Technical Reference Manual

AMIGA

A500/A2000

Commodore

Commodore® Amiga®
A500/A2000
Technical Reference
Manual

COPYRIGHT

This manual is copyright © 1986, 1987 by Commodore-Amiga, Inc. All Rights Reserved. This document may not, in whole or part, be copied, photocopied, reproduced, translated or transferred to any electronic medium or machine readable form without prior consent, in writing, from Commodore-Amiga, Inc.

Amiga is a registered trademark of Commodore-Amiga, Inc.
Commodore and CBM are registered trademarks of Commodore Electronics Limited.
Hayes is a registered trademark of Hayes Microcomputer Products, Inc.
IBM is a registered trademark of International Business Machines Corporation.
MacIntosh is a trademark of Apple Computer, Inc.

DISCLAIMER

THE INFORMATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED. THE ENTIRE RISK AS TO THE ACCURACY OF THE INFORMATION HEREIN IS ASSUMED BY YOU. COMMODORE-AMIGA DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE OF, OR THE RESULTS OF THE USE OF, THE INFORMATION IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS, OR OTHERWISE. IN NO EVENT WILL COMMODORE-AMIGA, INC. BE LIABLE FOR DIRECT, INDIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT IN THE INFORMATION EVEN IF IT HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME LAWS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF IMPLIED WARRANTIES OR LIABILITIES FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY.

Schematics represent current machine which is subject to change without notice.

Credits

The material for this manual was produced by Engineering, Documentation, and Technical Support staff at Commodore West Chester, Commodore Braunschweig, and Commodore-Amiga. Individuals contributing major portions of information and input are Dave Haynie, Jeff Porter, Phil Lindsay, Carolyn Scheppner, Lisa Siracusa, George Robbins, Andy Finkel, Eric Cotton, Jeff Boyer, Steve Ahlbom, Steve Beats, Dieter Preiss, Bernd Assmann, and Torsten Burgdorf.

This manual was compiled and edited by Steve Finkel.

Manual design by Jo-Ellen Temple and Wilson Harp.

A2000/A500 Technical Reference Manual
Table of Contents

Section 1	Summary of Differences	1
Section 2	System Block Diagrams	13
Section 3	Amiga Expansion	
3.1	Designing hardware for the Amiga Expansion Architecture	17
3.2	Driver Documentation	51
3.3	Software for Amiga Expansion	55
3.4	Amiga Expansion Connectors	
	100 Pin	75
	86 Pin	87
	Video Slot	101
Section 4	PC Bridgeboard	
4.1	Description of the PC/XT emulator for the Amiga 2000	109
4.2	BIOS entry points	121
4.3	Janus library	131
Section 5	Amiga Hard Disk/SCSI Controller	159
Section 6	Custom Chips	
	Fat Agnus Chip	187
	8520 Chip	213
Section 7	Miscellaneous Hardware Information	
7.1	Clock/calendar registers	223
7.2	Power budgets	225
7.3	A2000 PAL equations	229
7.4	B2000 Jumpers	235
Appendix A.	**Diagrams**	
A-1	Backplane Example	A-1
A-2	PIC Example	A-2
A-3	A500 Exterior (86-pin expansion connector)	A-3
A-4	Amiga 2000 Expansion Board Layout	A-4
A-5	Amiga 2000 Form Factor	A-5
A-6	Amiga 2000 Video Card	A-6
A-7	86-Pin Slot Expansion Board	A-7
A-8	A2000/B2000 Keyboard Connector Pinout	A-8
A-9	Amiga 500/2000 Mouse Diagram and Pinout	A-9
Appendix B.	**Schematics**	
	A2000 Schematics	A2000-1
	B2000 Schematics	B2000-1
	A500 Schematics	A500-1

Summary of Differences

This manual presents technical documentation for three different Amiga models, comparing them to the original Amiga, referred to as model A1000. Technical information included in this manual is relevant for the following Commodore Amiga models:

- the Amiga 500 (A500), a low-cost version of the original Amiga computer, software-compatible with the A1000. Unlike the A1000, the A500 has an integrated keyboard, provision for internal memory expansion up to 1 megabyte, new-style hardware connectors, and Kickstart code in ROM.

Two versions of the Amiga 2000:

- the A2000 is software-compatible with the A1000 and has internal slots, real time clock/calendar and new-style hardware connectors.

- the B2000, the cost-reduced version of the Amiga 2000, features some different custom chips, but is otherwise similar to the A2000.

The B2000 is still under development, and the information presented in this document is subject to change. The information included on the B2000 is intended to aid developers in designing software and peripherals that are applicable for both the current and upcoming version of the Amiga 2000.

Unless differences are specifically noted, information presented for the A2000 also holds true for the B2000. The differences between the two Amiga 2000 models are mainly hardware differences which will affect peripheral design, but not the way the computers function with software. Section 2 contains system block diagrams for all three new Amiga models.

KICKSTART IN ROM

Both the Amiga 2000 and the Amiga 500 feature version 1.2 of Kickstart built into ROM. Kickstart 1.2 (currently version 33.180) boots automatically when the Amiga is turned on.

XTRA KEYS ON THE EYBOARD

Both the Amiga 2000 and 500 feature 94-key keyboards, as compared to the A1000's 89-key keyboard. (The European versions of the keyboards have 96 keys.) The new keys are all located on the numeric keypad, and include:

KEY		SCAN CODE
Left parentheses	(	$5A
Right parentheses	)	$5B
Slash	/	$5C
Asterisk	*	$5D
Plus	+	$5E

In PC mode on the Amiga 2000 (using a Bridgeboard), these keys assume typical PC functions, including Number lock (left parenthesis), Print screen (asterisk) and Scroll lock (right parenthesis).

On some keyboards, the left Amiga key has been replaced by the Commodore key. This key performs identically in either case.

AW KEY CODES ON HE KEYBOARD

Keyboard Layout Showing Raw Key Codes

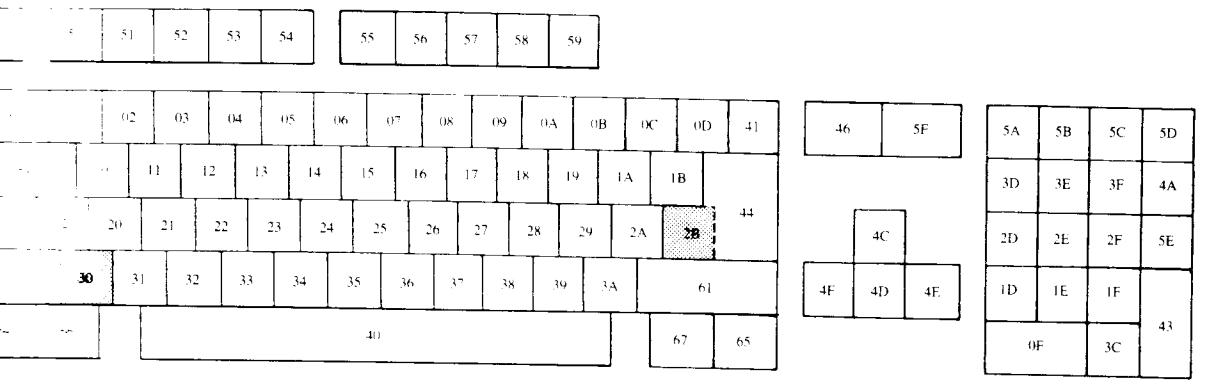


Figure 1.1 Key Codes

Note: On the U.S. keyboard, the keys with codes 44 and 60 are extended to include the European keys with codes 2B and 30, respectively. Also note that England uses the U.S. rather than the European keyboard, but not the U.S. keymap.

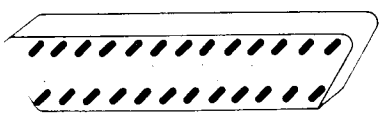
See Table 1-1 at the end of this section for a table of the raw key codes.

EXTERNAL SYSTEM I/O

This section describes each I/O interface in detail, and some of the tradeoffs made with respect to A1000 compatibility.

The Amiga 2000 and Amiga 500 have differences in the serial and parallel ports from the Amiga 1000, the main difference being changes in the sex of each port (changing the serial to female and the parallel to male), which allows the new Amigas to use standard interface cables.

RS232 and MIDI Port



The RS232 connector on the A500 and A2000 is form fit and function identical to a Commodore PC-10/20 with a few exceptions. **This is the OPPOSITE sex connector from the A1000.** The connector is a shielded male DB25P connector. The A1000 supplies various non-standard RS232 signals on the DB25 connector. These non-standard signals were removed wherever possible. The RS232 connector is NOT physically compatible with some MIDI interfaces but is compatible with the Amiga Modem/1200 RS (model 1680). Below is a comparison chart between the RS232 standard, a Hayes Smartmodem standard, the A1000 RS232, and the new Amiga 500/2000 RS232 connector.

PIN	RS232	A1000	A500/ A2000	PC10	HAYES®	DESCRIPTION
1	GND	GND	GND	GND	GND	Frame ground
2	T×D	T×D	T×D	T×D	T×D	Transmit Data
3	R×D	R×D	R×D	R×D	R×D	Receive Data
4	RTS	RTS	RTS	RTS	—	Request to send
5	CTS	CTS	CTS	CTS	CTS	Clear to send
6	DSR	DSR	DSR	DSR	DSR	Data set ready
7	GND	GND	GND	GND	GND	Signal ground
8	DCD	DCD	DCD	DCD	DCD	Carrier detect
9	—	—	+12v	+12v	—	+12 volt power
10	—	—	−12v	−12v	—	−12 volt power
11	—	—	AUDO	—	—	Audio output
12	S.SD	—	—	—	SI	Speed Indicate
13	S.CTS	—	—	—	—	
14	S.T×D	−5Vdc	—	—	—	−5 volt power
15	T×C	AUDO	—	—	—	Audio output
16	S.R×D	AUDI	—	—	—	Audio input
17	R×C	EB	—	—	—	Port clock 716KHz
18	—	INT2*	AUDI	—	—	Interrupt line/Audio input
19	S.RTS	—	—	—	—	
20	DTR	DTR	DTR	DTR	DTR	Data terminal ready
21	SQD	+5Vdc	—	—	—	+5 volt power
22	RI	—	RI	RI	RI	Ring indicator
23	SS	+12Vdc	—	—	—	+12 volt power
24	T×C1	C2*	—	—	—	3.58MHz clock
25	—	RESB*	—	—	—	Buffered system reset

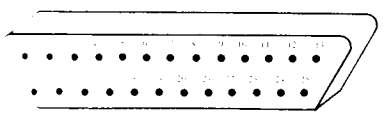
As you will notice, the A500 and 2000 deletes clocks and interrupt lines from the A1000. The +/−5Vdc and reset lines are also deleted. The +/−12Vdc lines are identical to a PC10/20.

The following signals (formerly on the RS232 connector) can be found on other connectors:

ResB = parallel connector
C2 = video connector

Centronics Port

The Centronics port also has some non-standard signals. Below is a table comparing the A1000 Centronics port with the A500/A2000 Centronics port. Again, this is the opposite sex from the A1000 and the same sex connector as an IBM®-PC (i.e., a female DB25 connector).



PIN	A1000	A500/A2000	PC10
1	DRDY*	STROBE*	STROBE*
2	Data 0	Data 0	Data 0
3	Data 1	Data 1	Data 1
4	Data 2	Data 2	Data 2
5	Data 3	Data 3	Data 3
6	Data 4	Data 4	Data 4
7	Data 5	Data 5	Data 5
8	Data 6	Data 6	Data 6
9	Data 7	Data 7	Data 7
10	ACK*	ACK*	ACK*
11	BUSY(data)	BUSY	BUSY
12	POUT(clk)	POUT	POUT
13	SEL	SEL	SEL
14	GND	+5v pullup	AUTOFDXT*
15	GND	NC	ERROR*
16	GND	RESET*	INIT*
17	GND	GND	SLCT IN*
18-22	GND	GND	GND
23	+5v	GND	GND
24	NC	GND	GND
25	Reset*	GND	GND

Video Output

The A500 and A2000, like the A1000, use a DB23 video connector.

This 23 pin connector contains all the signals necessary to work with a Genlock, but the current Genlock will need to be redesigned in order to meet the physical requirements of the A500 and A2000, in-

stead of the A1000. An A500 genlock will also have to supply its own power. Power will not be provided for the Genlock. All signals on the 23 pin connector are the same except for the power.

In addition to the 23 pin video connector, the A500/B2000 provides a monochrome composite video output, unlike the A1000. This provides the capability of using a low-cost, high persistence monochrome monitor with the A500 for viewing 640 × 400 interlaced video without as much flickering.

Power is provided for the A520 modulator and composite video adapter.

Mouse and Joystick Ports

The mouse and joystick ports of the A500 and A2000 are identical to the A1000, except that the current limiting protection circuitry has been eliminated. The A500 and A2000 use a different mouse than the one the A1000 uses. A diagram and information on this mouse is included in Appendix A of this manual.

A500 Expansion Port

The expansion port is electrically compatible with the A1000, but because of its physical location, it cannot accept any A1000 expansion peripherals without some further adapter. Power is supplied to this connector, but only enough for a ROM cartridge. The exact pinout of this 86 pin edge connector appears later in this document, in the section of Amiga expansion. The A500 diagram in Appendix A shows the new positioning of this port (relative to A1000) and the pin numbers.

A500 RAM Expansion

Associated with the built-in 512KB of RAM is a header socket to allow an additional 512KB of RAM and a battery backed-up real time clock board to be added. This small PCB (the A501 RAM Expansion Cartridge) can easily be installed by the user. The clock in this unit functions the same as that built into the A2000, which is reviewed in Section 7-I.

A500 Power Supply Connector

The A500 power supply connector is similar to that of the C128. The pinout of the square 5 pin DIN connector is as follows:

PIN	SIGNAL
1	+5Vdc @ 4.3A
2	Shield Ground
3	+12Vdc @ 1.0A
4	Signal Ground
5	−12Vdc @ .1A

External Disk Interface Connector

The 23 pin D-type connector with sockets (DB23S) at the rear of the Amiga is nominally used to interface to MFM devices.

The second disk drive port is similar to the A1000, and is therefore compatible with the 1010 or the 1020 disk drive. The CPU will power one external 1010 disk drive.

External Disk Connector Pin Assignment

Pin	Name	Dir	Notes
1	RDY*	I/O	If motor on, indicates disk installed and up to speed. If motor not on, Identification mode. See below.
2	DKRD*	I	MFM input data to Amiga.
3	GND		
4	GND		
5	GND		
6	GND		
7	GND		
8	MTRXD*	OC	Motor on data, clocked into drive's motor on flip flops by the active transistion of SELxB*. Guaranteed setup time is 1.4 µsec. Guaranteed hold time is 1.4 µsec.
9	SEL2B*/SEL3B*	OC	A500:Select drive 2/A2000: Select drive 3.
10	DRESB*	OC	Amiga system reset. Drives should reset their motor on flip flops and set their write protect flip flops.
11	CHNG*	I/O	Note: Nominally used as an open collector input. Drive's change flop is set at power-up or when no disk is installed. Flop is reset when drive is selected and the head stepped, but only if a disk is installed.
12	+5V		270 ma maximum; 410 ma surge. When below 3.75V, drives are required to reset their motor on flops, and set their write protect on flops.
13	SIDEB*	O	Side 1 if active, side 0 if inactive.
14	WPRO*	I/O	Asserted by selected, write protected disk.

