);`

Funktion: Löscht eine ConfigDev-Structure aus dem System.

Parameter: ConfigDev – Zu löschende ConfigDev-Structure.

Datentyp: Zeiger ConfigDev – A0

Referenz: siehe auch AddConfigDev

18.19: SetCurrentBinding

Syntax: `SetCurrentBinding(CurrentBinding,Size);`

Funktion: Setzt den Konfigurationsbereich einer Erweiterungskarte.

Parameter: CurrentBinding – Zeiger auf eine CurrentBinding-Structure
Size – Größe der Benutzer-Binddriver-Structure

Datentyp: Zeiger CurrentBinding – A0
Byte Size – D0

Sonstiges: Wenn Size größer angegeben wird, als die ideale CurrentBinding-Größe, wird nichts gesetzt.

Referenz: siehe auch GetCurrentBinding

18.20: WriteExpansionByte

Syntax: `Error = WriteExpansionByte(Board,Offset,Byte)`

Funktion: Schreibt ein Byte in den Konfigurationsbereich einer Erweiterungskarte.

Parameter: Board – Zeiger auf den Beginn des Erweiterungskarten-Decodierbereiches
Offset – Logische Adresse des zu schreibenden Bytes
Byte – Zu schreibendes Byte

Ergebnis: Error – ist ungleich 0, falls ein Fehler aufgetreten ist.

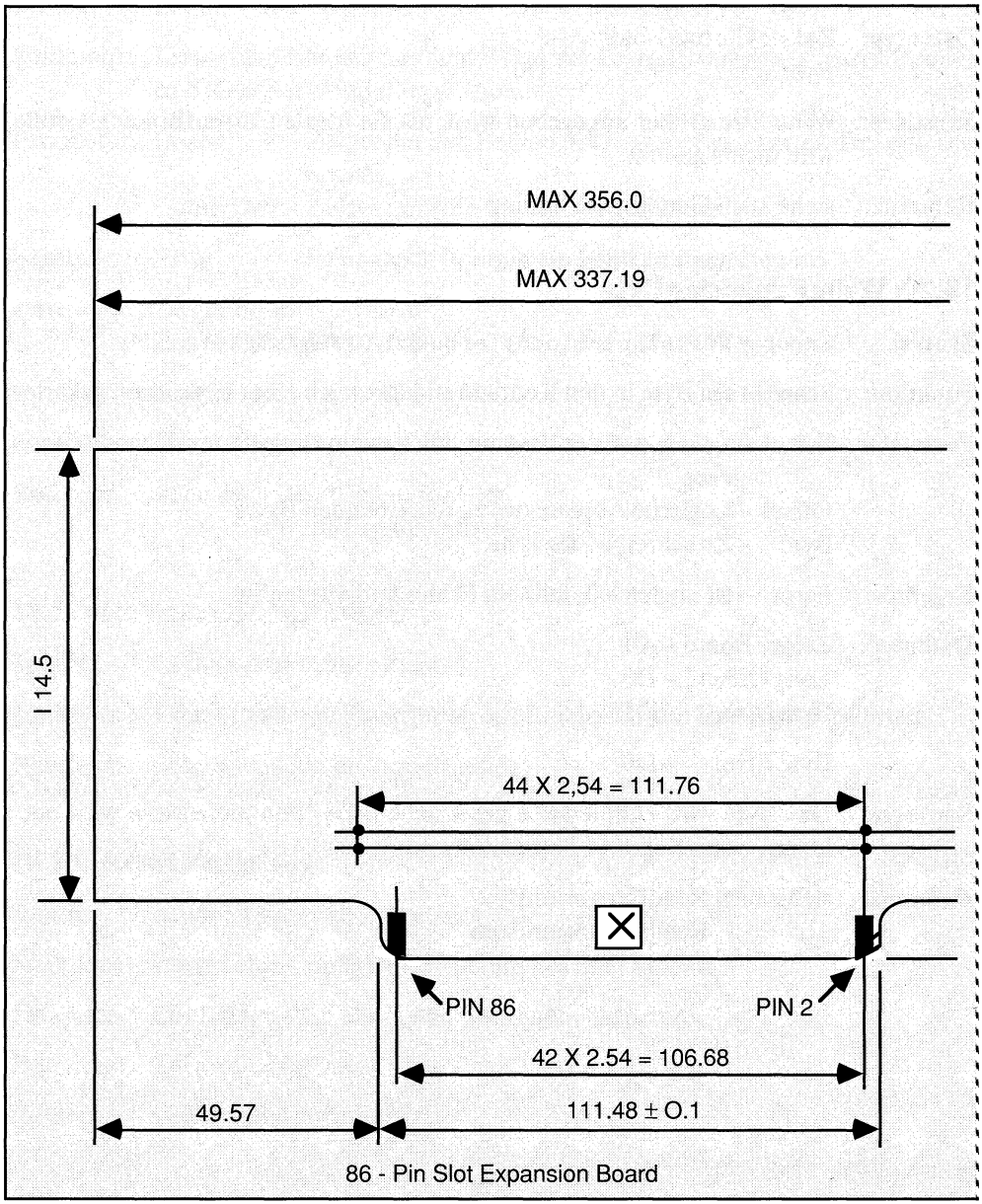
Datentyp: Zeiger Board – A0
Long Offset – D0
Byte Byte – D1

Byte Error – D0

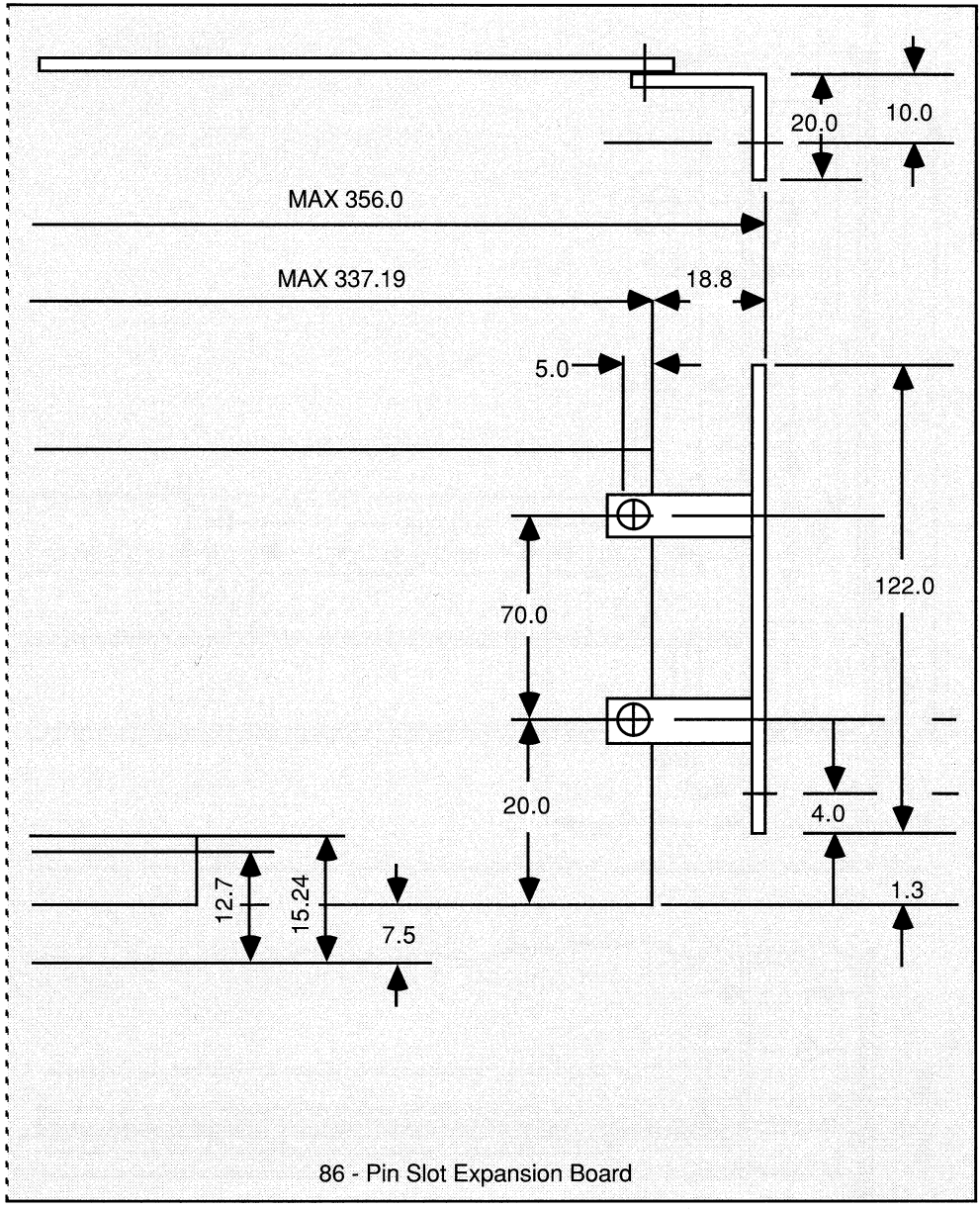
Sonstiges: Das Byte wird Nibbleweise geschrieben. Die Funktionsweise wird noch klarer, wenn Kapitel 6 betrachtet wird.