15	TKO*	I/O	Asserted by selected drive when read/write head is positioned over track 0.
16	DKWEB*	OC	Write gate (enable) to drive.
17	DKWDB*	OC	MFM output data from Amiga.
18	STEPB*	OC	Selected drive steps one cylinder in the direction indicated by DIRB.
19	DIRB	OC	Direction to step the head. Inactive to step towards center of disk (higher numbered tracks).
20	SEL3B*/ Not Used	OC	A500: Select drive 3/A2000: Not used.
21	SEL1B/SEL2B	OC	A500: Select drive 1/A2000: Select drive 2.
22	INDEX*	I/O	Index is pulse generated once per disk revolution, between the end and beginning of cylinders. The 8520 can be programmed to conditionally generate a level 6 interrupt to the 68000 whenever the INDEX* input goes active.
23	+12V		160 ma maximum; 540 ma surge.

Note: * in signal name denotes active low signal.

External Disk Connector Identification Mode

An identification mode is provided for reading a 32 bit serial identification data stream from an external device. To initialize this mode, the motor must be turned on then off. See pin 8, MTRXD* for a discussion of how to turn the motor on and off. The transition from motor on to motor off reinitializes the serial shift register.

After initialization, the SELxB* signal should be left in the inactive state.

Now enter a loop where SELxB* is driven active, read serial input data on RDY* (pin 1), and drive SELxB* inactive. Repeat this loop a total of 32 times to read in 32 bits of data. The most significant bit is received first.

External Disk Connector Defined Identifications

$0000 0000 – no drive present
$FFFF FFFF – Amiga standard 3.25 diskette
$5555 5555 – 48 TPI double density double sided

As with other peripheral ID's, users should contact Commodore Technical Support for ID Assignment.

The serial input data is active low and must therefore be inverted to be consistent with the above table.

External Disk Connector Limitations

1. The total cable length including daisy chaining must not exceed 1 meter.
2. A maximum of 3 external devices may reside on this interface (2 for the A2000).
3. Each device must provide a 1000 Ohm pullup resistor on every open collector input.

Full Bus Termination

Unlike the A1000 and the A500, both versions of the Amiga 2000 have an internal expansion bus, as a function of having an internal card cage.

Internal RAM Expansion on the A500

On the A500, memory at $C00000 is "slow" RAM (the processor is locked out by the custom chips) rather than fast RAM as suggested by A1000 external expansion. Thus, when ExecBase is transferred to $C00000 to free up chip RAM, there is no speed advantage. However, you would still be making real chip RAM available for other purposes. The B2000 functions as the A500 does in this regard.

EIA Ring Indicate Support

The A500, A2000 and B2000 support the RS232 RI lead to allow operation with modem standards. When the RI signal is asserted, the parallel port SEL line will be driven low. If this function is not desired, the RI lead should be disconnected in the modem cable.

Time of Day Clock

In the A500, the Time of Day clock is tied to the VSYNC signal rather than the power line. This results in the theoretical error of several minutes a day. For more precise timing, use the optional real-time clock.

In genlock mode, the genlock peripheral provides a 30 Hz V/Z signal, which results in the clock running half speed.

Light Pen

The light pen input on the A500 and B2000 has been moved to the second mouse port to allow use without a pass-thru mouse adapter. On a B2000, the light pen can be jumpered to port 0.

Monochrome Composite Video

The A500 and B2000 provide a full-bandwidth 16-level grey-scale composite video output. Color composite is available with an optional A520 composite color/rf video adapter.

Audio Filter Cut-out

The A500 and B2000 can cut out the anti-aliasing filter by programmatically turning off the "power on" LED. External bandwidth limiting to below 15 KHz will be required for most applications. This permits wider frequency response by using faster sampling rates.

A500 Reset

The A500 implements a "hard-wired" Control/Commodore/Amiga key reset rather than the "soft" A1000/A2000 keyboard reset. "Shut down" keyboard messages are not transmitted.

A2000 Expansion Bus IPL Lines

The A2000 does not run the processor IPL lines beyond the 86 pin MMU connector. Instead, additional interrupt request lines are allocated for future expansion devices. These lines are **not** supported by the current software.

Table 1-1 RAW KEY CODES

Raw Key Number	Keycap Legend	Unshifted Default Value	Shifted Default Value
00	` ~	` (Accent grave)	~ (tilde)
01	1 !	1	!
02	2 @	2	@
03	3 #	3	#
04	4 $	4	$
05	5 %	5	%
06	6 ^	6	^
07	7 &	7	&
08	8 *	8	*
09	9 (	9	(
0A	0)	0	)
0B	- _	- (Hyphen)	_ (Underscore)
0C	= +	=	+
0D	\ |	\	|
0E		(undefined)	
0F	0	0	0 (Numeric pad)
10	Q	q	Q
11	W	w	W
12	E	e	E
13	R	r	R
14	T	t	T
15	Y	y	Y
16	U	u	U
17	I	i	I
18	O	o	O
19	P	p	P
1A	[{	[	{
1B	] }	]	}
1C		(undefined)	
1D	1	1	1 (Numeric pad)
1E	2	2	2 (Numeric pad)
1F	3	3	3 (Numeric pad)
20	A	a	A
21	S	s	S
22	D	d	D
23	F	f	F
24	G	g	G
25	H	h	H
26	J	j	J
27	K	k	K
28	L	l	L
29	; :	;	:
2A	' "	' (single quote)	"

Raw Key Number	Keycap Legend	Unshifted Default Value	Shifted Default Value
2B		(RESERVED)	(RESERVED)
2C		(undefined)	
2D	4	4	4 (Numeric pad)
2E	5	5	5 (Numeric pad)
2F	6	6	6 (Numeric pad)
30		(RESERVED)	(RESERVED)
31	Z	z	Z
32	X	x	X
33	C	c	C
34	V	v	V
35	B	b	B
36	N	n	N
37	M	m	M
38	, <	, (comma)	<
39	. >	. (period)	>
3A	/ ?	/	?
3B		(undefined)	
3C	.	.	. (Numeric pad)
3D	7	7	7 (Numeric pad)
3E	8	8	8 (Numeric pad)
3F	9	9	9 (Numeric pad)
40	(Space bar)	20	20
41	BACK SPACE	08	08
42	TAB	09	09
43	ENTER	0D	0D (Numeric pad)
44	RETURN	0D	0D
45	ESC	1B	1B
46	DEL	7F	7F
47		(undefined)	
48		(undefined)	
49		(undefined)	
4A	-	-	- (Numeric Pad)
4B		(undefined)	
4C	Up Arrow	<CSI>A	<CSI>T
4D	Down Arrow	<CSI>B	<CSI>S
4E	Forward Arrow	<CSI>C	<CSI> A[1]
4F	Backward Arrow	<CSI>D	<CSI> @

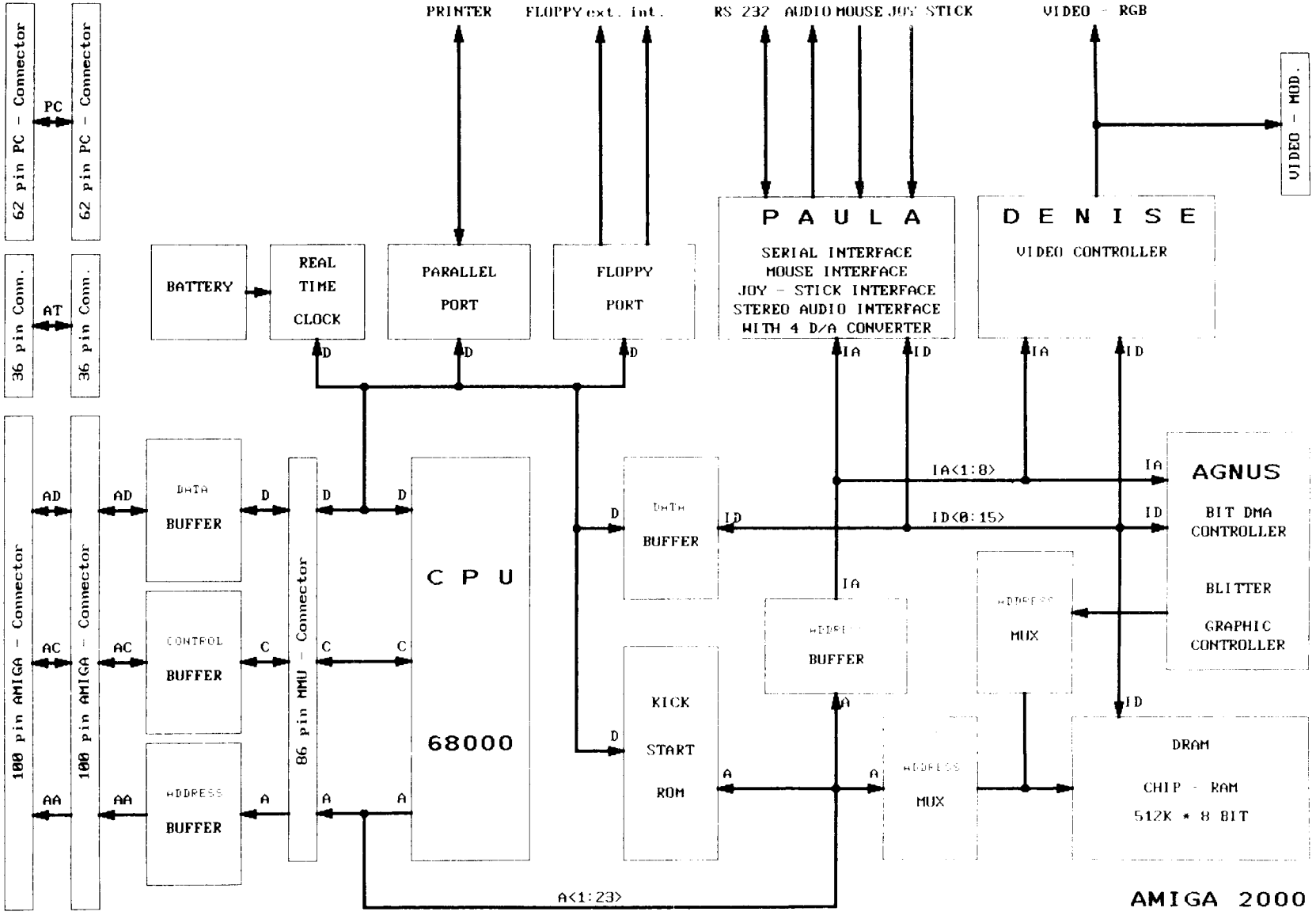
[1] In shifted Forward Arrow and Backward Arrow, note blank space after <CSI>.
<CSI> stands for Command Sequence Initiator.

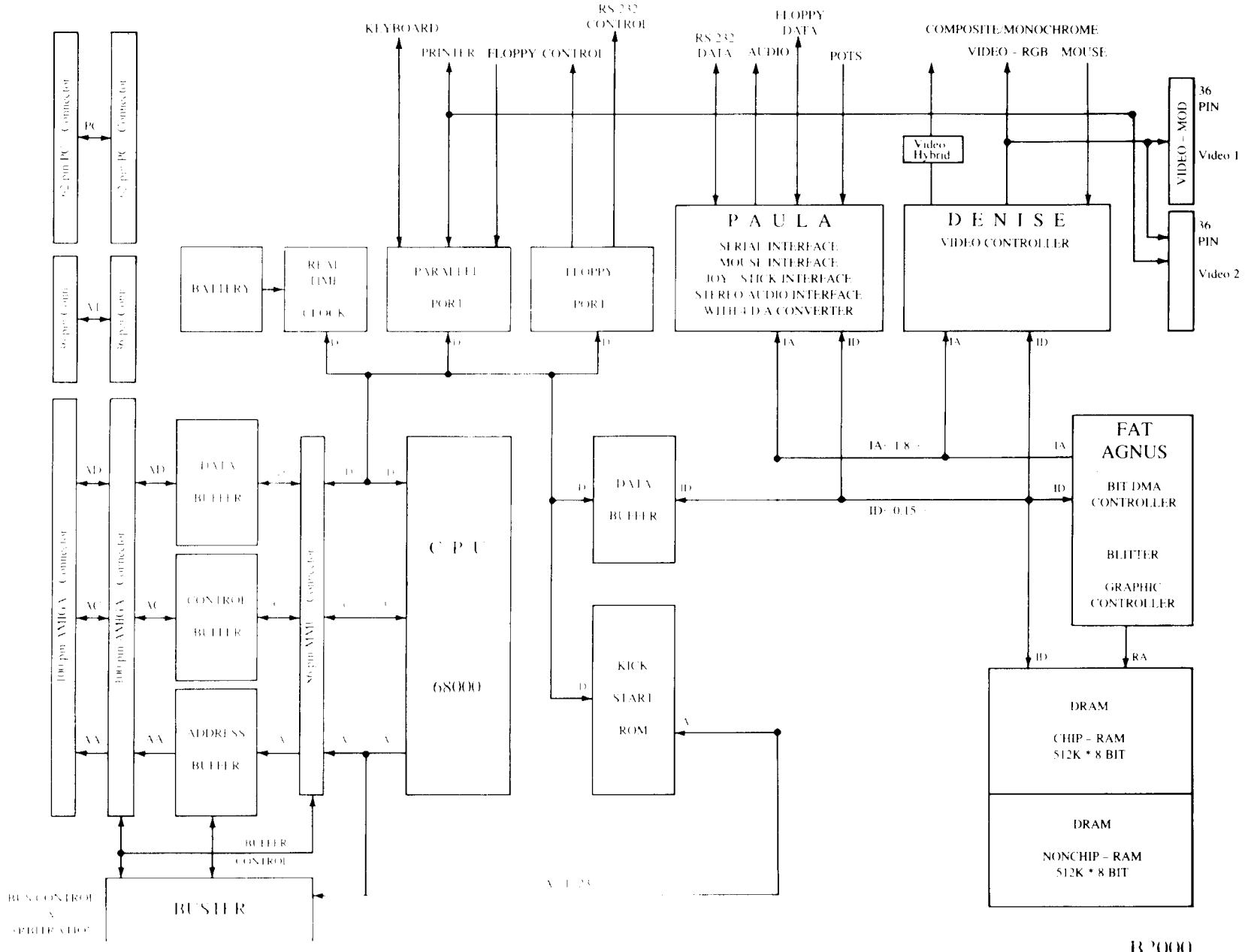
Raw Key Number	Keycap Legend	Unshifted Default Value	Shifted Default Value
50	F1	<CSI>0~	<CSI>10~
51	F2	<CSI>1~	<CSI>11~
52	F3	<CSI>2~	<CSI>12~
53	F4	<CSI>3~	<CSI>13~
54	F5	<CSI>4~	<CSI>14~
55	F6	<CSI>5~	<CSI>15~
56	F7	<CSI>6~	<CSI>16~
57	F8	<CSI>7~	<CSI>17~
58	F9	<CSI>8~	<CSI>18~
59	F10	<CSI>9~	<CSI>19~
5A	(	(	(
5B	)	)	)
5C	/	/	/
5D	*	*	*
5E	+	+	+
5F	HELP	<CSI>?~	<CSI>?~