Referenz: siehe auch ReadExpansionByte
ReadExpansionRom

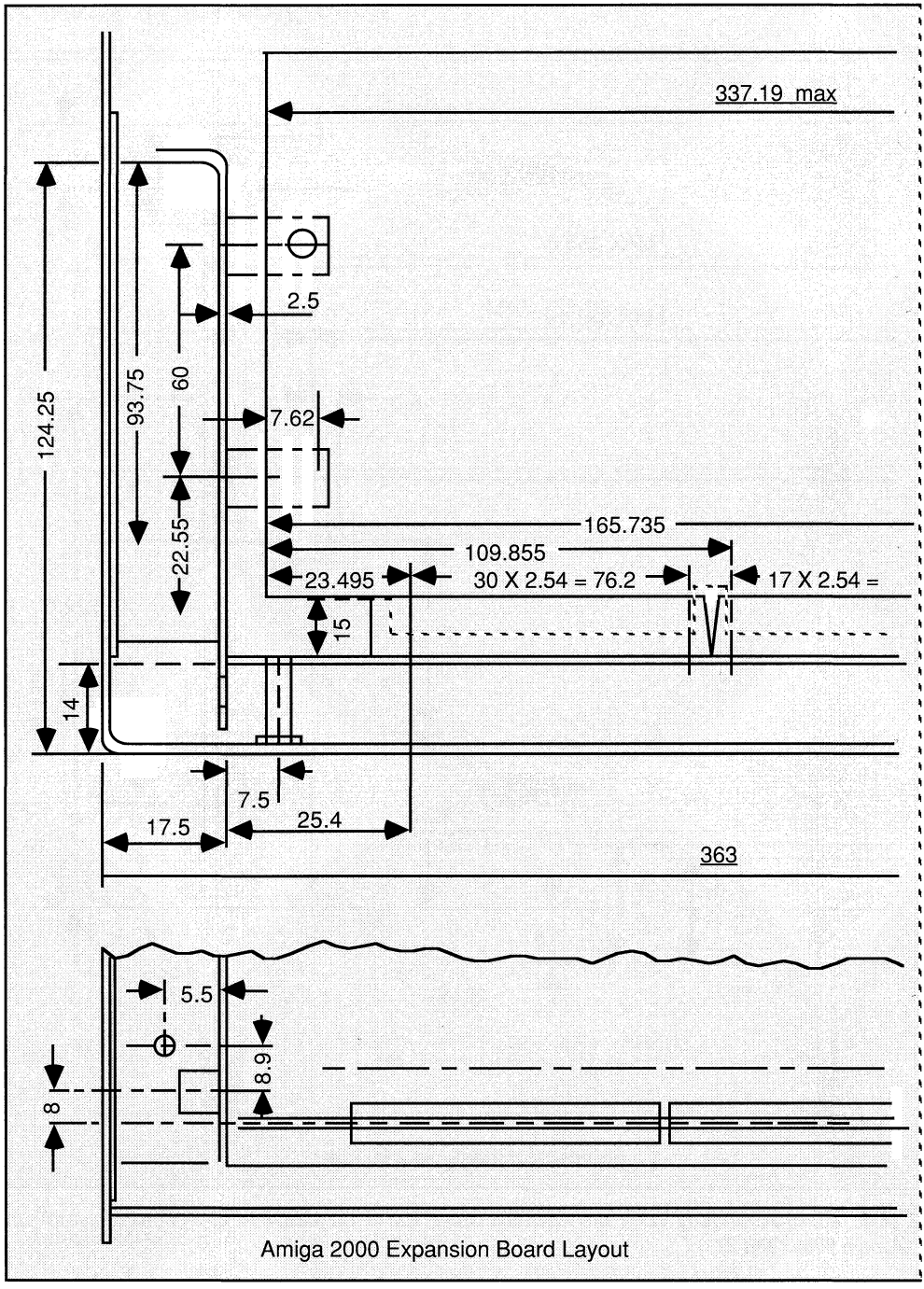
Anhang A: Kartengrößen



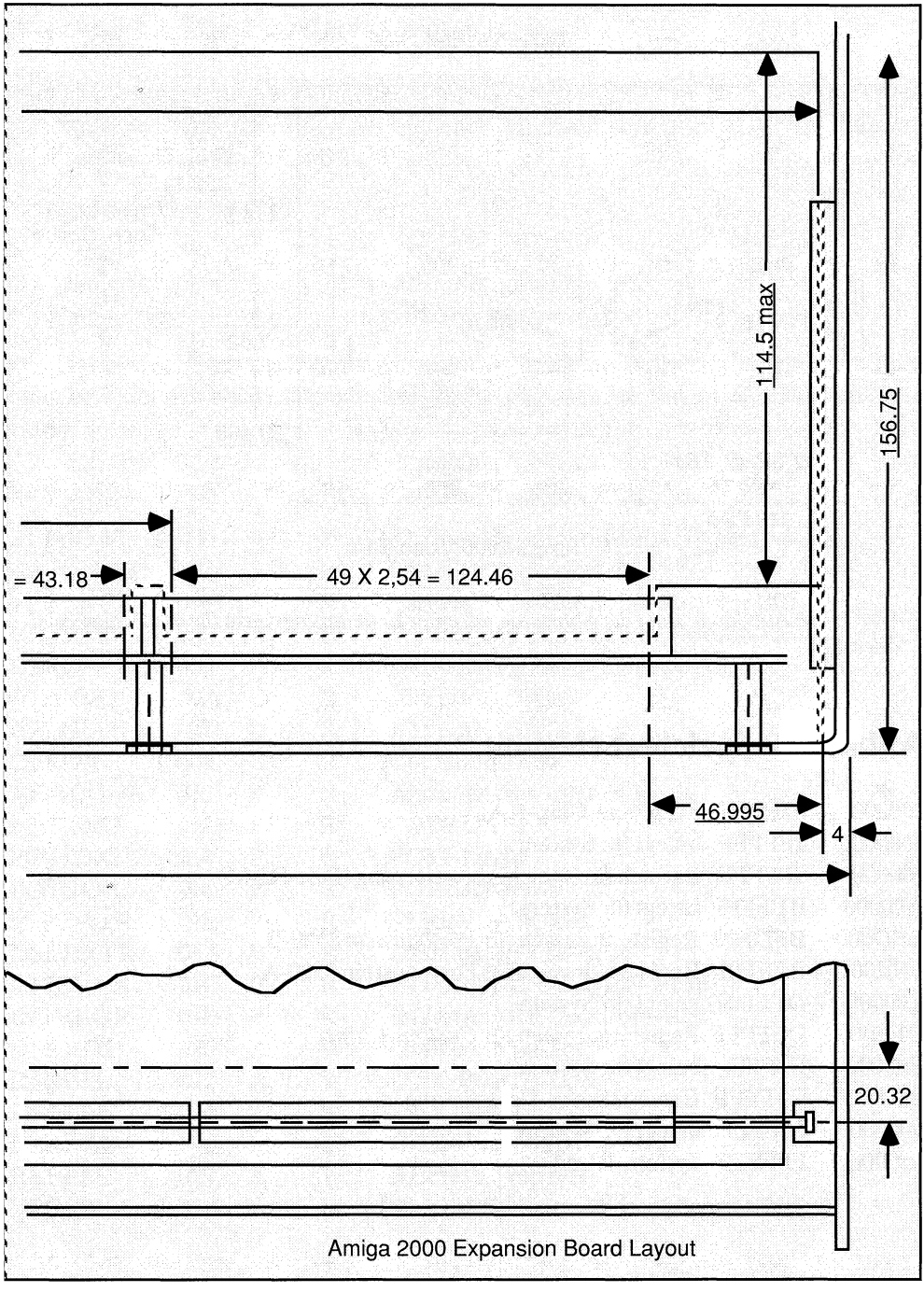
Z A-1: *Diese Abbildung zeigt die Abmaße einer Karte für den 86-Pin-Erweiterungsslot im Amiga 2000 (Teil 1)*



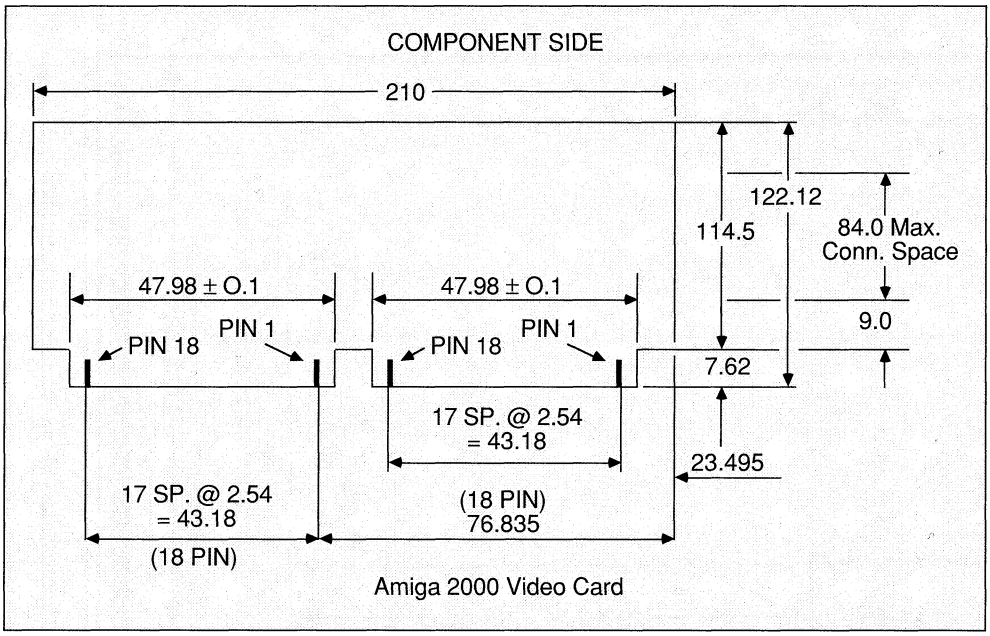
Z A-1: *Diese Abbildung zeigt die Abmaße einer Karte für den 86-Pin-Erweiterungsslot im Amiga 2000 (Teil 2)*



Z A-2: *Hier werden die Abmessungen für eine Brückenkarte, bzw. Amiga 2000-Erweiterungskarte gezeigt (Teil 1)*



ZA-2: *Hier werden die Abmessungen für eine Brückenkarte, bzw. Amiga 2000-Erweiterungskarte gezeigt (Teil 2)*



Z A-3: *Diese Abbildung zeigt die Abmessungen einer Erweiterungskarte für den Amiga 2000 Video-Slot*

Anhang B: Speicherbelegung

000000 – 07FFFF 512 Kbyte-Chip-RAM
080000 – 1FFFFF belegt für System
200000 – 9FFFFF Bereich für Speichererweiterung (Fast-RAM)
A00000 – BEFFFF belegt für System
BFD000 – BFDF00 Registeradressen für *Portbaustein* 8520-B
BFE001 – BFEF01 Registeradressen für Portbaustein 8520-A
C00000 – DFE000 belegt für System
DFF000 – DFFFFF Registeradressen für *Custom-Chips*
E00000 – E7FFFF belegt für System
E80000 – EFFFFF *Expansionslot-Decodierung*
F00000 – F7FFFF belegt für System
F80000 – FFFFFF System-ROM

Anhang C: Die Hardwareregister

Dies ist die komplette Auflistung der Hardwareregister. Über sie kann man direkt die Hardware ansprechen. Zu den angegebenen Adressen muß noch $DFF000 hinzuaddiert werden. Dies ist die Chip-Basisadresse.

Verwendete Abkürzungen:

A – Agnus-Chip
D – Denise-Chip
P – Paula-Chip
R – Read-Register
W – Write-Register
S – Strobe-Register

Die Hardwareregister nach Adressen geordnet (eine ausführliche Funktionsbeschreibung befindet sich bei der alphabetisch geordneten Registerliste im Anschluß an diese Liste):

Register	Adr	Chip	R/W	Funktionsbeschreibung
BLTDDAT	000	A	R	BLiTter-Destination-Data (Dummy-Adresse)
DMACONR	002	AP	R	DMA-CONtrol-Read
VPOSR	004	A	R	Vertical-POSition-Read (höchstes Bit)
VHPOSR	006	A	R	Vertical-and-Horizontal-POSition-Read
DSKDATR	008	P	R	DiSK-DATa-Read (Dummy-Adresse)
JOY0DAT	00A	D	R	JOYport0-DATa
JOY1DAT	00C	D	R	JOYport1-DATa
CLXDAT	00E	D	R	CoLission-Data
ADKCONR	010	P	R	Audio-DisK-CONtrol-Read
POT0DAT	012	P	R	POT0-DATa
POT1DAT	014	P	R	POT1-DATa
POTGOR	016	P	R	POT-data-Read
SERDATR	018	P	R	SERial-DATa-Read
DSKBYTR	01A	P	R	DiSK-dataBYTe-Read
INTENAR	01C	P	R	INTerrupt-ENAble-bits-Read
INTREQR	01E	P	R	INTerrupt-REQuest-bits-Read
DSKPTH	020	A	W	DiSK-PoinTer-High
DSKPTL	022	A	W	DiSK-PoinTer-Low
DSKLEN	024	P	W	DiSK-data-LENgth
DSKDAT	026	P	W	DiSK-dma-DATA-write
REFPTR	028	A	W	REFresh-PoinTeR
VPOSW	02A	A	W	Vertical-POSition-Write (höchstes Bit)

Register	Adr	Chip	R/W	Funktionsbeschreibung
VHPOSW	02C	A	W	Vertical-and-Horizontal-POSition-Write
COPCON	02E	A	W	COProzessor-CONtrol-register
SERDAT	030	P	W	SERial-port-DATa
SERPER	032	P	W	SERial-port-PERiod
POTGO	034	P	W	POT-port-data-write-and-go
JOYTEST	036	D	W	JOYport-TEST
STREQU	038	D	S	STRobe-for-horizontal-syncronisation-with-vertical-blank-and-EQU
STRVBL	03A	D	S	STRobe-for-horizontal-syncronisation-with-Vertical-BLank
STRHOR	03C	DP	S	STRobe-for-HORizontal-syncronisation
STRLONG	03E	D	S	STRobe-for-identification-of-LONG-horizontal-line
BLTCON0	040	A	W	BLiTter-CONtrol-register-0
BLTCON1	042	A	W	BLiTter-CONtrol-register-1
BLTAFWM	044	A	W	BLiTter-First-Word-Mask-for-source-A
BLTALWM	046	A	W	BLiTter-Last-Word-Mask-for-source-A
BLTCPTH	048	A	W	BLiTter-PoinTer-High-to-source-C
BLTCPTL	04A	A	W	BLiTter-PoinTer-Low-to-source-C
BLTBPTH	04C	A	W	BLiTter-PoinTer-High-to-source-B
BLTBPTL	04E	A	W	BLiTter-PoinTer-Low-to-source-B
BLTAPTH	050	A	W	BLiTter-PoinTer-High-to-source-A
BLTAPTL	052	A	W	BLiTter-PoinTer-Low-to-source-A
BLTDPTH	054	A	W	BLiTter-PoinTer-High-to-destination-D
BLTDPTL	056	A	W	BLiTter-PoinTer-Low-to-destination-D
BLTSIZE	058	A	W	BLiTter-start-and-SIZE
	05A			
	05C			
	05E			
BLTCMOD	060	A	W	BLiTter-MODulo-for-source-C
BLTBMOD	062	A	W	BLiTter-MODulo-for-source-B
BLTAMOD	064	A	W	BLiTter-MODulo-for-source-A
BLTDMOD	066	A	W	BLiTter-MODulo-for-destination-D
	068			
	06A			
	06C			
	06E			
BLTCDAT	070	A	W	BLiTter-source-C-DATa-register
BLTBDAT	072	A	W	BLiTter-source-B-DATa-register

Register	Adr	Chip	R/W	Funktionsbeschreibung
BLTADAT	074	A	W	BLiTter-source-A-DATa-register
	076			
	078			
	07A			
	07C			
DSKSYNC	07E	P	R	DiSK-SYNCronisation-pattern-register
COP1LCH	080	A	W	COPper-first-LoCation-High
COP1LCL	082	A	W	COPper-first-LoCation-Low
COP2LCH	084	A	W	COPper-second-LoCation-High
COP2LCL	086	A	W	COPper-second-LoCation-Low
COPJMP1	088	A	S	restart-COPper-at-first-location
COPJMP2	08A	A	S	restart-COPper-at-second-location
COPINS	08C	A	W	COPper-INStruction-fetch-identify
DIWSTRT	08E	A	W	DIsplay-Window-STaRT
DIWSTOP	090	A	W	DIsplay-Window-STOP
DDFSTRT	092	A	W	Display-bitplane-Data-Fetch-STaRT
DDFSTOP	094	A	W	Display-bitplane-Data-Fetch-STOP
DMACON	096	ADP	W	DMA-CONtrol-register
CLXCON	098	D	W	CoLlision-CONtrol
INTENA	09A	P	W	INTerrupt-ENAble-bits
INTREQ	09C	P	W	INTerrupt-REQuest-bits
ADKCON	09E	P	W	Audio-and-DisK-CONtrol
AUD0LCH	0A0	A	W	AUDio0-LoCation-High
AUD0LCL	0A2	A	W	AUDio0-LoCation-Low
AUD0LEN	0A4	P	W	AUDio0-data-LENgth
AUD0PER	0A6	P	W	AUDio0-PERiod
AUD0VOL	0A8	P	W	AUDio0-VOLume
AUD0DAT	0AA	P	W	AUDio0-DATa-register
	0AC			
	OAE			
AUD1LCH	0B0	A	W	AUDio1-LoCation-High
AUD1LCL	0B2	A	W	AUDio1-LoCation-Low
AUD1LEN	0B4	P	W	AUDio1-data-LENgth
AUD1PER	0B6	P	W	AUDio1-PERiod
AUD1VOL	0B8	P	W	AUDio1-VOLume
AUD1DAT	0BA	P	W	AUDio1-DATa-register
	0BC			
	0BE			
AUD2LCH	0C0	A	W	AUDio2-LoCation

Register	Adr	Chip	R/W	Funktionsbeschreibung
AUD2-LCL	0C2	A	W	AUDio2-LoCation-Low
AUD2LEN	0C4	P	W	AUD2-data-LENgth
AUD2PER	0C6	P	W	AUDio2-PERiod
AUD2VOL	0C8	P	W	AUDio2-VOLume
AUD2DAT	0CA	P	W	AUDio2-DATa-register
	0CC			
	0CE			
AUD3LCH	0D0	A	W	AUDio3-LoCation-High
AUD3LCL	0D2	A	W	AUDio3-LoCation-Low
AUD3LEN	0D4	P	W	AUDio3-data-LENgth
AUD3PER	0D6	P	W	AUDio3-PERiod
AUD3VOL	0D8	P	W	AUDio3-VOLume
AUD3DAT	0DA	P	W	AUDio3-DATa-register
	0DC			
	0DE			
BPL1PTH	0E0	A	W	Bit-PLane1-PoinTer-High
BPL1PTL	0E2	A	W	Bit-PLane1-PoinTer-Low
BPL2PTH	0E4	A	W	Bit-PLane2-PoinTer-High
BPL2PTL	0E6	A	W	Bit-PLane2-PoinTer-Low
BPL3PTH	0E8	A	W	Bit-PLane3-PoinTer-High
BPL3PTL	0EA	A	W	Bit-PLane3-PoinTer-Low
BPL4PTH	0EC	A	W	Bit-PLane4-PoinTer-High
BPL4PTL	0EE	A	W	Bit-PLane4-PoinTer-Low
BPL5PTH	0F0	A	W	Bit-PLane5-PoinTer-High
BPL5PTL	0F2	A	W	Bit-PLane5-PoinTer-Low
BPL6PTH	0F4	A	W	Bit-PLane6-PoinTer-High
BPL6PTL	0F6	A	W	Bit-PLane6-PoinTer-Low
	0F8			
	0FA			
	0FC			
	0FE			
BPLCON0	100	AD	W	Bit-PLane-CONtrol-register-0
BPLCON1	102	D	W	Bit-PLane-CONtrol-register-1
BPLCON2	104	D	W	Bit-PLane-CONtrol-register-2
	106			
BPL1MOD	108	A	W	Bit-PLane-MODulo-for-odd-planes (1,3,5)
BPL2MOD	10A	A	W	Bit-PLane-MODulo-for-even-planes (2,4,6)
	10C			
	10E			
BPL1DAT	110	D	W	Bit-PLane-1-DATa

Register	Adr	Chip	R/W	Funktionsbeschreibung
BPL2DAT	112	D	W	Bit-PLane-2-DATa
BPL3DAT	114	D	W	Bit-PLane-3-DATa
AUD2LCH	0C0	A	W	AUDio2-LoCation-High
AUD2LCL	0C2	A	W	AUDio2LoCation-Low
AUD2LEN	0C4	P	W	AUDio2-data-LENgth
BPL4DAT	116	D	W	Bit-PLane-4-DATa
BPL5DAT	118	D	W	Bit-PLane-5-DATa
BPL6DAT	11A	D	W	Bit-PLane-6-DATa
	11C			
	11E			
SPR0PTH	120	A	W	SPRite0-PoinTer-High
SPR0PTL	122	A	W	SPRite0-PoinTer-Low
SPR1PTH	124	A	W	SPRite1-PoinTer-High
SPR1PTL	126	A	W	SPRite1-PoinTer-Low
SPR2PTH	128	A	W	SPRite2-PoinTer-High
SPR2PTL	12A	A	W	SPRite2-PoinTer-Low
SPR3PTH	12C	A	W	SPRite3-PoinTer-High
SPR3PTL	12E	A	W	SPRite3-PoinTer-Low
SPR4PTH	130	A	W	SPRite4-PoinTer-High
SPR4PTL	132	A	W	SPRite4-PoinTer-Low
SPR5PTH	134	A	W	SPRite5-PoinTer-High
SPR5PTL	136	A	W	SPRite5-PoinTer-Low
SPR6PTH	138	A	W	SPRite6-PoinTer-High
SPR6PTL	13A	A	W	SPRite6-PoinTer-Low
SPR7PTH	13C	A	W	SPRite7-PoinTer-High
SPR7PTL	13E	A	W	SPRite7-PoinTer-Low
SPR0POS	140	AD	W	SPRite0-vertical-and-horizontal-start-POSition
SPR0CTL	142	AD	W	SPRite0-vertical-stop-position-and-ConTroL-data
SPR0DATA	144	D	W	SPRite0-image-DATa-register-A
SPR0DATB	146	D	W	SPRite0-image-DATa-register-B
SPR1POS	148	AD	W	SPRite1-vertical-and-horizontal-start-POSition
SPR1CTL	14A	AD	W	SPRite1-vertical-stop-position-and-ConTroL-data
SPR1DATA	14C	D	W	SPRite1-image-DATa-register-A
SPR1DATB	14E	D	W	SPRite1-image-DATa-register-B
SPR2POS	150	AD	W	SPRite2-vertical-and-horizontal-start-POSition
SPR2CTL	152	AD	W	SPRite2-vertical-stop-position-and-ConTroL-data
SPR2DATA	154	D	W	SPRite2-image-DATa-register-A
SPR2DATB	156	D	W	SPRite2-image-DATa-register-B
SPR3POS	158	AD	W	SPRite3-vertical-and-horizontal-start-POSition

Register	Adr	Chip	R/W	Funktionsbeschreibung
SPR3CTL	15A	AD	W	SPRite3-vertical-stop-position-and-ConTroL-data
SPR3DATA	15C	D	W	SPRite3-image-DATa-register-A
SPR3DATB	15E	D	W	SPRite3-image-DATa-register-B
SPR4POS	160	AD	W	SPRite4-vertical-and-horizontal-start-POSition
SPR4CTL	162	AD	W	SPRite4-vertical-stop-position-and-ConTroL-data
SPR4DATA	164	D	W	SPRite4-image-DATa-register-A
SPR4DATB	166	D	W	SPRite4-image-DATa-register-B
SPR4POS	160	AD	W	SPRite4-vertical-and-horizontal-start-POSition
SPR4CTL	162	AD	W	SPRite4-vertical-stop-position-and-ConTroL-data
SPR4DATA	164	D	W	SPRite4-image-DATa-register-A
SPR4DATB	166	D	W	SPRite4-image-DATa-register-B
SPR5POS	168	AD	W	SPRite5-vertical-and-horizontal-start-POSition
SPR5CTL	16A	AD	W	SPRite5-vertical-stop-position-and-ConTroL-data
SPR5DATA	16C	D	W	SPRite5-image-DATa-register-A
SPR5DATB	16E	D	W	SPRite5-image-DATa-register-B
SPR6POS	170	AD	W	SPRite6-vertical-and-horizontal-start-POSition
SPR6CTL	172	AD	W	SPRite6-vertical-stop-position-and-ConTroL-data
SPR6DATA	174	D	W	SPRite6-image-DATa-register-A
SPR6DATB	176	D	W	SPRite6-image-DATa-register-B
SPR7POS	178	AD	W	SPRite7-vertical-and-horizontal-start-POSition
SPR7CTL	17A	AD	W	SPRite7-vertical-stop-position-and-ConTroL-data
SPR7DATA	17C	D	W	SPRite7-image-DATa-register-A
SPR7DATB	17E	D	W	SPRite7-image-DATa-register-B
COLOR00	180	D	W	COLOR-register-00
COLOR01	182	D	W	COLOR-register-01
COLOR02	184	D	W	COLOR-register-02
COLOR03	186	D	W	COLOR-register-03
COLOR04	188	D	W	COLOR-register-04
COLOR05	18A	D	W	COLOR-register-05
COLOR06	18C	D	W	COLOR-register-06
COLOR07	18E	D	W	COLOR-register-07
COLOR08	190	D	W	COLOR-register-08
COLOR09	192	D	W	COLOR-register-09
COLOR10	194	D	W	COLOR-register-10
COLOR11	196	D	W	COLOR-register-11
COLOR12	198	D	W	COLOR-register-12
COLOR13	19A	D	W	COLOR-register-13
COLOR14	19C	D	W	COLOR-register-14
COLOR15	19E	D	W	COLOR-register-15
COLOR16	1A0	D	W	COLOR-register-16

Register	Adr	Chip	R/W	Funktionsbeschreibung
COLOR17	1A2	D	W	COLOR-register-17
COLOR18	1A4	D	W	COLOR-register-18
COLOR19	1A6	D	W	COLOR-register-19
COLOR20	1A8	D	W	COLOR-register-20
COLOR21	1AA	D	W	COLOR-register-21
COLOR22	1AC	D	W	COLOR-register-22
COLOR24	1B0	D	W	COLOR-register-24
COLOR23	1AE	D	W	COLOR-register-23
COLOR25	1B2	D	W	COLOR-register-25
COLOR26	1B4	D	W	COLOR-register-26
COLOR27	1B6	D	W	COLOR-register-27
COLOR28	1B8	D	W	COLOR-register-28
COLOR29	1BA	D	W	COLOR-register-29
COLOR30	1BC	D	W	COLOR-register-30
COLOR31	1BE	D	W	COLOR-register-31

Die Hardwareregister alphabetisch geordnet mit ausführlichen Funktionsbeschreibungen:

Register	Adr	Chip	R/W	Funktionsbeschreibung
ADKCON	09E	P	W	Audio-and-DisK-CONtrol
ADKCONR	010	P	R	Audio-DisK-CONtrol-Read

Bit	Funktion	
15	SET/CLR	Set/Clear-Bit (1=Set)
14	PRECOMP	00=kein, 01=140ns
13		10=280ns, 11=560ns
12	MFMPREC	MFM oder BCR-Format (1=MFM)
11	UARTBRK	UART-Break (1=Break)
10	WORDSYNC	Disk-Synchronisation. Wartet, bis gelesener Wert gleich dem Wert in DSKSYNC ist.
09	MSBSYNC	Disk-Synchronisation mit MSB. Wird beispielsweise bei APPLE-Format verwendet.
08	FAST	Clock-Rate: 1=Fast=MFM
07	USE3PN	Ben. Kan. 3 um nichts zu mod.
06	USE2P3	Ben. Kan. 2 um Period Kan. 3 zu mod.
05	USE1P2	Ben. Kan. 1 um Period Kan. 2 zu mod.

Register	Adr	Chip	R/W	Funktionsbezeichnung
		04	USE0P1	Ben. Kan. 0 um Period Kan. 1 zu mod.
		03	USE3VN	Ben. Kan. 3 um nichts zu mod.
		02	USE2V3	Ben. Kan. 2 um Volume Kan. 3 zu mod.
		01	USE1V2	Ben. Kan. 1 um Volume Kan. 2 zu mod.
		00	USE0V1	Ben. Kan. 0 um Volume Kan. 1 zu mod.
AUDxLCH	0x0	A	W	AUDiox-LoCation-High
AUDxLCL	0x2	A	W	AUDiox-LoCation-Low

Diese beiden Register enthalten die Startadresse der Audiodaten für Kanal x. Die 19-Bit-Adresse steht mit 3 Bits im High-Register und mit 16 Bits im Low-Register. Die Adresse muß Word-alligned sein.
Diese beiden Register enthalten die Startadresse der Audiodaten für Kanal x. Die 19-Bit-Adresse steht mit 3 Bits im High-Register und mit 16 Bits im Low-Register. Die Adresse muß Word-alligned sein.

AUDxLEN 0x4 P W AUDiox-data-LENgth

Dieses Register enthält die Länge der Audiodaten für Kanal x in Words.

AUDxPER 0x6 P W AUDiox-PERiod

Dieses Register enthält die DMA-Datenübertragungsrate in Clockzyklen. Der Minimalwert beträgt 124, welcher gleichzeitig die maximale Abspielgeschwindigkeit von 28,86 khz darstellt.

AUDxVOL 0x8 P W AUDiox-VOLume

Enthält die Lautstärke von Kanal x. Der maximale Wert beträgt 64.

Bit	Funktion
15–7	Werden nicht benutzt
06	Ist dieses Bit gesetzt, so wird die Lautstärke auf den maximalen Wert gesetzt, egal, wie die anderen Bits gesetzt sind.
05–0	Geben die Lautstärke von 0 bis 63 an.

AUDxDAT 0xA P W AUDiox-DATa-register

Dieses Register enthält jeweils zwei Byte der Sounddaten, die ausgegeben werden sollen als Zweierkomplement. Im Normalfall schreibt die DMA diese Daten aus dem Speicher in dieses Register. Es kann aber auch durch direktes Ansprechen mit der CPU ein Sound erzeugt werden.

Register	Adr	Chip	R/W	Funktionsbezeichnung
BLTxPTH	0xx	A	W	BLiTter-PoinTer-High-to-x
BLTxPTL	0xx	A	W	BLiTter-PoinTer-Low-to-x

Zeiger auf Blitterdaten für Source oder Destination-Data.

BLTxMOD 0xx A W BLiTter-MODulo-for-x

Dieses Register enthält den Modulowert für die Source-, bzw. Destinationdaten. Ist eine Zeile vom Blitter verarbeitet worden, so wird der Modulowert hinzuaddiert, die neue Zeile beginnt also entsprechend später im Speicher. Dies ist sehr wichtig, wenn der zu bearbeitende Bereich ein Ausschnitt aus einer größeren Bitplane ist.

BLTAFWM	044	A	W	BLiTter-First-Word-Mask-for-source-A
BLTALWM	046	A	W	BLiTter-Last-Word-Mask-for-source-A

Die Bitmuster werden per AND mit dem ersten, bzw. letzten Word einer Zeile aus Source A verknüpft.

BLTxDAT 0xx A W BLiTter-source-x-DATa-register

Dieses Register enthält ein Word von Daten aus Source x. Normalerweise wird es durch die DMA geschrieben, kann aber auch durch die CPU gesetzt werden.

BLTCON0	040	A	W	BLiTter-CONtrol-register-0
BLTCON1	042	A	W	BLiTter-CONtrol-register-1

BLTSIZE 058 A W BLiTter-start-and-SIZE

Dieses Register enthält die Höhe und Breite des Bereiches, der bearbeitet werden soll. Die Bits 6–15 enthalten die Höhe, und Bits 0–5 enthalten die Breite.

BPLxPTH	0xx	A	W	Bit-PLanex-PoinTer-High
BPLxPTL	0xx	A	W	Bit-PLanex-PoinTer-Low

Diese Register enthalten die Startadresse der Bitplane x.

BPLxDAT 11A D W Bit-PLane-x-DATa

Dieses Register enthält jeweils ein Word an Displaydaten, die ausgegeben werden. Normalerweise werden die Daten durch die DMA geschrieben, theoretisch kann dies aber auch durch die CPU geschehen. Bit 15 wird immer zuerst dargestellt, d.h. es erscheint links von den anderen Bits.

BPL1MOD 108 A W Bit-PLane-MODulo-for-odd-planes (1,3,5)

Dieses Register enthält den Modulowert für die ungeraden Bit-Planes. Ist eine Bildschirmzeile komplett dargestellt, so wird zur aktuellen Adresse der Modulowert hinzuaddiert, so daß die Bit-Plane größer als der dargestellte Bildbereich sein kann.

Register	Adr	Chip	R/W	Funktionsbeschreibung
BPL2MOD	10A	A	W	Bit-PLane-MODulo-for-even-planes (2,4,6)

Für dieses Register gilt das oben gesagte. Da es zwei getrennte Register für ungerade und gerade BitPlanes gibt, besteht die Möglichkeit im Dualplayfield-Modus zwei verschiedengroße Playfields darzustellen.