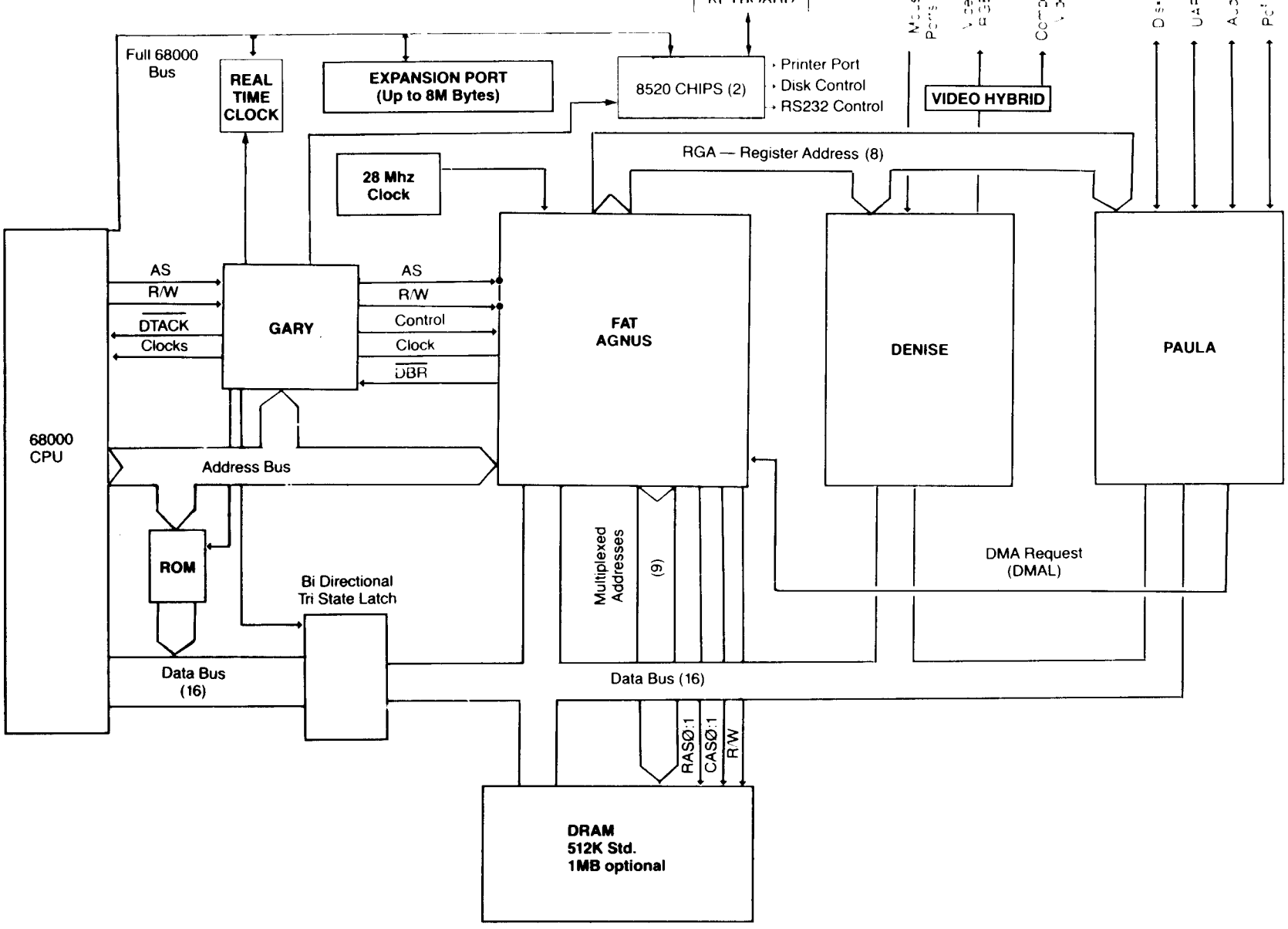
System Block Diagrams

INTRODUCTION

This section features system block diagrams for each new Amiga, the A2000, B2000 and A500, in that order.


PRINTER
FLOPPY ext. int.
RS 232 AUDIO MOUSE JOY STICK
VIDEO - RGB
VIDEO - MOD.
62 pin PC - Connector
PC
62 pin PC - Connector
36 pin Conn.
AT
36 pin Conn.
BATTERY
REAL TIME CLOCK
PARALLEL PORT
FLOPPY PORT
P A U L A
SERIAL INTERFACE
MOUSE INTERFACE
JOY - STICK INTERFACE
STEREO AUDIO INTERFACE
WITH 4 D/A CONVERTER
D E N I S E
VIDEO CONTROLLER
D
IA
ID
AD
AC
AA
100 pin AMIGA - Connector
100 pin AMIGA - Connector
DATA BUFFER
CONTROL BUFFER
ADDRESS BUFFER
86 pin MMU - Connector
C P U
68000
DATA BUFFER
KICK START ROM
ADDRESS BUFFER
ADDRESS MUX
IA<1:8>
ID<0:15>
AGNUS
BIT DMA CONTROLLER
BLITTER
GRAPHIC CONTROLLER
ADDRESS MUX
DRAM
CHIP - RAM
512K * 8 BIT
A<1:23>
AMIGA 2000


KEYBOARD
PRINTER
FLOPPY CONTROL
RS 232 CONTROL
RS 232 DATA
AUDIO
FLOPPY DATA
POTS
COMPOSITE/MONOCHROME
VIDEO - RGB
MOUSE
Video Hybrid
VIDEO - MOD
36 PIN
Video 1
36 PIN
Video 2
PC
AT
BATTERY
REAL TIME CLOCK
PARALLEL PORT
FLOPPY PORT
PAULA
SERIAL INTERFACE
MOUSE INTERFACE
JOY - STICK INTERFACE
STEREO AUDIO INTERFACE
WITH 4 D-A CONVERTER
DENISE
VIDEO CONTROLLER
100 pin AMIGA Connector
DATA BUFFER
CONTROL BUFFER
ADDRESS BUFFER
86 pin MMU Connector
CPU
68000
DATA BUFFER
KICK START ROM
FAT AGNUS
BIT DMA CONTROLLER
BLITTER
GRAPHIC CONTROLLER
DRAM
CHIP - RAM
512K * 8 BIT
DRAM
NONCHIP - RAM
512K * 8 BIT
BUFFER CONTROL
BUS CONTROL
BUSTER



A500 BLOCK DIAGRAM

Designing Hardware for the Amiga Expansion Architecture

INTRODUCTION

This section gives guidelines for designing hardware to reside on the Amiga expansion bus. The Amiga expansion bus is a relatively straightforward extension of the 68000 bus.

Hardware for the bus can be viewed as two categories: backplanes and PICs. Backplanes interface to the 86 pin connector of either another backplane or the Amiga itself. Backplanes buffer the bus and provide 100 pin connectors for PICs to plug into.

PIC is an acronym for plug-in card. A PIC is usually a card that plugs into the standard 100 pin Amiga connectors.

A sub-type of PIC is a combination of backplane and PIC integrated into one package. These combination products should follow all of the applicable backplane and PIC rules, especially auto-configuration.

Software never sees backplanes; all expansion hardware appears to the software as PICs.

WARNING

These specifications represent "worst case" design targets. Products that do not comply with these specifications can be expected to fail on worst case production units.

Following conservative design practices and allowing the widest safety margins is your best assurance against problems in the field.

EXPANSION ARCHITECTURE OVERVIEW

As shown in Figure 3.1, "Expansion Architecture Overview," the expansion bus is implemented as backplane (an expansion box) which accept PICs (boards). The recommended number of PICs to a backplane is five.

Due to timing considerations, it is not possible to daisy-chain more than two buffered backplanes without inserting wait states.

> **NOTE**
>
> You should also take extreme care in controlling signal radiation from your product, in order to pass FCC class B regulations.

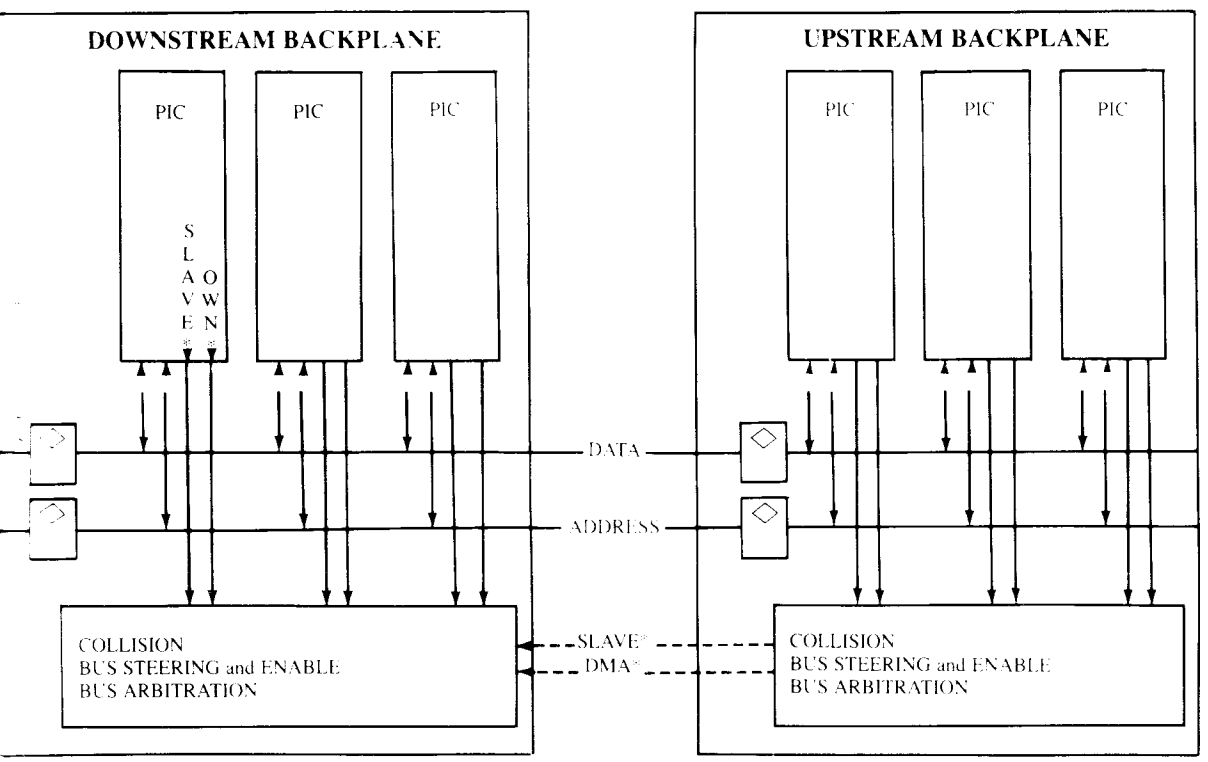


Figure 3.1. Expansion Architecture Overview

GLOSSARY

Active Active high signals are considered active when they are in the "one state" or "high state". Active low signals are considered active when they are "low" or in the "zero state". Active high signals do not have barred signal names. Active low signals do have barred signal names. Active means that the signal is

1. is true (non-barred) and is currently in the one state, or
2. is a barred signal name and is currently in the zero state.

An example is AS* (the * = bar). AS* is active when it is equal to zero. A counter example is the signal AS (the inverse of AS*), which is active when it is in the one state.

Auto Configuration The protocol (specified in this section) that Amiga uses to configure expansion cards into the system.

Downstream Downstream means closer to the Amiga. For instance, if two backplanes are daisy chained on the bus, the closer-in backplane is downstream from the further-out backplane. The concepts of upstream and downstream are important in determining which direction the address and data drivers should drive.

Master A PIC which is capable of initiating DMA cycles on the bus.

PIC A PIC is a plug-in card or a product which behaves in the system as a plug-in card. That is, it provides a resource that resides on the expansion bus, and follows the rules for auto-config, master protocol, slave protocol, etc.

Slave A slave is a PIC that can only respond to bus cycles. A slave cannot initiate bus cycles: in other words, it does not drive the address lines on the backplane, nor AS*, UDS*, LDS*.

Upstream Upstream means further away from the processor. For instance, all PICs are upstream from the buffers on the backplane that they are plugged into because the buffers are between the PIC and the Amiga.

ESIGN GUIDELINES
)R BACKPLANES

)llision Detection
rcuit

In this context, collisions are defined as any instance of two slaves attempting to respond to the same bus cycle.

All backplanes must have a collision detect circuit. The reason is that the PICs are auto-configurable and can be accidently instructed by software to respond to overlapping address spaces. Without collision detection, erroneous software can damage the hardware by causing bus contention.

Collision detect works in the following way: As soon as a PIC knows that it has been selected as the slave for this bus cycle, it asserts SLAVE* low and holds SLAVE* low until the end of the bus cycle (AS* going high).

The collision detect circuit (usually part of a PAL) detects whether more than one slave is responding and, if so, asserts BERR*. All data drivers on the expansion bus must be designed to enter high impedance mode whenever BERR* is active. Because data drivers are not turned on until S4 (ASDELAYED* active), BERR* will have disabled the drivers before the contention can begin.

Note that in order to detect all cases of multiple slave response, the circuit must watch A23-A19 for Amiga address spaces and also watch SLAVEIN* from the next box out. See discussion of the example schematic for specific PAL equations that implement collision detect.

Because BERR* is listened to by all PICs, it will in some systems be heavily loaded, so it should be driven with a hefty open collector or tri-state driver. Each backplane should provide a 1000-ohm pull-up resistor on BERR*.

us Arbitration Logic

The bus arbitration logic is based on the 68000 BR*, BG*, BGACK* protocol as described in the 68000 manual. In order to avoid metastable states in the backplane latches, all changes in state of the BR* lines from the PICs must be clocked by the rising edge of 7M.

The example design gives our current recommended bus arbitration logic. Refer to the ARBITRATE PAL equation in Table 3-3.

Buffer Control Logic

The buffer control logic controls output enable and direction of the bidirectional tri-state bus drivers. See the STEERING PAL equation, Table 3-2.

Data Driver Timing

It should be noted that the backplane drivers must not turn on until the rise of S4 during a read. This is okay because data from the Amiga internal RAMs is not valid during S4 anyway, so nothing is to be gained by turning the data buffers on earlier.

Clock Buffers, 7M, and ASDELAYED*

There are three clocks coming from the Amiga. These are CDAC, C1*, and C3*. The backplane must generate 7M (equivalent to the Processor clock) by the following equation: 7M = C1* XNOR C3*.

THE PROTOCOLS

The bus protocols are basically the same as standard 68000 protocols; however, the timing margins are tighter due to the potentially long paths of Amiga and PICs talking to each other across two buffered backplanes.

One unusual feature is that when you are doing a DMA transfer into or out of the Amiga display RAM (the half megabyte starting at address 000000), the DTACK* circuit will synch the master up with C1. Because C1 is twice as slow as 7M, there are two possible phase relationships between C1 and the beginning of the DMA bus cycle. If AS* is asserted during the last quartile of C1 (C1 low and C3 low, see Fig. 3.2, System clock timing diagram), we call this an "in sync" bus cycle, and DTACK* is given in time to do a normal 4-clock (7M) bus cycle. (Note: Occasionally, DTACK* is delayed due to contention with the graphics chips, but that does not matter in this discussion.)

However, DTACK works differently if the DMA controller asserts AS* in the other phase. In the second quartile (C1 high and C3 high), the DTACK* circuit holds off DTACK* long enough to insert one wait state, thus synching up the "out of sync" bus cycle.

Read or Write Cycle With Amiga as Master

Since the Amiga bus master is a 68000, the bus cycle is a 68000 cycle. However, the responding slave does not pull DTACK*. Our internal circuitry pulls DTACK* unless the slave pulls XRDY low.

Also, the slave (PIC) must pull its SLAVE* output low as soon as it is selected, and at the end of the cycle, disassert SLAVE* when AS* goes away.

Read or Write Cycle with a PIC as Master

A PIC as master must drive the bus using the same protocol as the 68000. Some of the timing margins must be better than those from the 68000, because the PIC is driving through several levels of buffers, and the Amiga logic is designed to the 68000 (8 megahertz part) specs. Specific timing requirements can be found in the tables later in this section.

Bus Arbitration

The bus arbitration scheme is based on the 68000 BR*,BG*,BGACK* protocol. PICs are required to assert BR* clocked by the rising edge of 7M. This makes it less expensive to design bus arbitration logic that will be reliable. Specifically, synchronous arbitration logic can be clocked on 7M without danger of going metastable.