BPLCON0	100	AD	W	Bit-PLane-CONtrol-register-0
BPLCON1	102	D	W	Bit-PLane-CONtrol-register-1
BPLCON2	104	D	W	Bit-PLane-CONtrol-register-2

Diese Register kontrollieren die Bildschirmdarstellung:

Bit	BPLCON0	BPLCON1	BPLCON2
15	HIRES		
14	BPU2		
13	BPU1		
12	BPU0		
11	HOMOD		
10	DUALPF		
09	COLOR		
08	GAUD		
07		PF2H3	
06		PF2H2	PF2PRI
05		PF2H1	PF2P2
04		PF2H0	PF2P1
03	LPEN	PF1H3	PF2P0
02	LACE	PF1H2	PF1P2
01	ERSY	PF1H1	PF1P1
00		PF1H0	PF1P0

Abkürzungen:
HIRES – High-Resolution-Modus
BPU – Anzahl der BitPlanes (000-110 = 0–6)
HOMOD – Hold-and-Modify-Modus
DUALPF – Dual-Playfield-Modus
Die ungeraden BitPlanes werden als PF1 und die geraden als PF2 dargestellt.
COLOR – Composite-Video-Modus
GAUD – Genlock-Audio-Modus
LPEN – Lightpen-Modus
LACE – Interlace-Modus

ERSY – Externe Video-Synchronisation
PF2PRI – PF2 wird vor PF1 dargestellt
PF2P – PF2-Priorität zu Sprites
PF1P – PF1-Priorität zu Sprites
PF2H – PF2-Horizontal-Code
PF1H – PF1-Horizontal-Code

CLXCON 098 D W CoLlision-CONtrol

Dieses Register setzt die Objekte, die auf Kollision getestet werden sollen:

Bit	Objekt
15	Sprite 6 + 7
14	Sprite 4 + 5
13	Sprite 2 + 3
12	Sprite 0 + 1
11	BitPlane 6
10	BitPlane 5
09	BitPlane 4
08	BitPlane 3
07	BitPlane 2
06	BitPlane 1
05	Kollisions-Code für BitPlane6
04	Kollisions-Code für BitPlane5
03	Kollisions-Code für BitPlane4
02	Kollisions-Code für BitPlane3
01	Kollisions-Code für BitPlane2
00	Kollisions-Code für BitPlane1

CLXDAT 00E D R CoLission-Data

Dieses Register zeigt die registrierten Kollisionen an:

Bit	registrierte Kollision
15	nicht benutzt
14	Sprite 4 oder 5 zu Sprite 6 oder 7
13	Sprite 2 oder 3 zu Sprite 6 oder 7
12	Sprite 2 oder 3 zu Sprite 4 oder 5
11	Sprite 0 oder 1 zu Sprite 6 oder 7
10	Sprite 0 oder 1 zu Sprite 4 oder 5
09	Sprite 0 oder 1 zu Sprite 2 oder 3
08	Playfield 2 zu Sprite 6 oder 7
07	Playfield 2 zu Sprite 4 oder 5

06 Playfield 2 zu Sprite 2 oder 3
05 Playfield 2 zu Sprite 0 oder 1
04 Playfield 1 zu Sprite 6 oder 7
03 Playfield 1 zu Sprite 4 oder 5
02 Playfield 1 zu Sprite 2 oder 3
01 Playfield 1 zu Sprite 0 oder 1
00 Playfield 1 zu Playfield 2

COLORxx 1xx D W COLOR-register-xx
Diese Register enthalten die aktuellen Farben, die dargestellt werden. Bits 0 bis 3 enthalten den Blauanteil, Bits 4 bis 7 den Grünanteil und die Bits 8 bis 11 den Rotanteil der Farbe.

COPCON 02E A W COProzessor-CONtrol-register
Wenn Bit 1 dieses Registers gesetzt ist, kann der Copper auch die Blitterhardware (-register) ansprechen.

COPJMP1 088 A S restart-COPper-at-first-location
COPJMP2 08A A S restart-COPper-at-second-location
Wird eine dieser Adressen angesprochen, so wird der Copper mit einem Zeiger auf eine neue Copperliste neu gestartet. Wird COPJMP1 angesprochen, so wird die Adresse im Register COP1LCH/L verwendet.

COP1LCH 080 A W COPper-first-LoCation-High
COP1LCL 082 A W COPper-first-LoCation-Low
COP2LCH 084 A W COPper-second-LoCation-High
COP2LCL 086 A W COPper-second-LoCation-Low
Diese Register enthalten die Adresse der Copperliste, die mit COPJMP1/2 angesprochen werden kann.

COPINS 08C A W COPper-INStruction-fetch-identify
Dies ist eine Dummy-Adresse des Coppers.

DIWSTRT 08E A W DIsplay-Window-STaRT
DIWSTOP 090 A W DIsplay-Window-STOP
Diese Register legen die Displaygröße und Position fest. DIWSTRT legt die vertikale und horizontale Startposition und DIWSTOP die Stopposition fest. Die Bits 0 bis 7 enthalten die horizontale und Bits 8 bis 15 die vertikale Position. Stellt man sich zu diesen je 8 Bits noch ein neuntes vor, so ist dieses bei DIWSTRT gleich 0 und bei DIWSTOP gleich 1. D.h. DIWSTRT ist vertikal auf die oberen 2/3 und horizontal auf die linken 3/4 des Screens beschränkt und DIWSTOP vertikal auf die untere Hälfte und horizontal auf das rechte Viertel.

DDFSTRT 092 A W Display-bitplane-Data-Fetch-STaRT
DDFSTOP 094 A W Display-bitplane-Data-Fetch-STOP

Diese Register kontrollieren das horizontale Timing der Display-DMA im Zugriff auf die Daten.

DMACON 096 ADP W DMA-CONtrol-register

DMACONR 002 AP R DMA-CONtrol-Read

Diese Register kontrollieren alle DMA-Kanäle. Im Lesezugriff kann noch der Blitterzustand abgefragt werden. Bit Funktion 15 Ist dieses Bit auf 1 gesetzt, so werden alle Kanäle, deren korespondierende Bits gesetzt sind, eingeschaltet. Die nicht gesetzten Bits werden nicht geändert. Ist dieses Bit auf 0 gesetzt, so werden alle Kanäle, deren Bits auf 1 stehen, abgeschaltet.

14	Blitter beschäftigt (Nur Lesezugriff)
13	Blitter logisch zurückgesetzt (Nur Lesezugriff)
12	keine Funktion
11	keine Funktion
10	Blitter hat Priorität vor CPU
09	Aktiviert alle folgenden DMA-Kanäle
08	BitPlane-DMA
07	Copper-DMA
06	Blitter-DMA
05	Sprite-DMA
04	Disk-DMA
03	Audiokanal0-DMA
02	Audiokanal1-DMA
01	Audiokanal2-DMA
00	Audiokanal3-DMA

DSKPTH 020 A W DiSK-PoinTer-High
DSKPTL 022 A W DiSK-PoinTer-Low

Dieses Registerpaar enthält die Adresse des Datenpuffers für die Diskdaten.

DSKLEN 024 P W DiSK-data-LENgth

Die Bits 0 bis 13 enthalten die Anzahl der Words, die geschrieben/gelesen werden soll. Ist Bit 14 auf 1 gesetzt, so wird geschrieben, ansonsten wird gelesen. Ist Bit 15 gesetzt, so wird die Disk-DMA eingeschaltet.

DSKDAT 026 P W DiSK-dma-DATA-write
DSKDATR 008 P R DiSK-DATa-Read (Dummy-Adresse)

Diese Register werden benötigt, um Daten von und zur Diskette zu übertragen.

Register	Adr	Chip	R/W	Funktionsbeschreibung
DSKBYTR	01A	P	R	DiSK-dataBYTe-Read

Dieses Register korrespondiert direkt mit der Floppyhardware. Im Lesezugriff wird jedes gelesene Byte in die unteren 8 Bits eingeschrieben und Bit 15 auf 1 gesetzt.

DSKSYNC	07E	P	R	DiSK-SYNChronisation-pattern-register

Ist Bit 10 im ADKCON-Register gesetzt, wird erst von der Diskette in das RAM gelesen, wenn ein gelesenes Word mit dem Word in diesem Register übereinstimmt (Synchronisation).

INTREQ	09C	P	W	INTerrupt-REQuest-bits
INTREQR	01E	P	R	INTerrupt-REQuest-bits-Read

Diese Register enthalten Interrupt-Request-Bits.
Die Bits stimmen mit den unter INTENA/R aufgelisteten überein.

INTENA	09A	P	W	INTerrupt-ENAble-bits
INTENAR	01C	P	R	INTerrupt-ENAble-bits-Read

Diese Register enthalten die eingeschalteten Interrupts:

Bit	Funktion/Interrupt
15	SET/CLR siehe unter DMACON
14	Master-Interrupt (kein Request)
13	Externer Interrupt
12	DISKSYNC-Word stimmt mit gelesenem Word überein (Synchronisation).
11	Empfangspuffer für serielle Schnittstelle ist voll
10	Audiokanal 3 hat Sounddaten abgespielt
09	Audiokanal 2 hat Sounddaten abgespielt
08	Audiokanal 1 hat Sounddaten abgespielt
07	Audiokanal 0 hat Sounddaten abgespielt
06	Blitterfunktion beendet
05	Neuer Bildaufbau beginnt
04	Copperinterrupt
03	E/A-Ports und Timer
02	Reserviert für Softwareinterrupts
01	Diskblock komplett eingelesen
00	Puffer für serielle Schnittstelle ist leer

JOY0DAT	00A	D	R	JOYport0-DATa
JOY1DAT	00C	D	R	JOYport1-DATa

Register	Adr	Chip	R/W	Funktionsbeschreibung

JOY0DAT repräsentiert den linken Joyport und 1 den rechten. Alle Bits sind »low-active«. Folgendermaßen kann man einen Joystick abfragen:

Joystick vorne = Bit 9 XOR Bit 8
links = Bit 9
hinten = Bit 1 XOR Bit 0
rechts = Bit 1

JOYTEST 036 D W JOYport-TEST

Schreibt den Wert in die Register JOY1/2DAT. Bis auf die Bits 0, 1, 8 und 9 müssen dort alle Bits mit den hier geschriebenen übereinstimmen.

POT0DAT 012 P R POT0-DATa
POT1DAT 014 P R POT1-DATa

Diese Register enthalten die Pot(-entiometer)-Werte der zwei Joyports.

POTGO 034 P W POT-port-data-write-and-go
POTGOR 016 P R POT-data-Read

Diese Register kontrollieren einen bidirektionalen 4-Bit-IO-Port, der die Pot-Pins verwendet.

Bit	Funktion
15	Ausgabe für Pot-Pin 36 einschalten
14	Datenbit Pin 36
13	Ausgabe für Pot-Pin 35 einschalten
12	Datenbit Pin 35
11	Ausgabe für Pot-Pin 33 einschalten
10	Datenbit Pin 33
09	Ausgabe für Pot-Pin 32 einschalten
08	Datenbit Pin 32
7–1	Reserviert für Custom-Chips
00	Pots zurücksetzen

REFPTR 028 A W REFresh-PoinTeR

Dieses Register wird für das dynamische RAM-Refresh verwendet.

SERDAT 030 P W SERial-port-DATa

Über dieses Register läßt sich auf die serielle Schnittstelle zugreifen. Bits 0 bis 8 enthalten die Datenbits und 9 ist das Stopbit.

Register	Adr	Chip	R/W	Funktionsbeschreibung
SERDATR	018	P	R	SERial-DATa-Read

Dieses Register wird immer dann neu gesetzt, wenn über die serielle Schnittstelle ein kompletter Datensatz, also bis zum Stopbit, empfangen wurde.

Bit	Funktion
15	Serieller Port ist überlaufen
14	Empfangspuffer voll
13	Übertragungspuffer leer
12	Wandlungspuffer leer
11	UART-Data empfangen
10	nicht benutzt
09	Stopbit
08	Je nach Übertragungsart Stop- oder Datenbit. Ist in SERPER Bit 15 gesetzt, so ist es ein Datenbit.
7–0	Datenbits

SERPER	032	P	W	SERial-port-PERiod

Dieses Register setzt die Baudrate und die Datenlänge. Ist Bit 15 gesetzt, so werden 9 Datenbits verwendet, ansonsten nur 8. Die Bits 0 bis 14 legen die Baudrate fest. Wird hier der Wert X gesetzt, so resultiert daraus die Baudrate 1/((X+1)*0.2794ms).

SPRxPTH	xxx	A	W	SPRitex-PoinTer-High
SPRxPTL	xxx	A	W	SPRitex-PoinTer-Low

Diese Register enthalten die Startadressen der Spriteimagedaten.

SPRxPOS	xxx	AD	W	SPRitex-vertical-and-horizontal-start-POSition
SPRxCTL	xxx	AD	W	SPRitex-vertical-stop-position-and-ConTroL-data

Die Bits 8 bis 15 in SPRxPOS und Bit 2 in SPRxCTL als Highbit setzen die vertikale Startposition des Sprites. Bits 0 bis 7 in SPRxPOS und Bit 0 in SPRxCTL als Lowbit setzen die horizontale Startposition des Sprites. Die Bits 8 bis 15 im SPRxCTL und Bit 1 im gleichen Register als Highbit setzen die vertikale Endposition des Sprites.

SPRxDATA	xxx	D	W	SPRitex-image-DATa-register-A
SPRxDATB	xxx	D	W	SPRitex-image-DATa-register-B

Diese Register werden verwendet um die Sprites darzustellen. Normalerweise werden die Daten von der DMA in diese Register geschrieben, theoretisch ist dies aber auch mit der CPU möglich.

Register	Adr	Chip	R/W	Funktionsbeschreibung
STREQU	038	D	S	STRobe-for-horizontal-synchronisation-with-vertical-blank-and-EQU
STRVBL	03A	D	S	STRobe-for-horizontal-synchronisation-with-Vertical-BLank
STRHOR	03C	DP	S	STRobe-for-HORizontal-synchronisation-
STRLONG	03E	D	S	STRobe-for-identification-of-LONG-horizontal-line

Strobes zur Synchronisation der Videoausgabe.

VPOSR	004	A	R	Vertical-POSition-Read (höchstes Bit)
VPOSW	02A	A	W	Vertical-POSition-Write (höchstes Bit)

Diese Register enthalten das höchste Bit der vertikalen Position des Rasterstrahls.

VHPOSR	006	A	R	Vertical-and-Horizontal-POSition-Read
VHPOSW	02C	A	W	Vertical-and-Horizontal-POSition-Write

Diese Register enthalten die vertikale und horizontale Position des Rasterstrahls. Die Bits 0 bis 7 enthalten die horizontale und 8 bis 15 die vertikale Position.

Anhang D: Registeradressen der Portbausteine

Der Amiga verfügt über zwei Portbausteine, den 8520-A und 8520- B. Der 8520-A wird auch mit CIAA und der 8520-B mit CIAB bezeichnet. Diese Bausteine lassen sich über folgende Speicheradressen ansprechen, die Portregister. Diese Register sind jeweils 8 Bits lang.

Register von CIAA und CIAB:

CIAA	*CIAB*	Reg.	Funktion
BFE001	BFD000	PRA	Peripherie-Datenregister A
BFE101	BFD100	PRB	Peripherie-Datenregister B
BFE201	BFD200	DDRB	Datenrichtungs-Register A für PRA
BFE301	BFD300	DDRA	Datenrichtungs-Register B für PRA
BFE401	BFD400	TALO	TIMER A Low-Register
BFE501	BFD500	TAHI	TIMER A High-Register
BFE601	BFD600	TBLO	TIMER B Low-Register
BFE701	BFD700	TBHI	TIMER B High-Register
BFE801	BFD800		Event LSB
BFE901	BFD900		Event 8 – 15
BFEA01	BFDA00		Event MSB
BFEB01	BFDB00		Keine Funktion

Register von CIAA und CIAB:

BFEC01	BFDC00	SDR	Datenregister für serielle Schnittstelle
BFED01	BFDD00	ICR	Interrupt-Kontroll-Register
BFEE01	BFDE00	CRA	Kontroll-Register A
BFEF01	BFDF00	CRB	Kontroll-Register B

Die Peripherie-Datenregister spielen eine zentrale Rolle, da fast alle IO-Operationen der Peripherie über diese Register abgewickelt werden. Hier diese Register im Detail:

Port	Adr.	Reg.
CIAA	BFE001	PRAA
Bit 7:	FIR1	Ist rechte Taste gedrückt
Bit 6:	FIR0	Ist linke Taste gedrückt
Bit 5:	RDY	Ist Disklaufwerk betriebsbereit
Bit 4:	TK0	Ist Head vom Disklaufwerk über Track 0
Bit 3:	WPRO	Ist Disk schreibgeschützt
Bit 2:	CHNG	Wurde Diskette gewechselt
Bit 1:	LED	LED-Status (1=hell, 0=dunkel)
Bit 0:	OVL	

Port	Adr.	Reg.
CIAA	BFE101	PRBA

Dieses Register ist direkt an die Datenleitungen des Parallell-Ports angeschlossen.

Port	Adr.	Reg.
CIAB	BFD000	PRAB
Bit 7:	DTR	
Bit 6:	RTS	
Bit 5:	CD	
Bit 4:	CTS	
Bit 3:	DSR	
Bit 2:	SEL	
Bit 1:	POUT	
Bit 0:	BUSY	

Port	Adr.	Reg.
CIAB	BFD000	PRAB
Bit 7:	MTR	
Bit 6:	SEL3	
Bit 5:	SEL2	
Bit 4:	SEL1	
Bit 3:	SEL0	
Bit 2:	SIDE	
Bit 1:	DIR	
Bit 0:	STEP	

Anhang E: Einsprungadressen der Bibliotheksfunktionen

Im folgenden werden sämtliche Bibliotheksfunktionen aufgeführt. In Klammern werden dann die Parameter angegeben, die mit übergeben werden müssen. Für die Assemblerprogrammierer sind noch die zugehörigen Register aufgeführt, die diese Parameter repräsentieren.

Um die Funktionen aufrufen zu können, muß zuvor die jeweilige Library geöffnet worden sein, und zu dem Index der Funktion die Librarybase hinzuaddiert werden. Um beispielsweise die Funktion Supervisor() der Execlibrary aufzurufen, geht man folgendermaßen vor:

```
move.l #$4,a6  ;Execbase zuweisen
jsr    -30(a6) ;Supervisor aufrufen
```

Die Tabelle ist folgendermaßen aufgebaut:

clist.library (Libraryname)

-$001E	-30	InitCLPool (CLPool,Size)(A0,D0)
Einsprungadressen		Funktionsname mit Parametern und Registern hex. und dez.

clist.library

-$001E	-30	InitCLPool (CLPool,Size)(A0,D0)
-$0024	-36	AllocCList (CLPool)(A1)
-$002A	-42	FreeCList (CList)(A0)
-$0030	-48	FlushCList (CList)(A0
-$0036	-54	SizeCList (CList)(A0)
-$003C	-60	PutCLChar (CList,Byte)(A0,D0)
-$0042	-66	GetCLChar (CList)(A0)
-$0048	-72	UnGetCLChar (CList,Byte)(A0,D0)
-$004E	-78	UnPutCLChar (CList)(A0)
-$0054	-84	PutCLWord (CList,Word)(A0,D0)
-$005A	-90	GetCLWord (CList)(A0)
-$0060	-96	UnGetCLWord (CList,Word)(A0,D0)
-$0066	-102	UnPutCLWord (CList)(A0)
-$006C	-108	PutCLBuf (CList,Buffer,Length)(A0,A1,D1)
-$0072	-114	GetCLBuf (CList,Buffer,MaxLength)(A0,A1,D1)
-$0078	-120	MarkCList (CList,Offset)(A0,D0)
-$007E	-126	IncrCLMark (CList)(A0)
-$0084	-132	PeekCLMark (CList)(A0)
-$008A	-138	SplitCList (CList)(A0)
-$0090	-144	CopyCList (CList)(A0)

-$0096	-150	SubCList (CList,Index,Length)(A0,D0,D1)
-$009C	-156	ConcatCList (SourceCList,DestCList)(A0,A1)

console.library

-$002A	-42	CDInputHandler (Events,Device)(A0,A1)
-$0030	-48	RawKeyConvert (Events,Buffer,Length,KeyMap) (A0,A1,D1,A2)

diskfont.library

-$001E	-30	OpenDiskFont (TextAttr)(A0)
-$0024	-36	AvailFonts (Buffer,BufBytes,Flags)(A0,D0,D1)

dos.library

-$001E	-30	Open (Name,AccessMode)(D1,D2)
-$0024	-36	Close (File)(D1)
-$002A	-42	Read (File,Buffer,Length)(D1,D2,D3)
-$0030	-48	Write (File,Buffer,Length)(D1,D2,D3)
-$0036	-54	Input ()
-$003C	-60	Output ()
-$0042	-66	Seek (File,Position,Offset)(D1,D2,D3)
-$0048	-72	DeleteFile (Name)(D1)
-$004E	-78	Rename (OldName,NewName)(D1,D2)
-$0054	-84	Lock (Name,Type)(D1,D2)
-$005A	-90	UnLock (Lock)(D1)
-$0060	-96	DupLock (Lock)(D1)
-$0066	-102	Examine (Lock,FileInfoBlock)(D1,D2)
-$006C	-108	ExNext (Lock,FileInfoBlock)(D1,D2)
-$0072	-114	Info (Lock,FileInfoBlock)(D1,D2)
-$0078	-120	CreateDir (Name)(D1)
-$007E	-126	CurrentDir (Lock)(D1)
-$0084	-132	IoErr ()
-$008A	-138	CreateProc (Name,Pri,SegList,StackSize) (D1,D2,D3,D4)
-$0090	-144	Exit (ReturnCode)(D1)
-$0096	-150	LoadSeg (FileName)(D1)
-$009C	-156	UnLoadSeg (Segment)(D1)
-$00A2	-162	GetPacket (Wait)(D1)
-$00A8	-168	Queue (Packet)(D1)
-$00AE	-174	DeviceProc (Name)(D1)
-$00B4	-180	SetComment (Name,Comment)(D1,D2)

-$00BA	-186	SetProtection (Name,Mask)(D1,D2)
-$00C0	-192	DateStamp (Date)(D1)
-$00C6	-198	Delay (Timeout)(D1)
-$00CC	-204	WaitForChar (File,Timeout)(D1,D2)
-$00D2	-210	ParentDir (Lock)(D1)
-$00D8	-216	IsInteractive (File)(D1)
-$00DE	-222	Execute (String,File,File)(D1,D2,D3)

exec.library

-$001E	-30	Supervisor ()
-$0024	-36	ExitIntr ()
-$002A	-42	Schedule ()
-$0030	-48	Reschedule ()
-$0036	-54	Switch ()
-$003C	-60	Dispatch ()
-$0042	-66	Exception ()
-$0048	-72	InitCode (StartClass,Version)(D0,D1)
-$004E	-78	InitStruct (InitTable,Memory,Size)(A1,A2,D0)
-$0054	-84	MakeLibrary (FuncInit,StructInit,LibInit,DataSize, CodeSize)(A0,A1,A2,D0,D1)
-$005A	-90	MakeFunctions (Target,FunctionArray,FuncDispBase) (A0,A1,A2)
-$0060	-96	FindResident (Name)(A1)
-$0066	-102	InitResident (Resident,SegList)(A1,D1)
-$006C	-108	Alert (AlertNum,Parameters)(D7,A5)
-$0072	-114	Debug ()
-$0078	-120	Disable ()
-$007E	-126	Enable ()
-$0084	-132	Forbid ()
-$008A	-138	Permit ()
-$0090	-144	SetSR (NewSR,Mask)(D0,D1)
-$0096	-150	SuperState ()
-$009C	-156	UserState (SysStack)(D0)
-$00A2	-162	SetIntVector (IntNumber,Interrupt)(D0,A1)
-$00A8	-168	AddIntServer (IntNumber,Interrupt)(D0,A1)
-$00AE	-174	RemIntServer (IntNumber,Interrupt)(D0,A1)
-$00B4	-180	Cause (Interrupt)(A1)
-$00BA	-186	Allocate (FreeList,ByteSize)(A0,D0)
-$00C0	-192	Deallocate (FreeList,MemoryBlock,ByteSize) (A0,A1,D0)
-$00C6	-198	AllocMem (ByteSize,Requirements)(D0,D1)
-$00CC	-204	AllocAbs (ByteSize,Location)(D0,A1)

-$00D2	-210	FreeMem (MemoryBlock,ByteSize)(A1,D0)
-$00D8	-216	AvailMem (requirements)(D1)
-$00DE	-222	AllocEntry (Entry)(A0)
-$00E4	-228	FreeEntry (Entry)(A0)
-$00EA	-234	Insert (List,Node,Pred)(A0,A1,A2)
-$00F0	-240	AddHead (List,Node)(A0,A1)
-$00F6	-246	AddTail (List,Node)(A0,A1)
-$00FC	-252	Remove (Node)(A1)
-$0102	-258	RemHead (List)(A0)
-$0108	-264	RemTail (List)(A0)
-$010E	-270	Enqueue (List,Node)(A0,A1)
-$0114	-276	FindName (List,Name)(A0,A1)
-$011A	-282	AddTask (Task,InitPC,FinalPC)(A1,A2,A3)
-$0120	-288	RemTask (Task)(A1)
-$0126	-294	FindTask (Name)(A1)
-$012C	-300	SetTaskPri (Task,Priority)(A1,D0)
-$0132	-306	SetSignal (NewSignals,SignalSet)(D0,D1)
-$0138	-312	SetExcept (NewSignals,SignalSet)(D0,D1)
-$013E	-318	Wait (SignalSet)(D0)
-$0144	-324	Signal (Task,SignalSet)(A1,D0)
-$014A	-330	AllocSignal (SignalNum)(D0)
-$0150	-336	FreeSignal (SignalNum)(D0)
-$0156	-342	llocTrap (TrapNum)(D0)
-$015C	-348	FreeTrap (TrapNum)(D0)
-$0162	-354	AddPort (Port)(A1)
-$0168	-360	RemPort (Port)(A1)
-$016E	-366	PutMsg (Port,Message)(A0,A1)
-$0174	-372	GetMsg (Port)(A0)
-$017A	-378	ReplyMsg (Message)(A1)
-$0180	-384	WaitPort (Port)(A0)
-$0186	-390	FindPort (Name)(A1)
-$018C	-396	AddLibrary (Library)(A1)
-$0192	-402	RemLibrary (Library)(A1)
-$0198	-408	OldOpenLibrary (LibName)(A1)
-$019E	-414	CloseLibrary (Library)(A1)
-$01A4	-420	SetFunction (Library,FuncOffset,FuncEntry) (A1,A0,D0)
-$01AA	-426	SumLibrary (Library)(A1)
-$01B0	-432	AddDevice (Device)(A1)
-$01B6	-438	RemDevice (Device)(A1)
-$01BC	-444	OpenDevice (DevName,Unit,IORequest,Flags) (A0,D0,A1,D1)

-$01C2	-450	CloseDevice (IORequest)(A1)
-$01C8	-456	DoIO (IORequest)(A1)
-$01CE	-462	SendIO (IORequest)(A1)
-$01D4	-468	CheckIO (IORequest)(A1)
-$01DA	-474	WaitIO (IORequest)(A1)
-$01E0	-480	AbortIO (IORequest)(A1)
-$01E6	-486	AddResource (Resource)(A1)
-$01EC	-492	RemResource (Resource)(A1)
-$01F2	-498	OpenResource (ResName,Version)(A1,D0)
-$01F8	-504	RawIOInit ()
-$01FE	-510	RawMayGetChar ()
-$0204	-516	RawPutChar (Char)(D0)
-$020A	-522	RawDoFmt ()
-$0210	-528	GetCC ()
-$0216	-534	TypeOfMem (Address)(A1)
-$021C	-540	Procedure (Semaport,BidMsg)(A0,A1)
-$0222	-546	Vacate (Semaport)(A0)
-$0228	-552	OpenLibrary (LibName,Version)(A1,D0)

graphics.library

-$001E	-30	BltBitMap (SrcBm,SrcX,SrcY,DestBm,DestX,DestY, SizeX,SizeY,Minterm,Mask,TempA) (A0,D0,D1,A1,D2,D3,D4,D5,D6,D7,A2)
-$0024	-36	BltTemplate (Source,SrcX,SrcMod,DestRP,DestX, DestY,SizeX,SizeY)(A0,D0,D1,A1,D2,D3,D4,D5)
-$002A	-42	ClearEOL (RastPort)(A1)
-$0030	-48	ClearScreen (RastPort)(A1)
-$0036	-54	TextLength (RastPort,String,Count)(A1,A0,D0)
-$003C	-60	Text (RastPort,String,Count)(A1,A0,D0)
-$0042	-66	SetFont (RastPortID,TextFont)(A1,A0)
-$0048	-72	OpenFont (TextAttr)(A0)
-$004E	-78	CloseFont (TextFont)(A1)
-$0054	-84	AskSoftStyle (RastPort)(A1)
-$005A	-90	SetSoftStyle (RastPort,Style,Enable)(A1,D0,D1)
-$0060	-96	AddBob (Bob,RastPort)(A0,A1)
-$0066	-102	AddVSprite (VSprite,RastPort)(A0,A1)
-$006C	-108	DoCollision (RastPort)(A1)
-$0072	-114	DrawGList (RastPort,ViewPort)(A1,A0)
-$0078	-120	InitGels (DummyHead,DummyTail,GelsInfo)(A0,A1,A2)
-$007E	-126	InitMasks (VSprite)(A0)
-$0084	-132	RemIBob (Bob,RastPort,ViewPort)(A0,A1,A2)