SYSTEM LEVEL ORGANIZATION (AND IDIOSYNCRASIES)

Address Override (OVR*)

Pin 17 OVR* can only be used in between address $200000 and A0000, and implies you have to supply your own DTACK*. OVR* is **not** supported for the purpose of disabling system decoding in the C00000 to DFFFFF range. Worst case 68000 timing requires modifications to the system decode gate array to accomplish this reliably. Other uses of OVR* are not supported.

INTERRUPTS

USE INT2* OR INT6* (DON'T PULL IPL0*-IPL2*)

There are two interrupt input lines on the Amiga: INT2* and INT6*. INT2* = pin 19, INT6* = pin 22, these lines assert levels 2 and 6 to the processor.

Do not assert the IPL0* thru IPL2* lines, because they are already driven by internal logic.

INTERRUPT LATENCY— BLITTER, MASKED INTS

Interrupt latency on the Amiga is highly application software dependent, this is because the Blitter can be operated in "nasty mode" at the software's option. If the blitter is "nasty" and is given a lot of work to do, the processor receives very few memory cycles, so the interrupt latency will suffer.

The software can also mask out interrupts using on-board interrupt control logic.

VPA Is Not Recommended

We recommend that you design your peripherals to run asynchronously on the 68000 bus, that is, a slow peripheral should be memory mapped and use pulling XRDY low as a means of making the 68000 run a slower cycle. The use of XRDY to delay DTACK is discussed elsewhere in this document.

We do not recommend using VPA. If you decide to use VPA, you must pull OVR* low 30ns before asserting VPA* low. Pulling OVR* low will tri-state VPA* in the current design PAL, thus allowing your logic to drive VPA*. Pulling OVR* will also prevent DTACK* from being asserted by the PAL. However, this will not disable the on-board 8520 CIA chips.

If your slave uses the VPA VMA protocol to be synchronous with the 68000's E clock, you must only use addresses in which A12 and A13 are high. This is because we have synchronous ports on board which are activated by (A12* AND VMA), also (A13* AND VMA).

Do Not Use Pins Marked EXP

Do not drive or load pins marked EXP or RESERVE.

TIMING GENERAL DISCUSSION

Timing specifications are listed in Table 3-1.

There are two main problems to be dealt with in the expansion architecture timing: propagation delays and skews in the clock, address, data, and control paths. The timing is tight; thus, we recommend using FAST and AS parts to buffer these lines. To guarantee meeting the timing requirements, you must be careful to not exceed the recommended operating conditions of the parts you chose, for example the capacitive loading. In calculating your loading, note that all PICs are specified to present no more than two "F" loads plus minimal trace capacitance to each connector pin. Backplanes are specified to present no more than one "F" load plus trace capacitance to the Amiga. Do not use "typical" numbers; reliable systems can be built by using "worst case" numbers.

xpansion Notes

1) The loading, buffering and layout requirements specified for the A1000/A500 expansion connector must be strictly followed for reliable operation. Unbuffered devices and bus line extension are known problem areas.

2) Unbuffered daisy-chaining of multiple external expansion devices is not supported.

3) The A500 provides only nominal amounts of power for expansion devices. All devices having significant power requirements are expected to be self-powered and should **not** make connections to the power pins on the expansion connector.

DESIGN GUIDELINES FOR PICs

Auto Configuration

All PICs implement the auto-configuration protocol. The auto config protocol is designed so that system auto-config software can interrogate the PICs ID locations, build a system table of the installed PICs, and place the PICs in the 68000 memory space.

General Description of Auto Configuration

If it is difficult to imagine how to implement this protocol while it's being described, don't worry. The design requires one PAL, one latch, and one address match circuit. Complete details are given in the example design.

Upon reset, all PICs come up in the unconfigured state. In the unconfigured state, the PIC responds to the 64 kilobyte address space starting at location E80000, if CONFIGIN* is active to the PIC. If CONFIGIN* is not active, the PIC does not respond to any bus cycles.

The processor comes out and reads nibbles of ID data on D15-D12 from the PIC. The table of ID data and the locations of control latches is detailed later in this section. This data includes such things as size of address space required, manufacturer's product number, and whether to add the PIC to the free memory pool (if it is a memory PIC.)

Under normal conditions, the processor determines how much address space the PIC requires and then loads the PIC's address latch with an appropriate base address. This permanently relocates the PIC at its new address (until Reset), and passes CONFIGOUT* out to the next PIC's CONFIGIN*, whereupon the process is enacted again until all PICs are configured.

The smallest unit of memory that a PIC can ask for is 64 kilobytes. The largest is eight megabytes. All PICs should be designed to be based on boundaries that match their space requirements; for example, one megabyte PICs should be designed to reside on one megabyte boundaries (match circuit matches A23-A20). There are two exceptions to this rule, however. Four megabyte PICs must be capable of being placed on four megabyte boundaries, as well as at hex 200000 and at hex 600000. Eight megabyte PICs should be capable of being placed on eight meg boundaries and at hex 200000. This

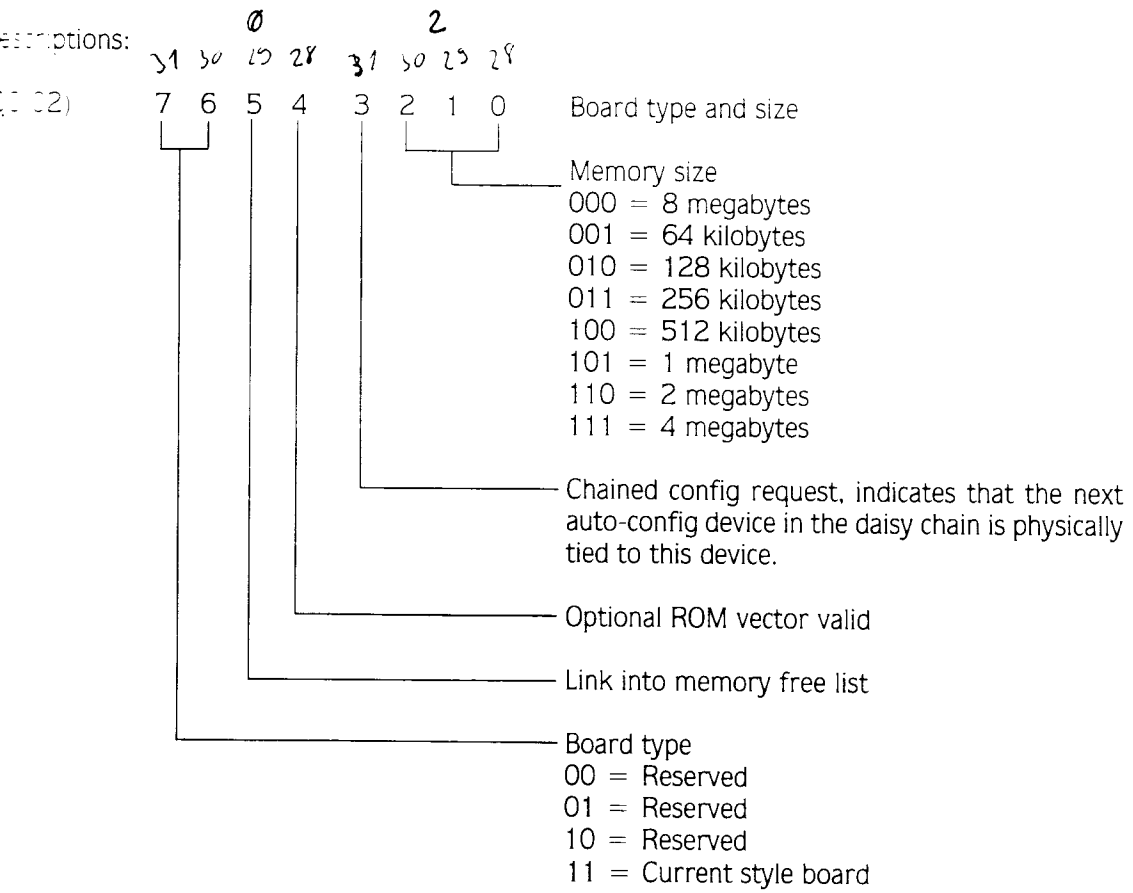
requirement is because the eight megabyte space reserved for expansion in the current machine begins at hex 200000 (See auto-config notes below).

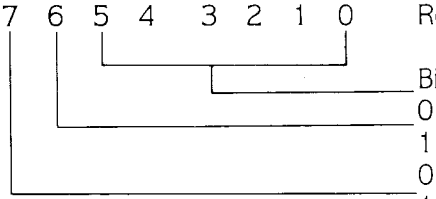
Auto-Config Notes

1) There is currently no provision for 6MB PICs. Designers of 8 MB memory boards should consider auto-configs as two PICs to allow partial loading flexibility.
2) PIC size/alignment rules are subject to change. If so, bit(s) will be defined to allow a PIC to specify that it is more flexible than the old rules require.
3) The address map is subject to change. A PIC should assume that it may be placed anywhere in the address space.
4) All expansion devices are strongly encouraged to use the auto-config protocols. Assignment of fixed I/O addresses is subject to negotiation.

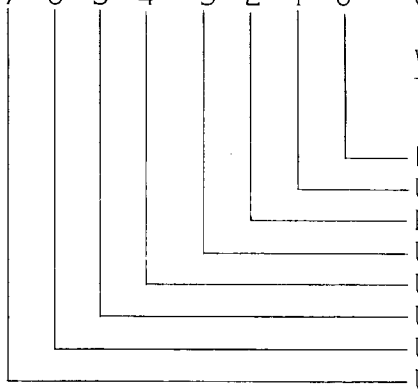
ddress Specification Table

ll nibbles except 00, 02, 40 and 42 should be inverted.



(04/06)	7 6 5 4 3 2 1 0	Product number, this number is defined by the manufacturer of the board and is used by auto-config software to initialize drivers for the board.
(08/0A)	7 6 5 4 3 2 1 0	Reserved, must be as specified  Bits are currently zero 0 means this board can be shut up 1 means this board cannot be shut up 0 means any space okay 1 means preference to be put in the 8 Meg space
0C/0E)	7 6 5 4 3 2 1 0	Reserved, must be 0
10/12)	7 6 5 4 3 2 1 0	Mfg # high byte
14/16)	7 6 5 4 3 2 1 0	Mfg # low byte; These 2 bytes are assigned by CBM. They are used by the auto-config software to initialize drivers for boards.
18/1A)	7 6 5 4 3 2 1 0	Optional serial number, byte 0 (msb)
1C/1E)	7 6 5 4 3 2 1 0	Optional serial number, byte 1
20/22)	7 6 5 4 3 2 1 0	Optional serial number, byte 2
24/26)	7 6 5 4 3 2 1 0	Optional serial number, byte 3 (lsb)
28/2A)	7 6 5 4 3 2 1 0	Optional ROM vector high byte
2C/2E)	7 6 5 4 3 2 1 0	Optional ROM vector low byte. If the 'ROM addr valid' bit (4 of nibble 0) is set, then these 2 bytes are the offset from the board's base address at which the start of the ROM code information is located (e.g., the hard disk driver). If the bit it not set, then these 2 bytes have no meaning.
30/32)	7 6 5 4 3 2 1 0	Reserved, read must be 0; write resets base address register
34/36)	7 6 5 4 3 2 1 0	Reserved, must be 0
38/3A)	7 6 5 4 3 2 1 0	Reserved, must be 0
3C/3E)	7 6 5 4 3 2 1 0	Reserved, must be 0

(40/42) 7 6 5 4 3 2 1 0 — Optional control status register

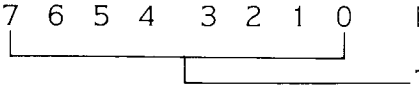


	Write	Read
7	Interrupt enable	Interrupt enable
6	User definable	don't care
5	Local reset	must be 0
4	User definable	don't care
3	User definable	INT2 pending
2	User definable	INT6 pending
1	User definable	INT7 pending
0	User definable	I am pulling INT

(44/46) 7 6 5 4 3 2 1 0 — Reserved

Write	Read
Not defined	must be 00

(48/4A) 7 6 5 4 3 2 1 0 — Base address register, write only



These bits are compared with A23 through A16 (or fewer) to determine the base address of this board.

(4C/4E) x x x x x x x x — Optional "shut up" address, a write to this address will cause the board to pass its config out and then never again respond to any address. RESET will re-enable the board. The actual address that has this effect is 4C. A write to 4E is ignored. This is write only.

(50/52)	7 6 5 4 3 2 1 0	Reserved, must be 00
(54/56)	7 6 5 4 3 2 1 0	Reserved, must be 00
(58/5A)	7 6 5 4 3 2 1 0	Reserved, must be 00
(5C/5E)	7 6 5 4 3 2 1 0	Reserved, must be 00
(60/62)	7 6 5 4 3 2 1 0	Reserved, must be 00
(64/66)	7 6 5 4 3 2 1 0	Reserved, must be 00
(68/6A)	7 6 5 4 3 2 1 0	Reserved, must be 00
(6C/6E)	7 6 5 4 3 2 1 0	Reserved, must be 00
(70/72)	7 6 5 4 3 2 1 0	Reserved, must be 00
(74/76)	7 6 5 4 3 2 1 0	Reserved, must be 00
(78/7A)	7 6 5 4 3 2 1 0	Reserved, must be 00
(7C/7E)	7 6 5 4 3 2 1 0	Reserved, must be 00

Note: The actual reserved values will be FF rather than 00, because the system will invert them. See the section on reading I/O locations for more information.

EXAMPLE BACKPLANE DESIGN

We have designed a backplane as an example implementation of our expansion architecture. This section is a detailed description of the schematic of that backplane. The schematic appears as Figure A-1 in Appendix A.

Backplane Schematic Overview

While reading this section, refer to the backplane schematics for the A2000 and PALS to see what is being described. The B2000 uses a gate array to handle steering; however, this example backplane design is functionally equivalent, and should be useful in that sense.

The bus comes in on the left from the processor via J10. Note that both the data bus and address bus are buffered through bi-directional buffers. The buffers are bi-directional in order to allow external DMA controllers.

The Bus Buffers and Their Control Logic

This subsection describes the bus buffers, their timing and control logic. In this discussion, "upstream" means away from the processor, and "downstream" means toward the processor. For instance, if you daisy chain two devices on the bus, the further away of the two is "upstream" from the closer (downstream) device.

Throughout this document, there are references to signals going active. Active is defined in the glossary for this section.

The Address and Control Buffers

The address lines, function codes, UDS*, LDS*, R/W, and AS* are all buffered in the same manner by 74F245s. Their buffer direction is determined by DMAOUT. They are enabled by ADDR_OE* (address output enable bar).

Generating DMAOUT

This section explains the PAL equation for DMAOUT found in the STEERING PAL equations. (Table 3-2, later in this section).

DMAOUT active means that the current bus master is upstream of the buffers. Since the buffers are at the extreme downstream end of this backplane, the master is either on this backplane or upstream from this backplane. Thus when DMAOUT is high, the drivers drive the address and control lines downstream (toward the Amiga).

The PAL equation for DMAOUT is very straightforward:

DMAOUT = DMAIN + OWN

DMAIN is active when the bus master is upstream from this backplane. So when DMAIN is active, DMAOUT must go active.

OWN* is the wire OR'ed signal which means that this backplane has the current bus master. Thus, because all PICs on this backplane are upstream from the address (and data) buffers, DMAOUT must be active when OWN (or OWN*) is active.

enerating ADDR_OE*

This section explains the PAL equation for ADDR_OE*. Refer to the STEERING PAL equation to see the equation (AOE).