-$008A	-138	RemVSprite (VSprite)(A0)
-$0090	-144	SetCollision (Type,Routine,GelsInfo)(D0,A0,A1)
-$0096	-150	SortGList (RastPort)(A1)
-$009C	-156	AddAnimObj (Obj,AnimationKey,RastPort)(A0,A1,A2)
-$00A2	-162	Animate (AnimationKey)(RastPort)(A0,A1)
-$00A8	-168	GetGBuffers (AnimationObj,RastPort,DoubleBuffer) (A0,A1,D0)
-$00AE	-174	InitGMasks (AnimationObj)(A0)
-$00B4	-180	GelsFuncE ()
-$00BA	-186	GelsFuncF()
-$00C0	-192	LoadRGB4 (ViewPort,Colors,Count)(A0,A1,D0)
-$00C6	-198	InitRastPort (RastPort)(A1)
-$00CC	-204	InitVPort (ViewPort)(A0)
-$00D2	-210	MrgCop (View)(A1)
-$00D8	-216	MakeVPort (View,ViewPort)(A0,A1)
-$00DE	-222	LoadView (View)(A1)
-$00E4	-228	WaitBlit ()
-$00EA	-234	SetRast (RastPort,Color)(A1,D0)
-$00F0	-240	Move (RastPort,x,y)(A1,D0,D1)
-$00F6	-246	Draw (RastPort,x,y)(A1,D0,D1)
-$00FC	-252	AreaMove (RastPort,x,y)(A1,D0,D1)
-$0102	-258	AreaDraw (RastPort,x,y)(A1,D0,D1)
-$0108	-264	AreaEnd (RastPort)(A1)
-$010E	-270	WaitTOF()
-$0114	-276	QBlit (Blit)(A1)
-$011A	-282	InitArea (AreaInfo,VectorTable,VectorTableSize) (A0,A1,D0)
-$0120	-288	SetRGB4 (ViewPort,Index,r,g,b)(A0,D0,D1,D2,D3)
-$0126	-294	QBSBlit (Blit)(A1)
-$012C	-300	BltClear (Memory,Size,Flags)(A1,D0,D1)
-$0132	-306	RectFill (RastPort,xl,yl,xu,yu)(A1,D0,D1,D2,D3)
-$0138	-312	BltPattern (RastPort,Ras,xl,yl,MaxX,MaxY,FillBytes) (A1,A0,D0,D1,D2,D3,D4)
-$013E	-318	ReadPixel (RastPort,x,y)(A1,D0,D1)
-$0144	-324	WritePixel (RastPort,x,y)(A1,D0,D1)
-$014A	-330	Flood (RastPort,Mode,x,y)(A1,D2,D0,D1)
-$0150	-336	PolyDraw (RastPort,Count,PolyTable)(A1,D0,A0)
-$0156	-342	SetAPen (RastPort,Pen)(A1,D0)
-$015C	-348	SetBPen (RastPort,Pen)(A1,D0)
-$0162	-354	SetDrMd (RastPort,DrawMode)(A1,D0)
-$0168	-360	InitView (View)(A1)

–$016E	–366	CBump (CopperList)(A1)
–$0174	–372	CMove (CopperList,Destination,Data)(A1,D0,D1)
–$017A	–378	CWait (CopperList,x,y)(A1,D0,D1)
–$0180	–384	VBeamPos ()
–$0186	–390	InitBitMap (Bm,Depth,Width,Height)(A0,D0,D1,D2)
–$018C	–396	ScrollRaster (RastPort,dx,dy,MinX,MinY,MaxX,MaxY) (A1,D0,D1,D2,D3,D4,D5)
–$0192	–402	WaitBOVP (ViewPort)(A0)
–$0198	–408	GetSprite (SimpleSprite,Num)(A0,D0)
–$019E	–414	FreeSprite (Num)(D0)
–$01A4	–420	ChangeSprite (Vp,SimpleSprite,Data)(A0,A1,A2)
–$01AA	–426	MoveSprite (Vp,SimpleSprite,x,y)(A0,A1,D0,D1)
–$01B0	–432	LockLayerRom (Layer)(A5)
–$01B6	–438	UnlockLayerRom (Layer)(A5)
–$01BC	–444	SyncSBitMap (1)(A0)
–$01C2	–450	CopySBitMap (11,12)(A0,A1)
–$01C8	–456	OwnBlitter ()
–$01CE	–462	DisownBlitter ()
–$01D4	–468	InitTmpRas (Tmpras,Buffer,Size)(A0,A1,D0)
–$01DA	–474	AskFont (RastPort,TextAttr)(A1,A0)
–$01E0	–480	AddFont (TextFont)(A1)
–$01E6	–486	RemFont (TextFont)(A1)
–$01EC	–492	AllocRaster (Width,Height)(D0,D1)
–$01F2	–498	FreeRaster (PlanePtr,Width,Height)(A0,D0,D1)
–$01F8	–504	AndRectRegion (Rgn,Rect)(A0,A1)
–$01FE	–510	OrRectRegion (Rgn,Rect)(A0,A1)
–$0204	–516	NewRegion ()
–$0210	–528	ClearRegion (Rgn)(A0)
–$0216	–534	DisposeRegion (Rgn)(A0)
–$021C	–540	FreeVPortCopLists (ViewPort)(A0)
–$0222	–546	FreeCopList (CopList)(A0)
–$0228	–552	ClipBlit (SrcRp,SrcX,SrcY,DestRp,DestX,DestY,SizeX, SizeY,Minterm)(A0,D0,D1,A1,D2,D3,D4,D5,D6)
–$022E	–558	XorRectRegion (Rgn,Rect)(A0,A1)
–$0234	–564	FreeCprList (CprList)(A0)
–$023A	–570	GetColorMap (Entries)(D0)
–$0240	–576	FreeColorMap (ColorMap)(A0)
–$0246	–582	GetRGB4 (ColorMap,Entry)(A0,D0)
–$024C	–588	ScrollVPort (Vp)(A0)
–$0252	–594	UCopperListInit (Copperlist,Num)(A0,D0)
–$0258	–600	FreeGBuffers (AnimationObj,RastPort,DoubleBuffer) (A0,A1,D0)

–$025E	–606	BltBitMapRastPort (SrcBm,SrcX,SrcY,DestRp,DestX, DestY,SizeX,SizeY,Minterm) (A0,D0,D1,A1,D2,D3,D4,D5,D6)

icon.library

–$001E	–30	GetWBObject (Name)(A0)
–$0024	–36	PutWBObject (Name,Object)(A0,A1)
–$002A	–42	GetIcon (Name,Icon,FreeList)(A0,A1,A2)
–$0030	–48	PutIcon (Name,Icon)(A0,A1)
–$0036	–54	FreeFreeList (FreeList)(A0)
–$003C	–60	FreeWBObject (WBObject)(A0)
–$0042	–66	AllocWBObject ()
–$0048	–72	AddFreeList (FreeList,Mem,Size)(A0,A1,A2)
–$004E	–78	GetDiskObject (Name)(A0)
–$0054	–84	PutDiskObject (Name,DiskObj)(A0,A1)
–$005A	–90	FreeDiskObject (Diskobj)(A0)
–$0060	–96	FindToolType (ToolTypeArray,TypeName)(A0,A1)
–$0066	–102	MatchToolValue (TypeString,Value)(A0,A1)
–$006C	–108	dumpRevision (NewName,OldName)(A0,A1)

intuition.library

–$001E	–30	OpenIntuition ()
–$0024	–36	Intuition (IEvent)(A0)
–$002A	–42	AddGadget (AddPtr,Gadget,Position)(A0,A1,D0)
–$0030	–48	ClearDMRequest (Window)(A0)
–$0036	–54	ClearMenuStrip (Window)(A0)
–$003C	–60	ClearPointer (Window)(A0)
–$0042	–66	CloseScreen (Screen)(A0)
–$0048	–72	CloseWindow (Window)(A0)
–$004E	–78	CloseWorkBench ()
–$0054	–84	CurrentTime (Seconds,Micros)(A0,A1)
–$005A	–90	DisplayAlert (AlertNumber,String,Height)(D0,A1,D1)
–$0060	–96	DisplayBeep (Screen)(A0)
–$0066	–102	DoubleClick (SSeconds,SMicros,CSeconds,CMicros) (D0,D1,D2,D3)
–$006C	–108	DrawBorder (RPort,Border,LeftOffset,TopOffset) (A0,A1,D0,D1)
–$0072	–114	DrawImage (RPort,Image,LeftOffset,TopOffset) (A0,A1,D0,D1)
–$0078	–120	EndRequest (Requester,Window)(A0,A1)
–$007E	–126	GetDefPrefs (Preferences,Size)(A0,D0)

-$0084	-132	GetPrefs (Preferences,Size)(A0,D0)
-$008A	-138	InitRequester (Req)(A0)
-$0090	-144	ItemAddress (MenuStrip,MenuNumber)(A0,D0)
-$0096	-150	ModifyIDCMP (Window,Flags)(A0,D0)
-$009C	-156	ModifyProp (Gadget,Ptr,Reg,Flags,HPos,VPos,HBody, VBody)(A0,A1,A2,D0,D1,D2,D3,D4)
-$00A2	-162	MoveScreen (Screen,dx,dy)(A0,D0,D1)
-$00A8	-168	MoveWindow (Window,dx,dy)(A0,D0,D1)
-$00AE	-174	OffGadget (Gadget,Ptr,Req)(A0,A1,A2)
-$00B4	-180	OffMenu (Window,MenuNumber)(A0,D0)
-$00BA	-186	OnGadget (Gadget,Ptr,Req)(A0,A1,A2)
-$00C0	-192	OnMenu (Window,MenuNumber)(A0,D0)
-$00C6	-198	OpenScreen (OSArgs)(A0)
-$00CC	-204	OpenWindow (OWArgs)(A0)
-$00D2	-210	OpenWorkBench ()
-$00D8	-216	PrintIText (Rp,IText,Left,Top)(A0,A1,D0,D1)
-$00DE	-222	RefreshGadgets (Gadgets,Ptr,Req)(A0,A1,A2)
-$00E4	-228	RemoveGadgets (RemPtr,Gadget)(A0,A1)
-$00EA	-234	ReportMouse (Window,Boolean)(A0,D0)
-$00F0	-240	Request (Requester,Window)(A0,A1)
-$00F6	-246	ScreenToBack (Screen)(A0)
-$00FC	-252	ScreenToFront (Screen)(A0)
-$0102	-258	SetDMRequest (Window,Req)(A0,A1)
-$0108	-264	SetMenuStrip (Window,Menu)(A0,A1)
-$010E	-270	SetPointer (Window,Pointer,Height,Width,XOffset, YOffset)(A0,A1,D0,D1,D2,D3)
-$0114	-276	SetWindowTitles (Window,WindowTitle,ScreenTitle) (A0,A1,A2)
-$011A	-282	ShowTitle (Screen,ShowIt)(A0,D0)
-$0120	-288	SizeWindow (Window,dx,dy)(A0,D0,D1)
-$0126	-294	ViewAddress ()
-$012C	-300	ViewPortAddress (Window)(A0)
-$0132	-306	WindowToBack (Window)(A0)
-$0138	-312	WindowToFront (Window)(A0)
-$013E	-318	WindowLimits (Window,MinWidth,MinHeight,MaxWidth, MaxHeight)(A0,D0,D1,D2,D3)
-$0144	-324	SetPrefs (Preferences,Size,Flag)(A0,D0,D1)
-$014A	-330	IntuiTextLength (IText)(A0)
-$0150	-336	WBenchToBack ()
-$0156	-342	WBenchToFront ()
-$015C	-348	AutoRequest (Window,Body,PText,NText,PFlag,NFlag, Width,Height)(A0,A1,A2,A3,D0,D1,D2,D3)

-$0162	-354	BeginRefresh (Window)(A0)
-$0168	-360	BuildSysRequest (Window,Body,PText,NText,Flags, Width,Height)(A0,A1,A2,A3,D0,D1,D2)
-$016E	-366	EndRefresh (Window,Complete)(A0,D0)
-$0174	-372	FreeSysRequest (Window)(A0)
-$017A	-378	MakeScreen (Screen)(A0)
-$0180	-384	RemakeDisplay ()
-$0186	-390	RethinkDisplay ()
-$018C	-396	AllocRemember (RememberKey,Size,Flags)(A0,D0,D1)
-$0192	-402	AlohaWorkbench (WBPort)(A0)
-$0198	-408	FreeRemember (RememberKey,ReallyForget)(A0,D0)
-$019E	-414	LockIBase (DontKnow)(D0)
-$01A4	-420	UnlockIBase (IBlock)(A0)

layers.library

-$001E	-30	InitLayers (Li)(A0)
-$0024	-36	CreateUpfrontLayer (Li,Bm,x0,y0,x1,y1,Flags,Bm2) (A0,A1,D0,D1,D2,D3,D4,A2)
-$002A	-42	CreateBehindLayer (Li,Bm,x0,y0,x1,y1,Flags,Bm2) (A0,A1,D0,D1,D2,D3,D4,A2)
-$0030	-48	UpfrontLayer (Li,Layer)(A0,A1)
-$0036	-54	BehindLayer (Li,Layer)(A0,A1)
-$003C	-60	MoveLayer (Li,Layer,dx,dy)(A0,A1,D0,D1)
-$0042	-66	SizeLayer (Li,Layer,dx,dy)(A0,A1,D0,D1)
-$0048	-72	ScrollLayer (Li,Layer,dx,dy)(A0,A1,D0,D1)
-$004E	-78	BeginUpdate (Layer)(A0)
-$0054	-84	EndUpdate (Layer)(A0)
-$005A	-90	DeleteLayer (Li,Layer)(A0,A1)
-$0060	-96	LockLayer (Li,Layer)(A0,A1)
-$0066	-102	UnlockLayer (Li,Layer)(A0,A1)
-$006C	-108	LockLayers (Li)(A0)
-$0072	-114	UnlockLayers (Li)(A0)
-$0078	-120	LockLayerInfo (Li)(A0)
-$007E	-126	SwapBitsRastPortClipRect (Rp,Cr)(A0,A1)
-$0084	-132	WhichLayer (Li,x,y)(A0,D0,D1)
-$008A	-138	UnlockLayerInfo (Li)(A0)
-$0090	-144	NewLayerInfo ()
-$0096	-150	DisposeLayerInfo (Li)(A0)
-$009C	-156	FattenLayerInfo (Li)(A0)
-$00A2	-162	ThinLayerInfo (Li)(A0
-$00A8	-168	MoveLayerInFrontOf (LayerToMove,LayerToBeIn FrontOf) (A0,A1)

mathffp.library

–$001E	–30	SPFix (Float)(D0)
–$0024	–36	SPFlt (Integer)(D0)
–$002A	–42	SPCmp (LeftFloat,RightFloat)(D1,D0)
–$0030	–48	SPTst (Float)(D1)
–$0036	–54	SPAbs (Float)(D0)
–$003C	–60	SPNeg (Float)(D0)
–$0042	–66	SPAdd (LeftFloat,RightFloat)(D1,D0)
–$0048	–72	SPSub (LeftFloat,RightFloat)(D1,D0)
–$004E	–78	SPMul (LeftFloat,RightFloat)(D1,D0)
–$0054	–84	SPDiv (LeftFloat,RightFloat)(D1,D0)

mathieeedoubbas.library

–$001E	–30	IEEEDPFix (Integer,Integer)(D0,D1)
–$0024	–36	IEEEDPFlt(Integer)(D0)
–$002A	–42	IEEEDPCmp(Integer,Integer,Integer,Integer) (D0,D1,D2,D3)
–$0030	–48	IEEEDPTst(Integer,Integer)(D0,D1)
–$0036	–54	IEEEDPAbs(Integer,Integer)(D0,D1)
–$003C	–60	IEEEDPNeg(Integer,Integer)(D0,D1)
–$0042	–66	IEEEDPAdd(Integer,Integer,Integer,Integer) (D0,D1,D2,D3)
–$0048	–72	IEEEDPSub(Integer,Integer,Integer,Integer) (D0,D1,D2,D3)
–$004E	–78	IEEEDPMul (Integer,Integer,Integer,Integer) (D0,D1,D2,D3)
–$0054	–84	IEEEDPDiv (Integer,Integer,Integer,Integer) (D0,D1,D2,D3)

mathtrans.library

–$001E	–30	SPATan (Float)(D0)
–$0024	–36	SPSin (Float)(D0)
–$002A	–42	SPCos (Float)(D0)
–$0030	–48	SPTan (Float)(D0)
–$0036	–54	SPSincos (LeftFloat,RightFloat)(D1,D0)
–$003C	–60	SPSinh (Float)(D0)
–$0042	–66	SPCosh (Float)(D0)
–$0048	–72	SPTanh (Float)(D0)
–$004E	–78	SPExp (Float)(D0)
–$0054	–84	SPLog (Float)(D0)

-$005A	-90	SPPow (LeftFloat,RightFloat)(D1,D0)
-$0060	-96	SPSqrt (Float)(D0)
-$0066	-102	SPTieee (Float)(D0)
-$006C	-108	SPFieee (Float)(D0)
-$0072	-114	SPAsin (Float)(D0)
-$0078	-120	SPAcos (Float)(D0)
-$007E	-126	SPLog10 (Float)(D0)

potgo.library

-$0006	-6	AllocPotBits (Bits)(D0)
-$000C	-12	FreePotBits (Bits)(D0)
-$0012	-18	WritePotgo (Word,Mask)(D0,D1)

timer.library

-$002A	-42	AddTime (Dest,Src)(A0,A1)
-$0030	-48	SubTime (Dest,Src)(A0,A1)
-$0036	-54	CmpTime (Dest,Src)(A0,A1)

translator.library

-$001E	-30	Translate (InputString,InputLength,OutputBuffer, BufferSize)(A0,D0,A1,D1)

Anhang F: Der Befehlssatz des MC68000

Der MC68000 besitzt einen umfangreichen Befehlssatz und eine Vielzahl von Adressierungsarten. An dieser Stelle wollen wir nun die Befehle des MC68000 auflisten.

Abkürzungen:

Label	Sprungkennzeichnung, bzw. Adresse
Reg	Register
An	Adressregister n
Dn	Datenregister n
Rn	Adress- oder Datenregister n
Quelle	Quelloperand
Ziel	Zieloperand
‹ea›	Effektive Adresse
#n	Konstante des Wertes n

Der Befehlssatz:

ABCD	Quelle,Ziel	Addition zweiter BCD-Zahlen
ADD	Quelle,Ziel	Binäre Addition
ADDA	Quelle,An	Binäre Addition mit einem Adreßregister
ADDI	#n,‹ea›	Addition mit einer Konstanten
ADDQ	#n,‹ea›	Schnelle Addition einer 3-Bit-Konstanten
ADDX	Quelle,Ziel	Addition mit Übertrag im X-Flag
AND	Quelle,Ziel	Logisches UND
ANDI	#n,‹ea›	Logisches UND mit einer Konstanten
ASL	n,‹ea›	Arithmetische Linksverschiebung (*2^n)
ASR	n,‹ea›	Arithmetische Rechtsverschiebung (/2^n)
Bcc	Label	Bedingte Verzweigung zum Label (beispielsweise BPL,BMI,BNE,BEQ)
BCHG	#n,‹ea›	Negiere Bit n
BCLR	#n,‹ea›	Lösche Bit n
BRA	Label	Verzweige nach Label (ähnlich wie JMP)
BSET	#n,‹ea›	Setze Bit n
BSR	Label	Verzweige in Unterprogramm (ähnlich wie JSR). Rücksprungadresse wird auf den Stack gelegt.
BTST	#n,‹ea›	Test Bit n. Ergebnis steht im Z-Flag
CHK	‹ea›,Dn	Prüfe ein Datenregister auf eine Grenze, löse gegebenenfalls die CHK-Exception aus.
CLR	‹ea›	Löschen eines Operanden
CMP	Quelle,Ziel	Vergleiche zwei Operanden
CMPA	‹ea›,An	Vergleich mit einem Adreßregister
CMPI	#n,‹ea›	Vergleich mit einer Konstanten
CMPM	Quelle,Ziel	Vergleich zweier RAM-Operanden
DBcc	Reg,Label	Prüfe auf Bedingung, dekrementiere Register und verzweige gegebenenfalls (ähnlich Bcc).
DIVS	Quelle,Ziel	Vorzeichenrichtige Division eines 32-Bit- durch einen 16-Bit-Operanden. Das Ergebnis steht im Low-Word des Zielregisters und der Divisionsrest im High-Word.
DIVU	Quelle,Ziel	Vorzeichenlose Division
EOR	Quelle,Ziel	Exklusives-Oder
EORI	#n, ‹ea›	Exklusives-Oder mit einer Konstanten
EXG	Reg,Reg	Tauschen zweier Registerinhalte
EXT	Dn	Vorzeichenrichtige Erweiterung auf doppelt Größe Byte –›Word,Word –› Longword)
JMP	Label	Verzweigung zum Label (ähnlich BRA)
JSR	Label	Verzweigung zu einer Unterroutine, wobei die Rücksprungadresse auf dem Stack gespeichert wird (ähnlich wie BSR).

LEA	‹ea›,An	Lade eine Adresse in ein Adreßregister
LINK	An,#n	Stackbereich aufbauen
LSL	n,‹ea›	Logische Linksverschiebung
LSR	n,‹ea›	Logische Rechtsverschiebung
MOVE	Quelle,Ziel	Übertragung eines Wertes von Quelle nach Ziel
MOVE	SR,‹ea›	Statusregisterinhalt wird übertragen
MOVE	‹ea›,SR	Statusregisterinhalt wird gesetzt
MOVE	‹ea›,CCR	Flags werden gesetzt
MOVE	USP,‹ea›	Userstackpointer wird übertragen
MOVE	‹ea›,USP	Userstackpointer wird gesetzt
MOVEA	‹ea›,An	Wert wird in Adreßregister übertragen
MOVEM	Reg,‹ea›	Mehrere Register werden übertragen
MOVEM	‹ea›,Reg	Mehrere Register werden gesetzt
MOVEP	Quelle,Ziel	Übertrage Daten zur Peripherie
MOVEQ	#n,Dn	Eine Byte-Konstante wird übertragen
MULS	Quelle,Ziel	Vorzeichenrichtige Multiplikation zweier Words. Das Ergebnis steht als Longword im Ziel.
MULU	Quelle,Ziel	Multiplikation ohne Vorzeichen
NBCD	Quelle,Ziel	Negation einer BCD-Zahl (Neunerkomplement)
NEG	‹ea›	Negation eines Operanden (Zweierkomplement)
NEGX	‹ea›	Negation mit Übertrag im X-Flag
NOP		No Operation/Keine Funktion
NOT	‹ea›	Inversion eines Operanden
OR	Quelle,Ziel	Logisches ODER
ORI	#n,‹ea›	Logisches ODER mit einer Konstanten
PEA	‹ea›	Adresse wird auf Stack abgelegt
RESET		Peripherie zurücksetzen
ROL	n,‹ea›	Linksrotation
ROR	n,‹ea›	Rechtsrotation
ROXL	n,‹ea›	Linksrotation mit Übertrag im X-Flag
ROXR	n,‹ea›	Rechtsrotation mit Übertrag im X-Flag
RTE		Rückkehr aus einer Exception
RTR		Rückkehr mit Laden der Flags
RTS		Rückkehr aus einer Unterroutine (JSR, BSR)
SBCD	Quelle,Ziel	Subtraktion zweier BCD-Zahlen
Scc	‹ea›	Setze Byte auf –1, wenn Bedingung erfüllt. (siehe auch Bcc und DBcc)
STOP		Prozessor stoppen, evtl. TRAPV-Exception starten
SUB	Quelle,Ziel	Binäre Subtraktion
SUBA	‹ea›,An	Binäre Subtraktion mit einem Adreßregister
SUBI	#n,‹ea›	Subtraktion einer Konstanten

SUBQ	#n,‹ea›	Schnelle Subtraktion einer 3-Bit-Konstanten
SUBX	Quelle,Ziel	Subtraktion mit Übertrag im X-Flag
SWAP	Dn	Tauschen der beiden Registerwords (obere 16 Bits mit unteren 16 Bits tauschen)
TAS	‹ea›	Prüfe Byte und setze Bit 7
TRAP	#n	Springe in Exception Nr. n
TRAPV		Prüfen, ob Overflow-Flag gesetzt
UNLK		An Stackbereich abbauen

Anhang G: Die Jumper des Amiga B2000

Der Amiga B2000 besitzt einige Jumper, bzw Lötbrücken auf der Platine, mit denen verschiedene Funktionen konfiguriert werden können:

J101: Dieser Jumper verbindet in der Normalstellung die Pins 1 und 2. In dieser Stellung verwendet FatAgnus die Adressleitung A23 zur Speicherverwaltung. In der anderen Stellung wird die Adressleitung A19 verwendet.

J200: Dieser Jumper setzt den Port, der für den Light-Pen verwendet werden soll. In der normalen Position sind die Pins 2 und 3 verbunden, wodurch der Port 1 gesetzt wird, also wie beim Amiga 500. In der anderen Position wird Port 0 verwendet, also wie beim Amiga 1000.

J300: Dieser Jumper setzt das Signal, das als Time-Base für die CIA-Chips verwendet werden soll. In der Normalstellung sind die Pins 1 und 2 verbunden, wodurch das AC-Line-Frequenz- Signal verwendet wird. In der anderen Position wird das Vertikale Synchronisationssignal verwendet.

J301: Ist dieser Jumper geschlossen, so wird ein zweites internes Laufwerk konfiguriert. Ist er offen, dann nicht.
J500: Normalerweise ist dieser Jumper geschlossen. Wird er geöffnet, so wird kein FastRam verwendet.
iguriert werden können:

Anhang H: Literaturnachweis

Folgende Literatur haben wir zu Informationszwecken herangezogen:

1.) Addison-Wesley Referenzmanuals 'Hardware' und 'Libraries and Devices'

2.) Commodore Technical Referencemanuals A500, A1000, A2000A, A2000B und SideCar.

3.) Amiga Programmierhandbuch von Frank Kremser & Jörg Koch MT-Nr. 90491, ISBN 3-89090-491-2

Anhang: I: Die beigefügte Diskette

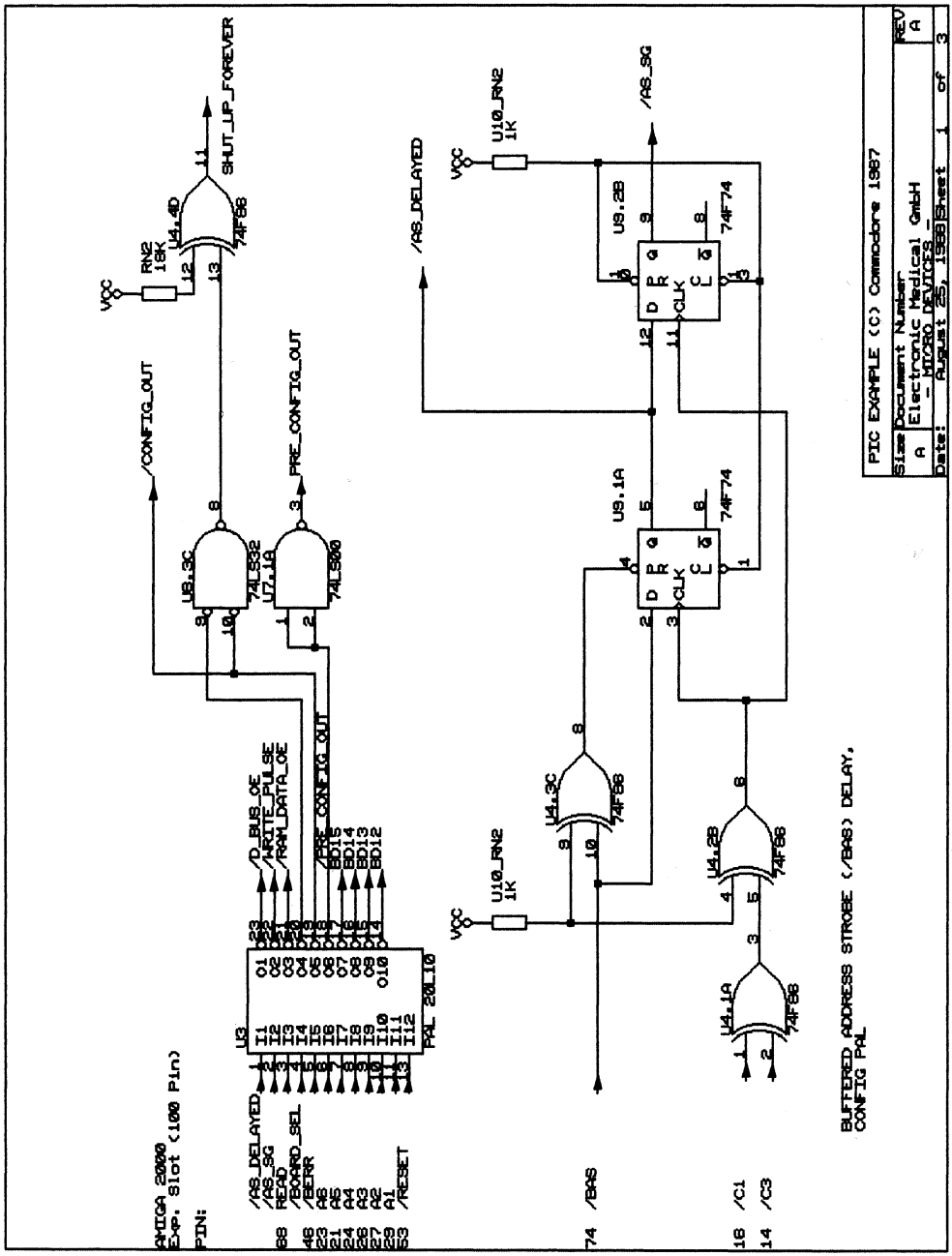
Dieser Anhang enthält ein Inhaltsverzeichnis der Unterverzeichnisse »CPrg« und »MCPrg«, die sich auf Diskette befinden. Diese Diskette enthält alle Programme, die in diesem Buch als Source und als lauffähiges Programm angeführt sind, wobei eventuell benötigte Dateien ebenfalls vorhanden sind. Um die Programe starten zu können, wird empfohlen, das CLI zu starten, mittels »cd«-Befehl auf das betreffende Programm zu starten. Es muß noch angemerkt werden, daß es vorkommen kann, daß einige C-Programme nicht laufen, wenn sie mit einem C-Compiler älteren Datums nachkompiliert werden. Zu den Assembler-Programmen ist anzumerken, daß eine eventuell vorhandene Speichererweiterung mittels »NoFastMem« abgeschaltet werden sollte, damit diese einwandfrei laufen.