ADDR_OE* is active (enabling the address drivers) most of the time. It only disables the drivers when ownership of the bus is changing (for example, a new master takes control). At these transition times, ADDR_OE* is inactive so that the tri-state drivers will not fight the drivers on the next backplane while they are changing direction.

Refer to the equation for AOE in the STEERING PAL equation (Table 3-2). AOE = ADDR_OE* inverted. The inverter is in the output stage of the PAL.

BGACK is asserted (BGACK* pulled low) by all bus masters (except the 68000) when they are the current master, so ADDR_OE* is active when BGACK is active.

The term (BG* * DMAOUT*) is true most of the time that the 68000 owns the bus. However, when the 68000 is about to give up the bus, BG* will go active and thus (BG* * DMAOUT*) will go inactive. It is important that the address drivers remain on until the end of the final 68000 bus cycle when the 68000 is giving up the bus, so the term AS holds AOE active when BG goes active during the bus cycle.

AS does not last quite long enough, so ASQ90 (which is a slightly delayed AS) holds AOE active long enough to finish the cycle.

ne Data Buffers

This section describes when and why the data drivers are turned on and off. It also describes control of data direction.

enerating DBOE*

Refer to the STEERING PAL equation for DBOE.

Note that all the bus drivers are enabled for every bus cycle unless BERR* is asserted. This allows for easier use of bus-monitoring tools such as state analyzers.

It is fairly difficult to avoid tri-state fights on the data buffers. In order to get data out to dynamic RAM PICs at an early enough time, we do not use the data strobes to enable the data drivers, because these strobes can go active very late in a write cycle.

On a read cycle we use the data strobes, so that in case the cycle turns out to be a Read-Modify-Write cycle, the drivers will be turned off (to avoid tri-state fight) while the R/W line is changing state.

Refer to the PAL equation for DBOE in the STEERING PAL appendix. The term (AS * RD*) turns on the drivers for all write cycles, including the write portion of Read-Modify-Write cycles. Note that since AS turns off the data drivers, the data hold time is not guaranteed beyond AS going inactive, so it is poor design practice to try to use the rising edge of AS*, UDS*, or LDS* to latch data.

The terms (UDS * RD * ASQ) and (LDS * RD * ASQ) turn on the drivers for all read cycles. The UDS and LDS turn off the drivers in the middle of a Read-Modify-Write cycle.

The ASQ (ASDELAYED equivalent) keeps the data buffers from turning on until after there has been enough time for the collision detect circuit to assert BERR* low and thus disable the data drivers before they fight (see collision detection).

Generating D_TO_PROC*

The inverse of the D_TO_PROC* signal is called D2P in the PAL equation.

Collision Detection

Each backplane or device that passes the bus or allows more than one slave device must have a collision detect circuit. This circuit will usually be implemented in a PAL. This circuit must detect any instance of two slaves responding to the same bus cycle and assert BERR* immediately upon detecting such an error.

The collision circuit has an input (see schematic) SLAVEIN* which is passed from the upstream backplane or device (if any is present). If no upstream device is present, the pull-up resistor will hold SLAVEIN* inactive (high). SLAVEIN* tells the circuit whether or not an upstream PIC is responding to the current bus cycle as a slave.

The circuit also has one input for each slot on this backplane. If any PIC on this backplane is responding as a slave, the corresponding SLAVEn* will be active.

By the way, don't worry about two slaves colliding on the upstream of the backplane; that backplane has a collision detect circuit of its own.

Thus, each of the seven product terms indicates that a collision is not happening at this time. Only one of them needs to be true to know that a collision is not happening at this time.

Bus Arbitration Circuit

The bus arbitration circuit's main job is to determine which PIC will receive BG* active (Bus Grant) when the 68000 asserts BG*. The circuit we recommend does this based on priority, where the closest PIC to the 68000 is the highest priority. You could implement something fancier as long as only one PIC owns the bus at a time.

PICs are only allowed to assert BR* off the rising edge of 7M. This allows the bus arbitration circuit to operate synchronously, clocked by the rising edge of 7M.

The output of the bus arbitration circuit only changes when the 68000 changes the state of BG*. If the 68000 is asserting BG*, the arbitration circuit passes BG* active to the highest priority active requester. When the 68000 disasserts BG*, the arbitration disasserts BG* also. Therefore no PIC has a grant.

RES* and RESB*

Note that there are two reset lines going to every PIC, RES* on pin 53 and RESB* on pin 94. The RESB* line is intended to be the normal reset input to the PIC. All normal PICs will use this line as an input, so it is buffered.

RES* is intended only to be used by those PICs which are designed to have the capability of resetting the system. Normal PICs will not drive nor load this line. Note that because RES* is not buffered, it can reset the Amiga, as well as resetting all PICs (via RESB*).

CONFIG_IN* CONFIG_OUT* Daisy Chain

The CONFIG_IN* signal will be passed to CONFIG_OUT* at the appropriate time if there is a PIC plugged in the slot. On this backplane, we have used 74LS32s to pass CONFIG_OUT* to the next slot if there is no PIC. The pull down resistor allows the CONFIG_IN* signal to pass directly through the gate to CONFIG_IN* of the next slot if there is no PIC installed, thus bypassing the empty slot. If a PIC is installed, the PIC's CONFIG_OUT* driver overrides the pull down resistor.

Another method that would work is to use special pins on the connector at pins 11 and 12, such that 11 and 12 short to each other when there is no PIC inserted in the connector. This would eliminate the need for the 74LS32 gates.

The collision circuit also monitors A23 through A19 and OVR* on the bus, so that the internal reserved address spaces of the Amiga can be checked. An access to any of the internal address spaces will make the Amiga respond as the slave unless OVR* (override) is asserted.

Any two slave responses on the same cycle constitute a collision.

Refer to the COLLISION PAL equation in Table 3-5 for this discussion.

nerating the PROC rm

Before generating the collision detection equation, we must make the equation that detects whether the Amiga processor board is responding to this cycle as a slave. This signal is called PROC internally to the PAL. While it comes out on pin 18, it is not used external to the PAL.

The term BAS * /A23 * /A22 * /A21 * /RESET * /OVR will be true when the processor board memory is responding to the 2 megabyte space starting at hex 000000.

Similarly, the next term will be true when the processor board is responding to the 2 megabyte space that starts at hex A00000.

The next term detects the processor board responding to the 2 megabyte space starting at C00000.

The next term detects the processor board responding to the 1/2 megabyte space starting at E00000.

And the last term detects the proc board responding to the 1/2 megabyte space starting at F80000. This takes care of all the spaces used by the processor board.

nerating NOTCOLIS

Why the inverted name? We would have preferred to call this signal / COLLISION but our PAL assembler does not allow a NOT sign in the name on the left side of the equal sign. NOTCOLIS goes out through the output inverter and becomes/NOTCOLIS which is logically equivalent to NOTNOTCOLIS = COLLISION, so NOTCOLIS being true inside the PAL will make COLLISION false outside the PAL.

Now that PROC will tell us when the responding slave is inside the Amiga, we are ready to do collision detection.

In our example, we have seven possible slaves to keep track of. They are the Amiga board (PROC), five PICs on this backplane, and SLAVEIN* from the upstream backplane or device. If six of the seven are inactive at all times, we know that no two are active at the same time.

Because the slave lines go inactive between bus cycles, there should not be a case of one slave going active before the previous one went inactive.

ACKPLANE TIMING ENERATION

Clock Buffers

The clock buffers for C1*, C3*, and CDAC were chosen for minimum propagation delay and minimum skew. Notice that buffered clocks are passed to the 100 pin edge connectors, but that the unbuffered clocks are passed to the 86 pin connector that goes on to the next box in order to minimize propagation delay to the next backplane.

enerating 7M

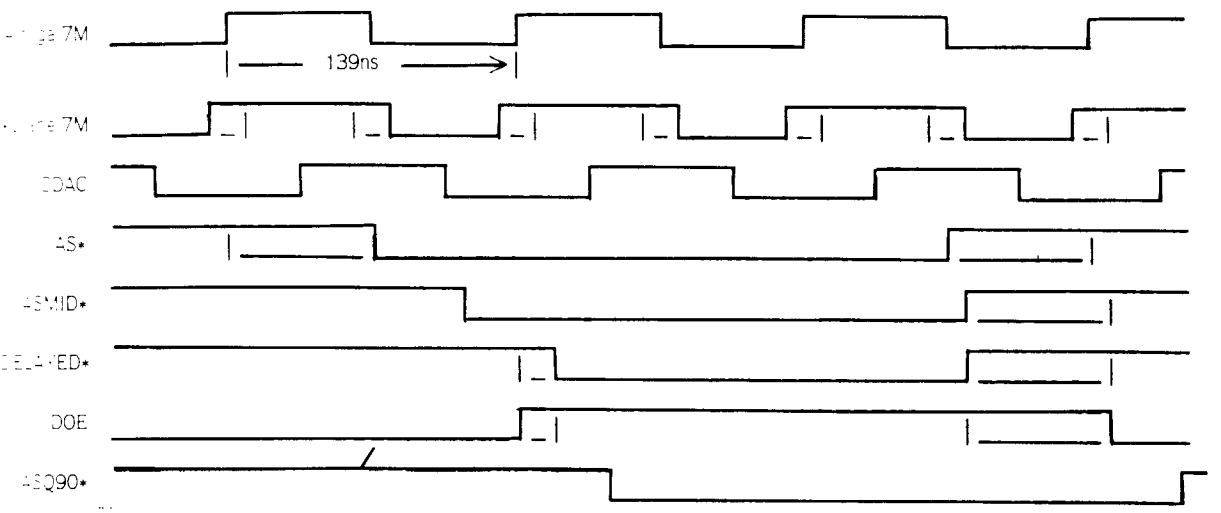
We generate 7M (equivalent to the processor clock) by:

7M = C1* XNOR C3*

This yields a 7.16Mhz clock which is used to generate ASDELAYED*, DOE, and ASQ90*. 7M is also passed to the PICs on pin 92 of the edge connectors, so they will have a cheap clock for accessing the bus.

OE, ASDELAYED*, SQ90*

DOE (Data output enable) and ASDELAYED* are the compliment of each other. ASDELAYED* is used in the steering PAL (ASQ = ASDELAYED in the PAL equations) to time turning on of the data drivers during a read cycle. DOE is passed to the PICs on pin 93 of the edge connectors, to tell the PICs when to turn on data drivers during a read cycle.



Backplane Timing Signals

EXAMPLE PIC DESIGN

This section is a description of the schematic for a small 16 kilobyte RAM board that we designed as our first test PIC for the expansion architecture. The schematic for this board is Figure A.2, in Appendix A. It is valuable as an example because it implements all of the basic features of a slave PIC.

The PIC at System Startup

The heart of auto-config is in U1 (address register), U2 (address comparator), and U3 (ID PAL and control PAL).

When the board comes out of Reset, CONFIG_OUT* is inactive, and does not pass the config token on to the next PIC. CONFIG_IN* may or may not be active at first. If it is not active, the board will not respond to any bus cycles. For instance, we can see at U11 that SLAVE* is disabled when CONFIG_IN* is inactive (high), because this does not allow BOARD_SEL* to go active.

In turn, BOARD_SEL* is an input to U3, the control PAL. Without BOARD_SEL*, all ten of the PAL outputs are held inactive (see PAL equations for test ram).

Reading the ID Locations

Eventually, during execution of the auto config code, CONFIG_IN* will be asserted to this PIC between bus cycles (AS* inactive). Notice that the address latch is tri-stated off so that the pull-up and pull-down resistors are inputing a pattern of E8 to the address comparator. When the backplane addresses E8xxxx, this board will now respond because CONFIG_IN* is active but CONFIG_OUT* is not yet active. In other words, CONFIG_IN* is enabling board select, and CONFIG_OUT* has not yet allowed the address latch to move the board to a different address space.

Notice that whenever BOARD_SEL* goes active, SLAVE* will go active unless SHUT_UP_FOREVER is latched active. SHUT_UP_FOREVER* is a feedback latch in the PAL. It is only set by the software if the board cannot be configured into the system (for instance, if the user has plugged in too many large address space PICs and there is no room left for this one).

If you analyze the PAL equations for BD15 through BD12, you will see that their data drivers turn on for all reads ANDed with BOARD_SEL active, until CONFIG_OUT* is set active (or some exception happens such as reset, bus error, or shutup).

By the way, if you're not used to PALs, it's normal old Boolean: * means AND, / is negation, + is OR, IF(term) means "If the term evaluates to TRUE then turn on the tri-state driver".

Further analysis of the BD15-BD12 equations will show that almost all addresses put out ones; however, remember that most of the nibbles are inverted because the spec says they have to be. The inversion makes it possible to implement the codes in active low PALs; it is just a cost reduction.

Analysis of the equations shows that the only nibbles (we don't care about above HEX 80) outputting any zeros are:

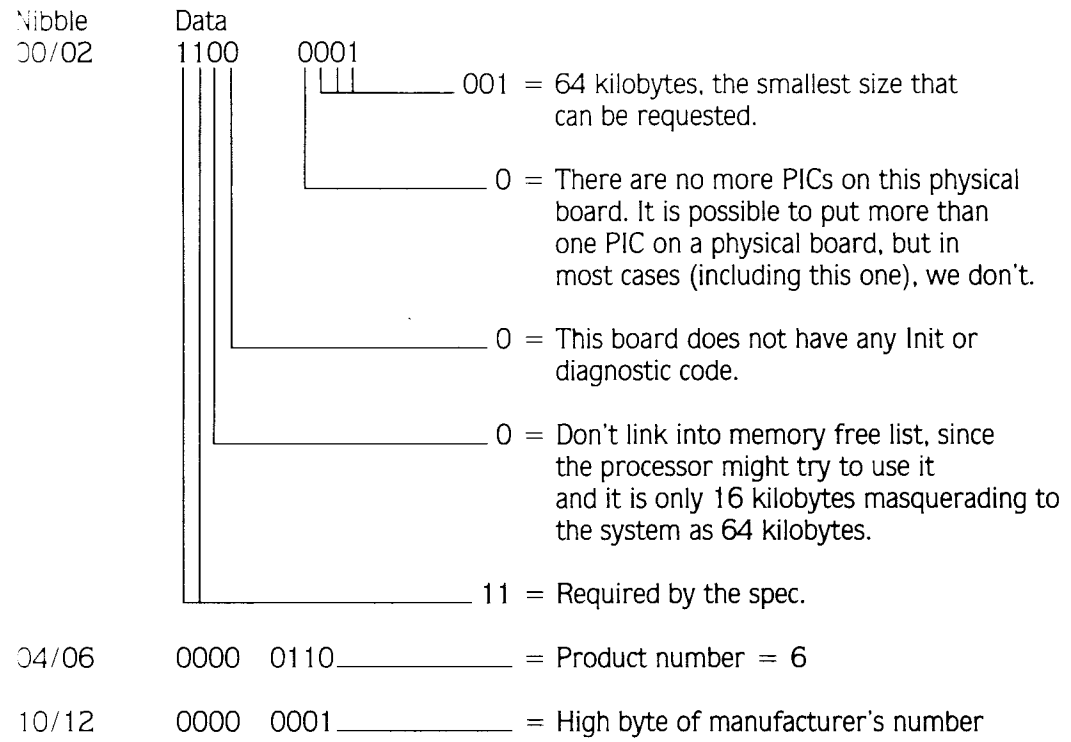
```
00/02   1100 0001
04/06   1111 1001
10/12   1111 1110
40/42   0000 0000
```

To interpret this code, we need to remember that the spec says that all nibbles get inverted except 00, 02, 40, and 42. So our new table looks like this:

```
00/02   1100 0001
04/06   0000 0110
10/12   0000 0001
40/42   0000 0000
```

And all the other nibbles that were ones are now inverted to zeros.

...strate, let's look at what these codes mean:



14/16 0000 0000_____ = Low byte of manufacturer's number

40/42 0000 0000_____ = Because this PIC does not generate INTs

When you want to program your own ID PAL, just work back to the equations. First determine what ID pattern you need by reading about the nibbles in the spec. Write down a table of ones and zeros. Invert all of these except nibbles 00, 02, 40, and 42. Then, doing one data line at time, write a product term for each binary zero that you want to output from the ID PAL.

Passing CONFIG_OUT*

The equations for CONFIG_OUT* in this implementation make two feedback latches in the PAL. The first latch PRE_CONFIG_OUT* is set during the bus cycle in which the processor does a write to the address register. In fact, in this design the rising edge of PRE_CONFIG_OUT latches the final Address value into the address latch.

The second latch outputs CONFIG_OUT*. This latch goes active after AS* goes inactive at the end of the bus cycle in which the new address was written. Notice that CONFIG_OUT* enables the address latch U1, so it now provides the new address range to the comparator.

CONFIG_OUT* enables the next PIC in the chain, and remains active until a system reset or power down occurs.

TABLE 3-1—TIMING SPECIFICATIONS

Timing Requirements for Backplane

TIMING REQUIREMENTS FOR BACKPLANE				
Num	Characteristic	Min	Max	Unit
1	AS* UDS* LDS* Delay	2	8	ns
2	Address 23-1 delay	2	8	ns
3	7M(S4 RISE) to Data Enable during Read	0		ns
4	7M (S4 RISE) to Data Valid		35	ns
5	Data 15-0 Delay to Output		8	ns
6	SLAVEIN or SLAVE to SLAVEOUT Delay	0	25	ns

ming Requirements for PIC

TIMING REQUIREMENTS FOR PIC AS SLAVE (RD & WR CYCLES)				
[illegible]	Characteristic	Min	Max	Unit
[illegible]	AS* low to SLAVE* Low	0	35	ns
[illegible]	AS* high to SLAVE* high	0	50	ns
[illegible]	AS* low to XRDY low (to insert wait)	0	60	ns
[illegible]	Read Data Valid to local 7M low (S7)	60		ns
[illegible]	AS* low to OVR* low	0	50	ns
[illegible]	AS* high to OVR* high	0	50	ns

TIMING REQUIREMENTS FOR PIC AS MASTER (RD & WR CYCLES)				
[illegible]	Characteristic	Min	Max	Unit
[illegible]	7M high(S2) to AS* low	0	67	ns
[illegible]	Address 23-1 Valid to AS* low	30		ns
[illegible]	7M high (S4) to Data Valid Wr Cycle		0	ns

ming to Backplane

TIMING TO BACKPLANE				
[illegible]	Characteristic	Min	Max	Unit
[illegible]	AS* Low to CDAC Low (Setup)	20		ns
[illegible]	AS* High to CDAC High (Setup)	20		ns

ming to PIC

TIMING TO PIC (PIC IN SLAVE MODE)				
[illegible]	Characteristic	Min	Max	Unit
[illegible]	Valid Address to AS* Low	10		ns
[illegible]	Valid Data from 7M High(S4) on Wr to PIC		35	ns

TIMING TO PIC (PIC IN MASTER MODE)				
[illegible]	Characteristic	Min	Max	Unit
[illegible]	Valid Data setup to Local 7M low(S7)	15		ns

2000 SYSTEM BUS LOADING

The following numbers and notations are used for standard load and drive values:

Type		From A2000 (IC input load)		To A2000 (IC output drive)
F-Driver TTL	FD	20μA @ 2.7V −1.6mA @ 0.5V	fd	2.0V @ −15mA 0.5V @ 64mA
F-Series TTL	F	20μA @ 2.7V −0.6mA @ 0.5V	f	2.7V @ −1mA 0.5V @ 20mA
LS-Driver TTL	LSD	20μA @ 2.7V −0.4mA @ 0.4V	lsd	2.0V @ −15mA 0.5V @ 24mA
LS-Series TTL	LS	20μA @ 2.7V −0.4mA @ 0.4V	ls	2.7V @ −400μA 0.5V @ 8mA
MOS	MOS	10μA @ 2.4V −10μA @ 0.4V	mos	2.4V @ −200μA 0.4V @ 3.2mA
Open Collector			oc	FROM RESISTOR 0.5V @ 8mA

Any lesser input load can be used on a signal in place of a greater load or equivalent load. Varying the number of load elements while still meeting the DC loading criteria can be done if necessary, but it is not a good idea, as it can still exceed the expected capacitive loading on the signal.

A final type of drive is the open collector (oc). Some PIC outputs must be open collector, as they are in a wired-or configuration with the same output from other PICs or motherboard signals.

Most of the system bus signals provide a standard drive to their respective connectors. If your drivers can meet the input specification, don't worry about what is actually required. However, even if your loading doesn't exceed the specified drive capacity of slot signal mentioned above, consult the following chart for specific signals that may provide less drive than a standard signal of that type. Signals that match the STANDARD loading are not separately listed.

Named Signals	DIR	Expansion Slots (each)	Coprocessor Slot	Video Slot
STANDARD	I	2F	1F	1F
STANDARD	O	10f	10f	10f
/DTACK	I	1F	1F	
	O	10f	10f	
/OVR	O	oc	oc	
XRDY	O	oc	oc	
/INT2	O	oc	oc	
/INT6	O	oc	oc	
/EINT1	O	oc		
/EINT4	O	oc		
/EINT5	O	oc		

amed gnals	DIR	Expansion Slots (each)	Coprocessor Slot	Video Slot
:NT7	O	oc		
LAVEn	O	2f		
FGOUTn	O	2f		
COPCFG	O		2f	
Clock	I	1F	1F	
MHz Clock	I	1F	1F	
SERR	I	1F	1F	
	O	oc	oc	
PA	I	1F	1F	
	O	oc	oc	
VA	I	1F	1F	
	O	10f	10f	
RST	I	1F	1F	
	O	oc	oc	
FLT	I	1F	1F	
	O	oc	oc	
CAN	O	oc		
SR	O	2f		
CBR	I		2F	
	O		2f	
CBC	I		2F	
	O		2f	
SCACK	I	1F	1F	
	O	oc	oc	
SCSS	O		2f	
CLK	O			2f
CLKEN	O			2f

TABLE 3-2

PAL16L8
STEERING150R17 REV3
11-17-85
AMIGA

```
/SLVOUT RD /ASQ /ASQ90 COLLIS /BG /AS /BGACK /DMAIN GND
/OWN /AOE /UDS /BERR /DMAOUT /LDS /DBOE /RES /D2P VCC

DBOE    = AS  * /RD        * /BERR +          ;DATA DRIVERS DURING WRITE CYCLE
          UDS *  RD * ASQ * /BERR +           ;TURN ON DRIVERS LATE FOR RD
          LDS *  RD * ASQ * /BERR             ;UDS AND LDS PROTECT RD MOD WR
                                              ;TO AVOID TRI__STATE FIGHT
D2P     = /DMAOUT *  SLVOUT *  RD +           ;DOWNSTREAM READS UPSTREAM SLAVE
           DMAOUT * /SLVOUT * /RD +           ;UPSTREAM WRITES DOWNSTREAM SLAVE
           DMAOUT *  SLVOUT                   ;MASTER AND SLAVE ARE UPSTREAM

AOE     = BGACK            +
          /BG * /DMAOUT +
          AS               +                  ;AS KEEPS ADDR WHEN /BG DROPS
          ASQ90                               ;ASQ90 MAINTAINS VALID ADDR ON
                                              ; LAST PROC CYCLE

DMAOUT  = DMAIN + OWN

IF (/RES * COLLIS) BERR = VCC
```

DESCRIPTION

SLVOUT = SLAVEOUT,ASQ = AS DELAYED,ASQ90 = AS CLKD ON LOW EDGE OF 7M,
BG = BUS GRANT,OWN = LOCAL OWN
COLLIS = BUS COLLISION,AOE = ADDR OUTPUT EN,DOE = DATA OE
RES = RESET,D2P = DATA TO PROCESSOR
UDS LDS PROTECT AGAINST RDMODIFYWRITE 3STFIGHT & BERR = /DOE

ABLE 3-3

AL16R6
RBITRATE REV1
6-86
MIGA

```
' BRIN /RES /BGIN /BR5 /BR4 /BR3 /BR2 /BR1 GND
GND /BGOUT /BGOLD /BG5 /BG4 /BG3 /BG2 /BG1 /BR VCC

C1  = BGIN * /BGOLD * BR1 *                                  /RES + ;GENERATE BG1
      BGIN * BG1     *                                       /RES   ;HOLD UNTIL /BG

C2  = BGIN * /BGOLD * BR2 * /BR1 *                           /RES +
      BGIN * BG2     *                                       /RES

C3  = BGIN * /BGOLD * BR3 * /BR1 * /BR2 *                    /RES +
      BGIN * BG3     *                                       /RES

C4  = BGIN * /BGOLD * BR4 * /BR1 * /BR2 * /BR3 *                  /RES +
      BGIN * BG4     *                                            /RES

C5  = BGIN * /BGOLD * BR5 * /BR1 * /BR2 * /BR3 * /BR4 *           /RES +
      BGIN * BG5     *                                            /RES

GOLD = BGIN                                                  ;STORE OLD STATE OF BG

R    = BRIN * /RES +                                         ;BR IS RQST TO 68K
       BR1  * /RES +
       BR2  * /RES +
       BR3  * /RES +
       BR4  * /RES +
       BR5  * /RES

GOUT = BGIN * BGOLD * /BG1 * /BG2 * /BG3 * /BG4 * /BG5
```

ESCRIPTION

C1 IS HIGHEST PRIORITY

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

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

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

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

F  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

TABLE 3-5

PAL16L8
COLLISION
11-17-85
AMIGA

```
BAS /SLV1 /SLV2 /SLV3 /SLV4 /SLV5 /SLVIN A23 A22 GND
A21 /SLVOUT A20 A19 /OVR /RESET P17 /PROC /NOTCOLIS VCC

SLVOUT = SLV1 + SLV2 + SLV3 + SLV4 + SLV5 + SLVIN

NOTCOLIS =          /SLV1 * /SLV2 * /SLV3 * /SLV4 * /SLV5 * /SLVIN +
           /PROC *          /SLV2 * /SLV3 * /SLV4 * /SLV5 * /SLVIN +
           /PROC * /SLV1 *          /SLV3 * /SLV4 * /SLV5 * /SLVIN +
           /PROC * /SLV1 * /SLV2          * /SLV4 * /SLV5 * /SLVIN +
           /PROC * /SLV1 * /SLV2 * /SLV3          * /SLV5 * /SLVIN +
           /PROC * /SLV1 * /SLV2 * /SLV3 * /SLV4          * /SLVIN +
           /PROC * /SLV1 * /SLV2 * /SLV3 * /SLV4 * /SLV5

PROC    = BAS * /A23 * /A22 * /A21 *                  /RESET * /OVR +
          BAS *  A23 * /A22 *  A21 *                  /RESET * /OVR +
          BAS *  A23 *  A22 * /A21 *                  /RESET * /OVR +
          BAS *  A23 *  A22 *  A21 * /A20 * /A19 * /RESET * /OVR +
          BAS *  A23 *  A22 *  A21 *  A20 *  A19 * /RESET * /OVR
```

DESCRIPTION

EMPTY

INTERFACING TO THE 68K BUS CONNECTOR ON THE AMIGA 500

This section gives the necessary information for interfacing to the 68000 bus connector on the left side of the Amiga A500 (or the right side of the A1000).

THE CONNECTOR ON THE AMIGA

The connector is a standard dual row 86 finger (43 on a side) edge connector, spaced on .1" centers. Here are some part numbers of connectors that are compatible:

solder tail AMP 2-530841-1
wire wrap AMP 4-530396-7
card extender AMP 1-530826-2

See accompanying drawing for physical dimensions of this connector on the A500, Figure A-3 in Appendix A.

TIMING

For this discussion, see Figure 3.2.

Clocks

The entire computer board is run synchronously to the 3.57954Mhz color clock (C1). This is accomplished by generating a number of sub-multiple frequencies from our master 28.63636Mhz crystal oscillator. The following are the primary clocks on the board:

Name	Description
C1	The 3.579545Mhz Color Clock
C2	C1 shifted 45 degrees later
C3	C1 shifted 90 degrees later
C4	C1 shifted 135 degrees later
7M	C1 XORed with C3* (7.15909Mhz)
DAC	7M shifted 90 degrees later

7M is the processor clock for the 68000 microprocessor. C1-C4 and DAC are used to clock the custom chips and for determining the timing of signals to the memory arrays.

The above frequencies are true for NTSC Amigas. A PAL Amiga will operate slightly slower, with a main clock of 28.37516Mhz. This is divided down to get 7M = 7.09379Mhz and C1 = 3.546895Mhz. A special circuit is required to take five fourths of C1 to derive the PAL colorburst frequency of 4.43361875Mhz.

The following clocks are available at the edge connector:

Name	Pin	Description
C3*	14	C3 inverted
CDAC	15	DAC equivalent
C1*	16	C1 inverted

Note that 7M (the processor clock) is not available at the connector; it can be easily generated by:

C3* XNOR C1* = 7M equivalent

If you need a 14.31818Mhz synchronous clock, you can generate it by:

(7Mequiv) XOR (CDAC) = 14M equivalent

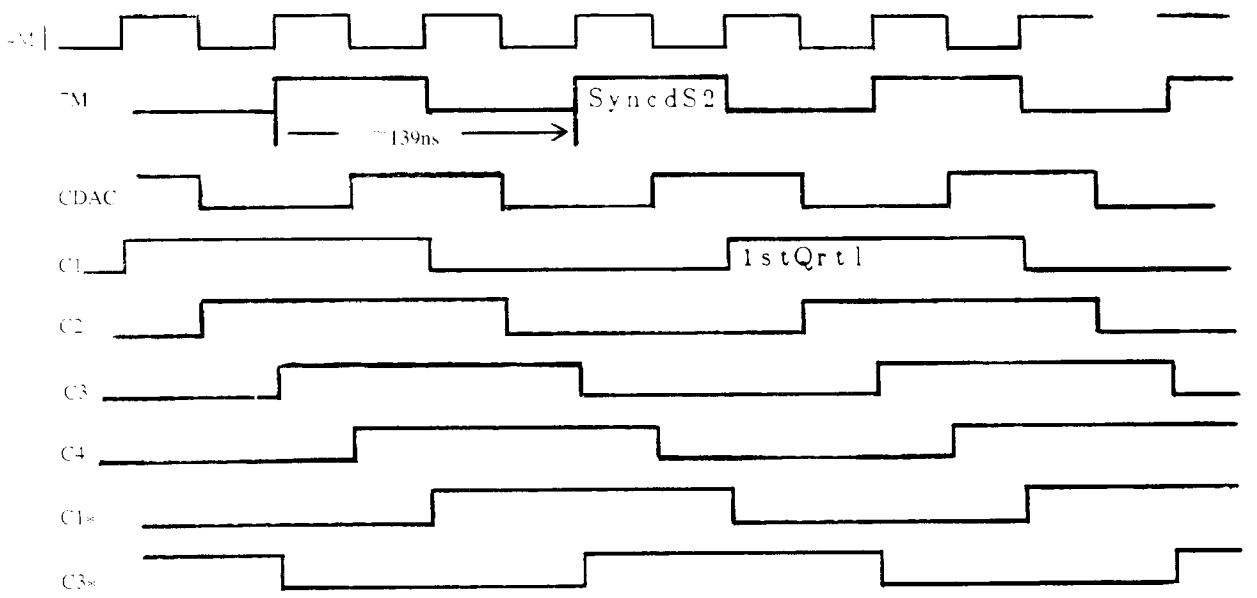


Fig. 3.2 Amiga System Clocks

Bus Timing

The 68000 is connected directly to the 86 pin connector, there are no buffers between the 68000 and the connector. Two control inputs, VPA* and DTACK* are driven by logic on the Amiga and should not be driven by your circuitry, unless OVR* is used to disable this logic.