Die C-Programme:

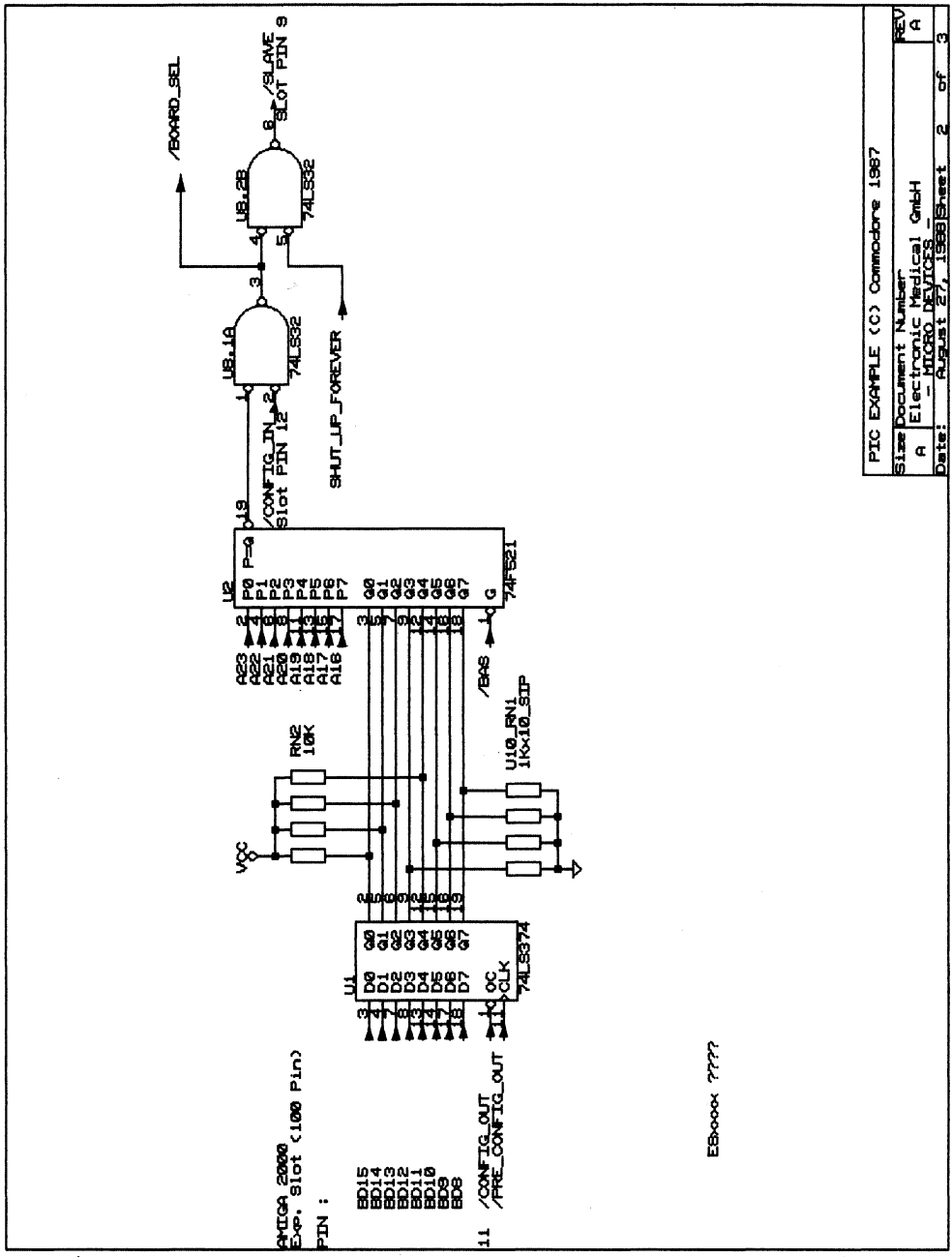
„fastdir	AMIGAFILE
Audio1	Audio1.c
Audio2	Audio2.c
Audio3	Audio3.c
Blitter	Blitter.c
BOOTBLOCK	BOOTBLOCK.S
Copper1	Copper1.c
Copper2	Copper2.c
Copper3	Copper3.c
Disk	Disk.c
Expansion	Expansion.c
Janus	Janus.c
Joyport1	Joyport1.c
Joyport2	Joyport2.c
PCJANUS	Play1
Play1.c	Play2
Play2.c	Play3
Play3.c	Sprite
Sprite.c	Tastatur1
Tastatur1.c	Tastatur2.COMP
Taste.ASM	Taste.o

Die Maschinenprogramme:

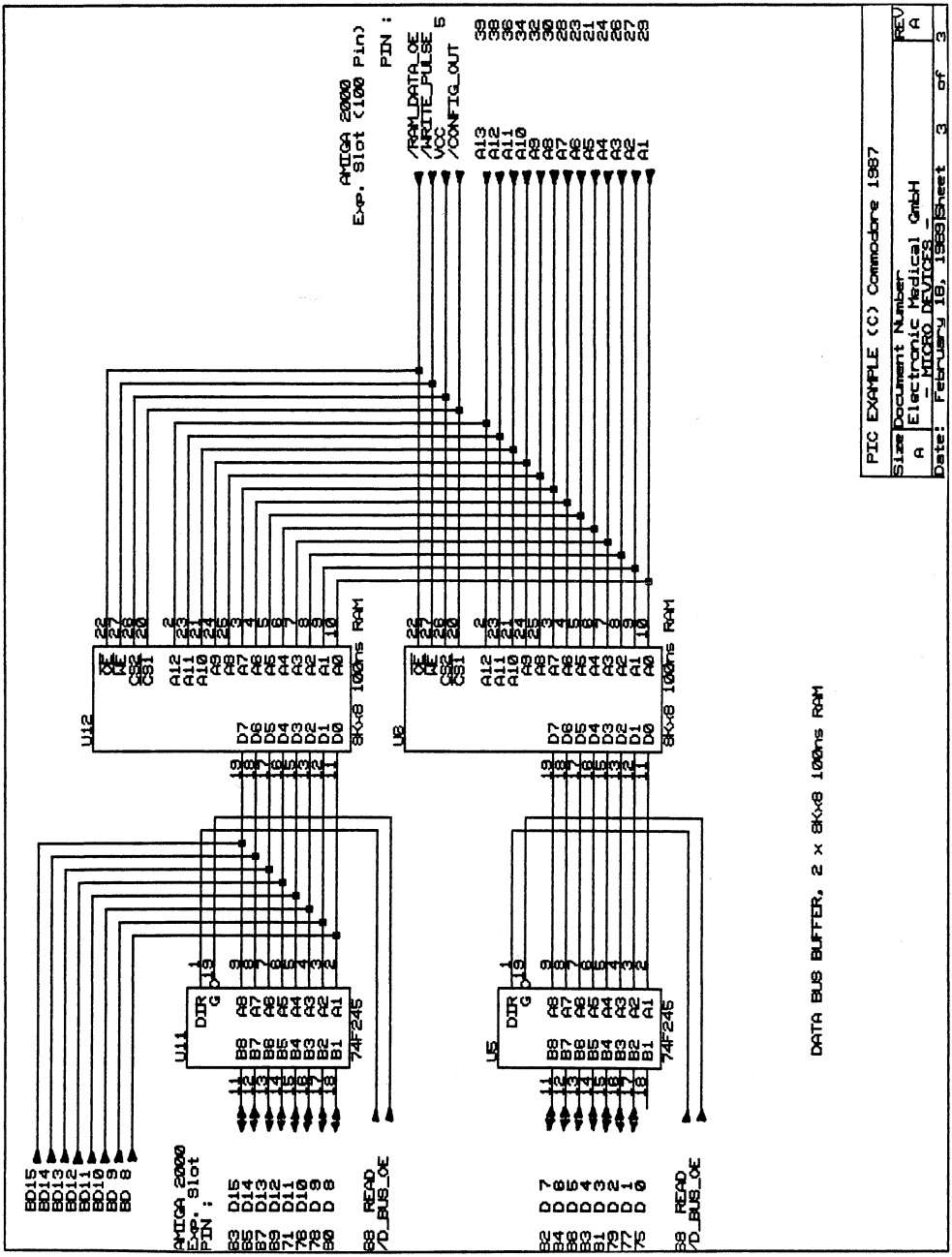
„fastdir
Beam.S
Blitter1.S
Copper1.S
Copper2.S
Floppy1
Floppy2
Floppy3
Floppy4
Irq
Led
Parallel
Play1
Play2
Resetfest
SOUND
Sound1.S
Sound2.S
Sound3.S
Sound4.S
Sound5.S
Sound6.S
Sprite1.S
Sprite2.S
Sprite3.S
Sprite4.S
Beam
Blitter1
Copper1
Copper2
DEMOSOUND
Floppy1.S
Floppy2.S
Floppy3.S
Floppy4.S
Irq.S
Led.S
Parallel.S
Play1.S
Play2.S
Resetfest.S
Sound1
Sound2
Sound3
Sound4
Sound5
Sound6
Sprite1
Sprite2
Sprite3
Sprite4



Der Schaltplan für die Erweiterungskarte aus Kap. 6.1 (Teil 1).



Der Schaltplan für die Erweiterungskarte aus Kap. 6.1 (Teil 2).



Der Schaltplan für die Erweiterungskarte aus Kap. 6.1 (Teil 3).

Wir danken der Firma Commodore Büromaschinen GmbH für die freundliche Erlaubnis, verschiedene Zeichnungen und Programme aus dem „A500/A2000Technical Reference Manual“ und dem „A500Technical-Manual“ übernehmen zu dürfen.

A 500/A 2000 Technical Reference Manual:

- Seite 2 Figure 1.1: Key Codes
- Seite 14, 15 und 16: Blockdiagramme
- Seite 119 und 120: PC-Emulator
- Seite 188 und 189: FATAgnus
- Seite 213 f.: CIA 8520
- App. A, A-4, A-5, A-6, A-7
- Seite 43: PAL-Programm
- App. A A-2: Demoplatine

A 500 Technical Manual:

- Seite 3-6: MC 68000
- Seite 3-8, 3-9: Paula und Blockdiagramm
- Seite 3-10, 3-11: Denise und Blockdiagramm
- Seite 3-14, 3-15: Garry und Blockdiagramm

Stichwortverzeichnis

AMIGA System-Handbuch

Die Autoren:
JÖRG KOCH, geboren 1967, befindet sich zur Zeit in der Ausbildung zum Energie-Anlagen-Elektroniker. Neben der Ausbildung beschäftigt er sich als freier Autor auf dem Spezialgebiet Software-Entwicklung unter anderem mit den Programmiersprachen Modula und C auf verschiedenen 16-Bit-Rechnern.
FRANK KREMSER, geboren 1967, erwarb auf vielen Gebieten der Informatik Grundkenntnisse, die er in Form von Programmen und Berichten in Fachzeitschriften einem breiten Publikum zugänglich gemacht hat.
Beide Autoren haben 1987 mit einem Lernprogramm für den Amiga den Wettbewerb »Goldene Diskette«, unter der Schirmherrschaft von Bundesforschungsminister Dr. Heinz Riesenhuber, erneut gewonnen, nachdem sie bereits 1986 mit einem Programm für den Apple erfolgreich waren.

Bundesforschungsminister Dr. Heinz Riesenhuber übergibt Frank Kremser und Jörg Koch aus Marburg die »Goldene Diskette«.

Warum soll Computerkenntnis bei der Anwendung von Software enden? Es gibt schließlich auch bei der Hardware des Amiga 500, 1000, A2000 und B2000 viele Dinge, die wissenswert sind. Das Amiga-System-Handbuch hilft sowohl dem Computer-Einsteiger weiter, der seinen Arbeitsspeicher vergrößern will, als auch dem erfahrenen Bastler, der Agnus, Denise, Paula, Garry und Buster, die Custom-Chips des Amigas, endlich einmal persönlich kennenlernen möchte. Zu allen Kapiteln gibt es Schaubilder und Schaltpläne. Und damit Sie sich im Dschungel der Elektronik gut zurechtfinden, sind fast alle Amiga-Ecken und -Winkel mit Farbfotos dokumentiert. Auf der beigefügten Diskette befinden sich alle im Buch als Listing aufgeführten Utilities in ablauffähiger Version. Somit können Sie ohne lästige Tipparbeit in die Tiefen Ihres Amigas hinabsinken.

Aus dem Inhalt:

- Geschichte des Amigas
- Vorgänge beim Booten
- Die Chips: Funktionen und Pinbelegungen
- Die Slots: Belegung der Steckplätze des Amigas – die Möglichkeiten der Video-, PC- und AT-Slots – Beschreibung der Autokonfiguration der Erweiterungskarten – Hardwarebeispiel
- Die Peripherie: Aufbau, Funktionsweise und Aufzeichnungsverfahren der Floppy – Belegung und Ansteuerung der parallelen und seriellen Schnittstelle – Arbeitsweise der Tastatur – Aufbau und Steuerung der Maus – Beschreibung der Monitore und Verbesserungsvorschläge
- MS-DOS-Erweiterung: alle Informationen zum SideCar, zur PC/XT- und zur PC/AT-Karte und deren RAM-Speicher
- RAM-Erweiterungen: statisches und dynamisches RAM – Möglichkeiten und Schwierigkeiten der Speichererweiterung
- Bastelanregungen: Anleitungen zum Bau eines Digitizers, eines Scanners und eines Low-Cost-Genlock-Interfaces für ca. 20 DM – Bestückungslisten und Schaltpläne
- Janus- und Expansion-Library
- Anhang: Speicherbelegung – Hardwareregister – Einsprungadressen der Bibliotheksfunktionen – Befehlssatz des MC68000 – Literaturhinweise

Die Begleitdiskette:

Auf der Begleitdiskette sind alle im Buch aufgeführten Programme als C- oder Assembler-Source-Code sowie in ablauffähiger Version enthalten.

Hardware-Anforderungen:

Amiga 500, 1000, A2000 oder B2000

Markt&Technik

ISBN 3-89090-550-1

4 001057 905504 07900

DM 79,– sFr 72,70 öS 616,20