Many boxes are being designed which pass the bus (buffered) out in daisy chain fashion.

In order to allow your device to be the second in the chain, take into account an extra level of signal buffers on:

AS*, UDS*, LDS*, Address, Data, Clocks

Furthermore, if you are designing a DMA device, the Amiga provides data in response to a Read very late (50ns prior to the fall of S6). If your DMA device is looking at this data through two or three 74F245's (7ns each), this data will not be valid at your DMA controller until approximately 25ns prior to the fall of S6.

CPU bus timing is based on an 8Mhz 68000, with only one exception: under normal operation, the bus control PAL asserts DTACK* for you. DO NOT ASSERT DTACK*; do not attach any outputs to the DTACK* line.

Slave Bus Timing

Details of 68000 timing are available in the Motorola 68000 hardware manual. If you are designing a bus slave, most bus timing is per the 68000 spec, except that the CPU will pull DTACK* for you. If you need to delay our assertion of DTACK*, you must pull XRDY (Pin 18) no later than 60ns after the assertion of AS*. You should release XRDY when you are ready to complete the bus cycle.

Also remember that in the expansion architecture, data drivers should not turn on during a Read cycle until S4.

For those of you who have not designed anything on the 68K bus before, this description is intended to make looking at the Motorola timing diagrams easier. For more details and timing specs see Motorola hardware manual (fold out timing diagrams in the back of the book.)

See Figure 3.2 in this section. Motorola labels the states of the processor clock S0-S7. The processor starts driving the address lines during S1, and asserts AS* (Address Strobe) during S2. If the cycle is a read, the data strobes (UDS*,LDS*) are asserted during S2 also (they are delayed until S4 on a write).

The board responds to AS* by asserting DTACK* (unless you delay DTACK by pulling XRDY low). In order to run a normal 4 clock bus cycle, DTACK* meets the setup time prior to S5. DTACK* is the acknowledge to the bus cycle. If DTACK* is not asserted, the 68000 stays in the middle of the bus cycle until DTACK* (or BERR* or VPA*) is asserted. Once DTACK* is asserted, the processor completes the read (or write) and ends the cycle by disasserting the strobes (AS*,UDS*,LDS*) and tri-stating its bus drivers.

If the slave you are designing cannot respond fast enough to successfully complete a 4 clock bus cycle, it must pull XRDY low within 60ns after the assertion of AS* (and of course the correct address). Our board then will not assert DTACK* until you release XRDY. You should drive XRDY with an open collector output; we provide a 1K pullup resistor on our board.

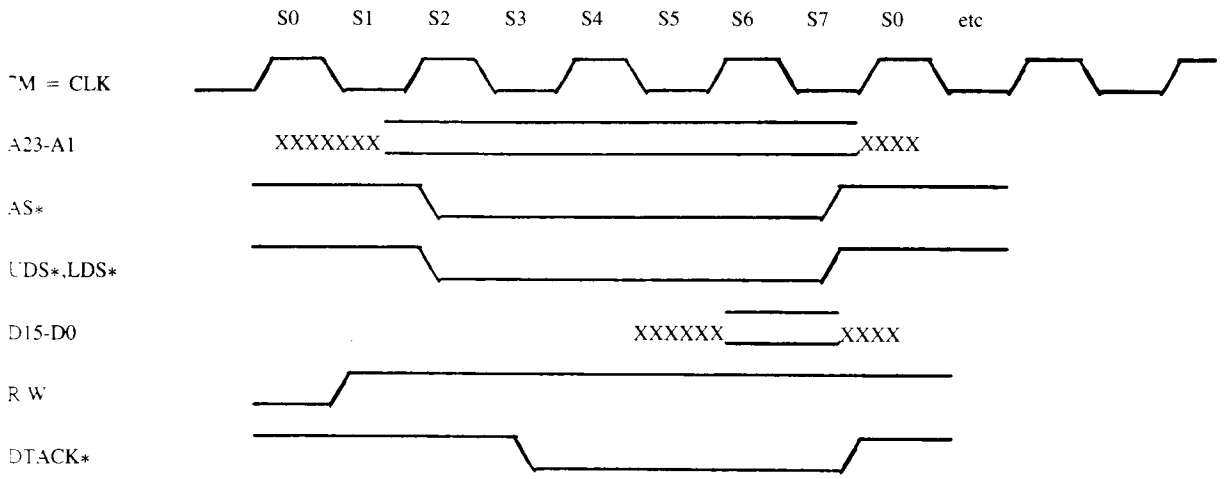


Fig. 3.3 Standard 4 Clock Read Cycle

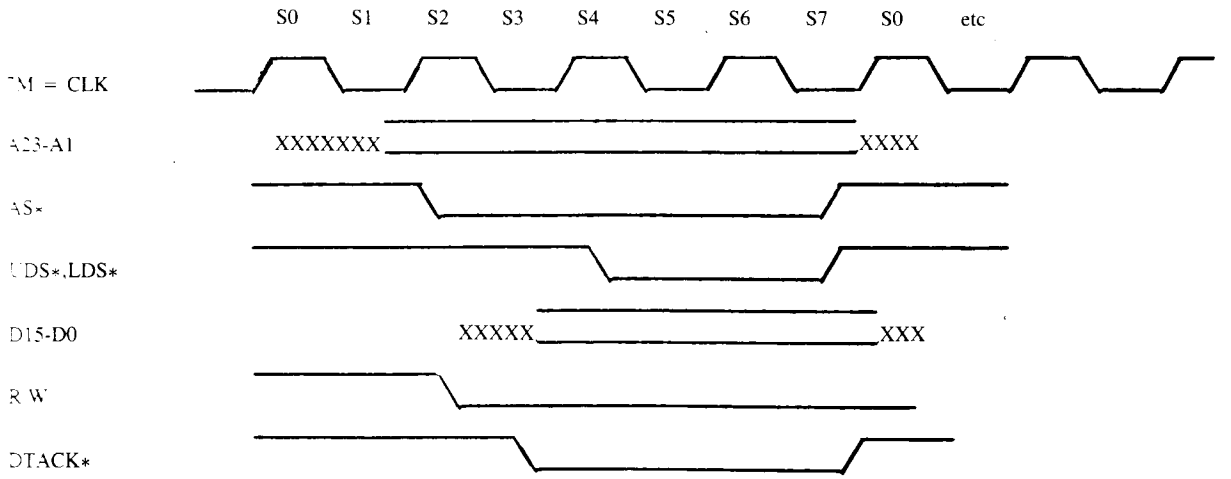


Fig. 3.4 Standard 4 Clock Write Cycle

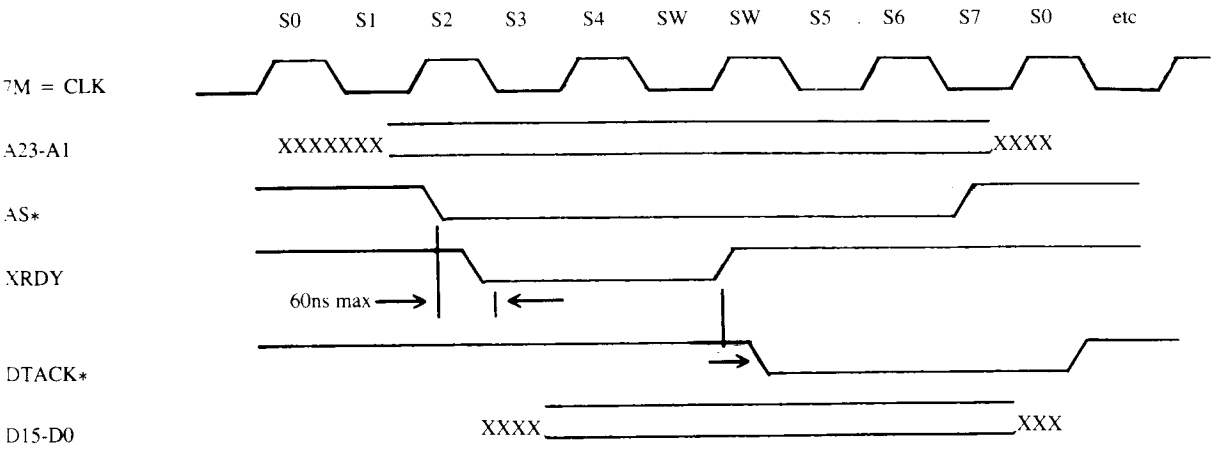


Fig. 3.5 Using XRDY to Delay DTACK*

Master Bus Timing

All bus masters must run synchronously to 7M (equivalent), as does the 68000 in the Amiga.

The necessary information for designing a bus master is in the 68000 hardware manual. A master must meet all of the bus timing specs of an 8Mhz 68000; for example, valid address must precede AS* by at least 30ns.

If you are designing a bus master card that will plug into a box, remember that the address will have to propagate through the address drivers built into the box; you should probably allow for the prop delay of three 74F245's in addition to the required 30ns.

The strobes, such as AS*,UDS*,LDS*, must all function as they would basically on the 68000 spec. A master must also respond to DTACK*, HALT*, and BERR* correctly.

BGACK* and OWN* Timing to Avoid Bus Contention

The basic timing for bus arbitration conforms very closely to the 68000 and the 68440. When the new master has received BG* and all other signals necessary to take mastership, it must assert OWN* before it asserts BGACK*. This gives the address drivers on the bus time to change direction, if necessary, before BGACK* turns them on.

At the end of the DMA cycle, BGACK* must be disasserted before OWN* is disasserted.

BR* should always be asserted off the rising edge of 7M, and should be valid no later than 60ns after that edge.

Driver documentation

This section discusses how the "binddrivers" program finds your driver and links it into the system. It also hints on how to write your code to take advantage of this.

First off, the expansion library goes out and configures the expansion boards in the system. It puts each board in its own address space, and links memory boards into the memory free pool. This is done by the expansion.library's ConfigChain entry point. This code is intended to be run early on in system startup, before any other code is around.

Later on, after the DOS is running, the binddrivers program should be run. This program searches the directory "SYS:Expansion" for workbench icon files. If it finds one with a tooltypes variable "PRODUCT" then it parses the rest of the line (see below) and looks for an unconfigured board that matches the description.

This method makes user installation of a new driver trivial: the user only has to copy a workbench icon into the expansion directory on his sys disk. Everything else is automatic the next time he boots.

In addition, the bootdrivers program may be run repeatedly without ill effect. Devices will not be configured twice, so binddrivers may be run after a new driver is installed (so the user does not have to reboot after installing a driver).

OVERVIEW

Here is an overview of the process:

search:
for each file that ends in .info, do test ( ).

test:

1. Call GetDiskObject( ) on this file. If not a workbench object, return.
2. Call FindToolType ( ) to see if there is a PRODUCT definition. If not, return.
3. If the description does not match an unconfigured board, return. If there are boards, link them all together and record them in a static area.
4. LoadSeg ( ) the code file. If LoadSeg fails, return.
5. Search the first hunk for a Resident structure. If no structure, UnLoadSeg ( ) the segment and return.

6. InitResident ( ) the loaded code. If an error (NULL) is returned, UnLoadSeg ( ) the segment.

your driver code:

Find the list of boards. Mark them a configured, and record your driver in them (for system debugging). Return non-zero value if everything went ok. If something went wrong (or you just want to be unloaded) then return NULL.

Now for some more detail.

1. GetDiskObject ( ) is a routine in icon.library. It will read in the disk object, and return a pointer to it. Part of a disk object structure is a "tooltypes" field.

2. The FindToolType ( ) routine (also in the icon.library) searches the tooltypes database associated with the disk object. If there is an entry for PRODUCT then it is assumed that this is an info file for a driver. The PRODUCT field is of the format:

   PRODUCT = <idlist>

   <idlist> ::= <id> | <idlist>BAR<id>
   <id> ::= <manufacturer> | <manufacturer>SLASH<product>
   <manufacturer> ::= <a decimal number>
   <product> ::= <a decimal number>
   BAR ::= <a vertical bar — '|'>
   SLASH ::= <a forwards slant char — '/'>

Spaces are not legal. Some examples:

```
PRODUCT = 1000/30          ; matches man 1000, product 30
PRODUCT = 1000             ; matches any man 1000 board
PRODUCT = 1000/20|1000/21  ; matches man 1000, product 20
                             or 21
```

3. Each unconfigured board in the system is searched. An unconfigured board has the CDB_CONFIGME bit set in the cd_Flags byte. Search all these unconfigured boards to find the ones that match any of the product codes. Link all these boards together using the cd_NextCD field of the ConFigDev structure. Record the head of this list, along with the product field and the name of the file that was loaded in a CurrentBinding structure. This structure may be retrieved via the GetCurrentBinding ( ) call.

4. Attempt to load in the driver. The driver may be a devices, library, task, process, or anything else that you may want. The only requirement is that it have a Resident structure in its first hunk. (By the way, this means that you can not directly use startup.obj).

This is why we refer to loading a "driver" rather than a "device" — you can write any sort of code you want to handle your device.

5. Binddriver will search the first hunk for a Resident structure. If it cannot find one, it will assume some awful mistake has been made, and will unload the segment.

6. Finally we get to running some of YOUR code. InitResident ( ) is used to start you off and running. The return value from InitResident (and therefore the return value from your init entry point) will be checked on exit. If it is zero then the segment will be unloaded. This can be useful if you only need to do a bit of initialization and then can go away, such as allocate additional expansion memory for a non-expansion architecture board.

HINTS FOR WRITING YOUR DRIVER CODE:

Your driver will be launched via InitResident ( ) as discussed above. If you use the underdocumented, but very useful RTF_AUTOINIT option you will have a library node constructed for you, and then have the code you specified enter. If you don't use RTF_AUTOINIT, then your code will be entered directly.

You should (among everything else you might be doing) open the expansion.library and ask for the current buildings (GetCurrentBinding()). In this structure will be the head of a singly linked list of ConfigDev structures. The structures are linked via the cd NextCD field. You should deal with each member of the list — they are for you!

There are two actions you must take. One is to unset the CDB CONFIGME bit in the cd_Flags. If you do not do this then the board is still available for other drivers (of course, you may actually want this . . .). If you do unset the CONFIGME bit, please also record your "node" in the cd_Driver structure. It is assumed that this is in an exec node, whose LN_NAME field points your name, and LN_TYPE field is your type of "thing" — library, resource, device, task, etc. I know that this will not always apply to you, but try it anyway. It will help the rest of us debug the system when something goes wrong.

You have now done everything you wanted to. Your init routine is about to return. If you return a zero, then your code will be unloaded. If you return non-zero, then you will stay around.

Software for Amiga Expansion

This section contains listings and information on the following expansion software commands:

expansion.library/AddDosNode
expansion.library/MakeDosNode
System/Libraries/Expansion/AddConfigDev
System/Libraries/Expansion/AllocBoardMem
System/Libraries/Expansion/AllocConfigDev
System/Libraries/Expansion/AllocExpansionMem
System/Libraries/Expansion/ConfigBoard
System/Libraries/Expansion/ConfigChain
System/Libraries/Expansion/FindConfigDev
System/Libraries/Expansion/FreeBoardMem
System/Libraries/Expansion/FreeConfigDev
System/Libraries/Expansion/FreeExpansionMem
System/Libraries/Expansion/GetCurrentBinding
System/Libraries/Expansion/ObtainConfigBinding
System/Libraries/Expansion/ReadExpansionByte
System/Libraries/Expansion/ReadExpansionRom
System/Libraries/Expansion/ReleaseConfigBinding
System/Libraries/Expansion/RemConfigDev
System/Libraries/Expansion/SetCurrentBinding
System/Libraries/Expansion/WriteExpansionByte

EXPANSION.LIBRARY/ ADDDOSNODE

NAME

AddDosNode — mount a disk to the system

SYNOPSIS

```
ok = AddDosNode( bootPri, flags, deviceNode )
D0               D0      D1     A0
```

FUNCTION

This routine makes sure that your disk device (or a device that wants to be treated as if it was a disk...) will be entered into the system. If the dos is already up and running, then it will be entered immediately. If the dos has not yet been run then the data will be recorded, and the dos will get it later.

We hope to eventually try and boot off a disk device. We will try and boot off of each device in turn, based on priority, if there is no boot floppy in the floppy disk drive. As of this writing that facility does not yet exist.

There is only one additional piece of magic done by AddDosNode. If there is no executable code specified in the deviceNode structure (e.g. dn_SegList, dn_Handler, and dn_Task are all null) then the standard dos file handler is used for your device.

Documentation note: a "task" as used here is a dos-task, not an exec-task. A dos-task, in the strictest sense, is the "address of an exec-style message port. In general, it is a pointer to a process's pr_MsgPort field (e.g. a constant number of bytes after an exec port).

INPUTS

bootPri — a BYTE quantity with the boot priority for this disk.
This priority is only for which disks should be looked at: the actual disk booted from will be the first disk with a valid boot block. If no disk is found then the "bootme" hand will come up and the bootstrap code will wait for a floppy to be inserted. Recommend priority assignments are:

+5 — unit zero for the floppy disk. The floppy should always be highest priority to allow the user to abort out of a hard disk boot.
0 — the run of the mill hard disk
–5 — a "network" disk (local disks should take priority).
–128 — don't even bother to boot from this device.

flags — additional flag bits for the call:
ADN_TARTPROC (bit 0) — start a handler process immediately.
Normally the process is started only when the device node is first referenced. This bit is meaningless if you have already specified a handler process (non-null dn_Task).

deviceNode — a legal DOS device node, properly initialized.
Typically this will be the result of a MakeDosNode() call, but feel free to manufacture your own if you need to. If deviceNode is null then AddDosNode does nothing.

RESULTS

ok - non-zero everything went ok, zero if we ran out of memory or some other weirdness happened.

EXAMPLES

```
/* enter a bootable disk into the system. Start a file handler
** process immediately.
*/
AddDosNode( 0, ADNF_STARTPROC, MakeDosNode( paramPacket )
);
```

BUGS

The flexible boot strategy is only that — strategy. It still needs to be reflected in code somewhere.

SEE ALSO

MakeDosNode

EXPANSION.LIBRARY/ MAKEDOSNODE

NAME

MakeDosNode — construct dos data structures that a disk needs

SYNOPSIS

```
deviceNode = MakeDosNode( parameterPkt )
D0                        A0
```

FUNCTION

This routine manufactures the data structures needed to enter a dos disk device into the system. This consists of a DeviceNode, a FileSysStartupMsg, a disk environment vector, and up to two bcpl strings. See the libraries/dosextens and libraries/filehandler include files for more information.

MakeDosNode will allocate all the memory it needs, and then link the various structure together. It will make sure all the structures are long-word aligned (as required by the DOS). It then returns the information to the user so he can change anything else that needs changing. Typically he will then call AddDosNode() to enter the new device into the dos tables.

INPUTS

parameterPkt - a longword array containing all the information needed to initialize the data structures. Normally I would have provided a structure for this, but the variable length of the packet caused problems. The two strings are null terminated strings, like all other exec strings.

longword	description
0	string with dos handler name
1	string with exec device name
2	unit number (for OpenDevice)
3	flags (for OpenDevice)
4	# of longwords in rest of environment
5-n	file handler environment (see libraries/file-handler.h)

RESULTS

deviceNode — pointer to initialize device node structure, or null if there was not enough memory.

EXAMPLES

```
/* set up a 3.5" amiga format floppy drive for unit 1 */

char execName[] = "trackdisk.device";
char dosName[] = "df1";

ULONG parmPkt[] = [
  (ULONG) dosName,
  (ULONG) execName,
   1,                    /* unit number */
   0,                    /* OpenDevice flags */
   /* here is the environment block */
   11,                   /* table upper bound */
   512>>2,               /* # longwords in a block */
   0,                    /* sector origin — unused */
   2,                    /* number of surfaces */
   1,                    /* secs per logical block — unused */
   11,                   /* secs per track */
   2,                    /* reserved blocks — 2 boot blocks */
   0,                    /* ?? — unused */
   0,                    /* interleave */
   0,                    /* lower cylinder */
   79,                   /* upper cylinder */
   5,                    /* number of buffers */
];
struct Device Node *node, *MakeDosNode();
node = MakeDosNode( parmPkt );
```

BUGS

SEE ALSO

AddDosNode

SYSTEM/LIBRARIES/ EXPANSION/ ADDCONFIGDEV

NAME

AddConfigDev — add a new ConfigDev structure to the system

SYNOPSIS

```
AddConfigDev( configDev )
              A0
```

FUNCTION

This routine adds the specified ConfigDev structure to the list of Configuration Devices in the system.

INPUTS

configDev — a valid ConfigDev structure.

RESULTS

EXCEPTIONS

SEE ALSO

RemConfigDev

BUGS

SYSTEM/LIBRARIES/ EXPANSION/ ALLOCBOARDMEM

NAME

AllocBoardMem — allocate standard device expansion memory

SYNOPSIS

```
startSlot = AllocBoardMem( slotSpec )
D0                         D0
```

FUNCTION

This function allocates numslots of expansion space (each slot is E_SLOTSIZE bytes). It returns the slot number of the start of the expansion memory. The EC_MEMADDR macro may be used to convert this to a memory address.

AllocBoardMem( ) knows about the intracacies of expansion board hardware and will allocate the proper expansion memory for each board type.

INPUTS

slotSpec — the memory size field of the Type byte of an expansion board

RESULTS

startSlot — the slot number that was allocated, or -1 for error.

EXAMPLES

```
struct ExpansionRom *er;
slot = AllocBoardMem( er->er_Type & ERT_MEMMASK )
```

EXCEPTIONS

SEE ALSO

AllocExpansionMem, FreeExpansionMem, FreeBoardMem

BUGS

SYSTEM/LIBRARIES/ EXPANSION/ ALLOCCONFIGDEV

NAME

AllocConfigDev — allocate a ConfigDev structure

SYNOPSIS

```
configDev = AllocConfigDev()
DO
```

FUNCTION

This routine returns the address of a ConfigDev structure. It is provided so new fields can be added to the structure without breaking old, existing code. The structure is cleared when it is returned to the user.

INPUTS

RESULTS

configDev — either a valid ConfigDev structure or NULL.

EXCEPTIONS

SEE ALSO

FreeConfigDev

BUGS

SYSTEM/LIBRARIES/ EXPANSION/ ALLOCEXPANSIONMEM

NAME

AllocExpansionMem — allocate expansion memory

SYNOPSIS

```
startSlot = AllocExpansionMem( numSlots, slotOffset )
D0                             D0        D1
```

FUNCTION

This function allocates numslots of expansion space (each slot is E_SLOTSIZE bytes). It returns the slot number of the start of the expansion memory. The EC_EMADDR macro may be used to convert this to a memory address.

Boards that fit the expansion architecture have alignment rules. Normally a board must be on a binary boundary of its size. Four and Eight megabyte boards have special rules. User defined boards might have other special rules.

The routine AllocBoardMem( ) knows about all the allocation rules for standard boards. Most users will want to use that routine if they want memory for a standard expansion device.

If AllocExpansionMem( ) succeeds, the startSlot will satisfy the following equation:

(startSlot — slotOffset) MOD slotAlign = 0

INPUTS

numSlots — the number of slots required.
slotOffset — an offset from that boundary for startSlot.

RESULTS

startSlot — the slot number that was allocated, or -1 for error.

EXAMPLES

```
AllocExpansionMem( 2, 0 )
```

Tries to allocate 2 slots on a two slot boundary.

```
AllocExpansionMem( 64, 32 )
```

This is the allocation rule for 4 meg boards. It allocates 4 megabytes (64 slots) on an odd 2 meg boundary.

EXCEPTIONS

SEE ALSO

FreeExpansionMem, AllocBoardMem, FreeBoardMem

BUGS

NAME

SYSTEM/LIBRARIES/ EXPANSION/ CONFIGBOARD

ConfigBoard — configure a board

SYNOPSIS

```
error = ConfigBoard( board, configDev )
D0A0                 A1
```

FUNCTION

This routine configures an expansion board. The board will generally live at E_EXPANSIONBASE, but the base is passed as a parameter to allow future compatibility. The configDev parameter must be a valid configDev that has already had ReadExpansionRom() called on it.

ConfigBoard will allocate expansion memory and place the board at its new address. It will update configDev accordingly. If there is not enough expansion memory for this board then an error will be returned.

INPUTS

board — the current address that the expansion board is responding.
configDev — an initialized ConfigDev structure.

RESULTS

error — non-zero if there was a problem configuring this board

EXCEPTIONS

SEE ALSO

FreeConfigDev

BUGS

SYSTEM/LIBRARIES/ EXPANSION/ CONFIGCHAIN

NAME

ConfigChain — configure the whole damn system

SYNOPSIS

```
error = ConfigChain( baseAddr )
D0                   A0
```

FUNCTION

This is the big one! This routine will take a base address (generally E_EXPANSIONBASE) and configure all the devices that live there. This routine will call all the other routines that might need to be called. All boards that are found will be linked into the configuration list.

INPUTS

baseAddr — the base address to start looking for boards.

RESULTS

error — non-zero if something went wrong.

EXCEPTIONS

SEE ALSO

FreeConfigDev

BUGS

NAME

FindConfigDev — find a matching ConfigDev entry

SYNOPSIS

```
configDev = FindConfigDev( oldConfigDev, manufacturer, product )
D0                         A0            D0            D1
```

FUNCTION

This routine searches the list of existing ConfigDev structures in the system and looks for one that has the specified manufacturer and product codes.

If the oldConfigDev is NULL the the search is from the start of the list of configuration devices. If it is not null then it searches from the first configuration device entry AFTER oldConfigDev.

A code of -1 is treated as a wildcard — e.g. it matches any manufacturer (or product)

INPUTS

oldConfigDev — a valid ConfigDev structure, or NULL to start from the start of the list.
manufacturer — the manufacturer code being searched for, or -1 to ignore manufacturer numbers.
product — the product code being searched for, or -1 to ignore product numbers.

RESULTS

configDev — the next ConfigDev entry that matches the manufacturer and product codes, or NULL if there are no more matches.

EXCEPTIONS

EXAMPLES

```
/* to find all configdevs of the proper type */
struct ConfigDev *cd = NULL;
while( cd = FindConfigDev( cd, MANUFACTURER, PRODUCT ) ) [
   /* do something with the returned ConfigDev */
]
```

SEE ALSO

BUGS

SYSTEM/LIBRARIES/ EXPANSION/ FREEBOARDMEM

NAME

FreeBoardMem — allocate standard device expansion memory

SYNOPSIS

```
FreeBoardMem( startSlot, slotSpec )
D0            D1
```

FUNCTION

This function frees numslots of expansion space (each slot is E_SLOTSIZE bytes). It is the inverse function of AllocBoardMem().

INPUTS

startSlot — a slot number in expansion space.
slotSpec — the memory size field of the Type byte of an expansion board

RESULTS

EXAMPLES

```
struct ExpansionRom *er;
int startSlot;
int slotSpec;

slotSpec = er->er_Type & ERT_MEMMASK;
startSlot = AllocBoardMem( er->er_Type & ERT_MEMMAK );

if( startSlot != -1 ) [
FreeBoardMem( startSlot, slotSpec );
]
```

EXCEPTIONS

If the caller tries to free a slot that is already in the free list, FreeBoardMem will Alert() (e.g. crash the system).

SEE ALSO

AllocExpansionMem, FreeExpansionMem, AllocBoardMem

BUGS

SYSTEM/LIBRARIES/ EXPANSION/ FREECONFIGDEV

NAME

FreeConfigDev — allocate a ConfigDev structure

SYNOPSIS

```
FreeConfigDev( configDev )
                  A0
```

FUNCTION

This routine frees a ConfigDev structure as returned by AllocConfigDev.

INPUTS

configDev — a valid ConfigDev structure.

RESULTS

EXCEPTIONS

SEE ALSO

AllocConfigDev

BUGS

SYSTEM/LIBRARIES/ EXPANSION/ FREEEXPANSIONMEM

NAME

FreeExpansionMem — allocate standard device expansion memory

SYNOPSIS

```
FreeExpansionMem(startSlot, numSlots )
                        D0        D1
```

FUNCTION

This function allocates numslots of expansion space (each slot is E_SLOTSIZE bytes). It is the inverse function of AllocExpansionMem( ).

INPUTS

startSlot — the slot number that was allocated, or -1 for error.
numSlots — the number of slots to be freed.

RESULTS

EXAMPLES

EXCEPTIONS

If the caller tries to free a slot that is already in the free list, FreeExpansionMem will Alert( ) (e.g. crash the system).

SEE ALSO

AllocExpansionMem, AllocBoardMem, FreeBoardMem

BUGS

SYSTEM/LIBRARIES/ EXPANSION/ GETCURRENTBINDING

NAME

GetCurrentBinding — sets static board configuration area

SYNOPSIS

```
actual = GetCurrentBinding( currentBinding, size )
                                  A0 D0:16
```

FUNCTION

This function writes the contents of the "currentBinding" structure out of a private place. It may be set via SetCurrentBinding(). This is really a kludge, but it is the only way to pass extra arguments to a newly configured device.

A CurrentBinding structure has the name of the currently loaded file, the product string that was associated with this driver, and a pointer to the head of a singly linked list of ConfigDev structures (linked through the cd_NextCD field).

Many devices may not need this information; they have hard coded into themselves their manufacture number. It is recommended that you at least check that you can deal with the product code in the linked ConfigDev structures.

INPUTS

currentBinding — a pointer to a CurrentBinding structure

size — the size of the user's binddriver structure. No more than this much data will be copied. If size is larger than the libraries idea a CurrentBinding size, then the structure will be null padded.

RESULTS

actual — the true size of a CurrentBinding structure is returned.

EXAMPLES

EXCEPTIONS

SEE ALSO

GetCurrentBinding

BUGS

SYSTEM/LIBRARIES/ EXPANSION/ OBTAINCONFIGBINDING

NAME

ObtainConfigBinding — try to get permission to bind drivers

SYNOPSIS

ObtainConfigBinding( )

FUNCTION

ObtainConfigBinding gives permission to bind drivers to ConfigDev structures. It exists so two drivers at once do not try and own the same ConfigDev structure. This call will block until it is safe to proceed.

Individual drivers do not need to call this routine. It is intended for BindDriver program, and others like it. If your drivers won't be loaded via the standard method, you may need to lock out others.

It is crucially important that people lock out others before loading new drivers. Much of the data that is used to configure things is statically kept, and others need to be kept from using it.

This call is build directly on Exec SignalSemaphore code (e.g. ObtainSemaphore).

INPUTS

RESULTS

EXCEPTIONS

SEE ALSO

ReleaseConfigBinding

BUGS

SYSTEM/LIBRARIES/ EXPANSION/ READEXPANSIONBYTE

NAME

ReadExpansionByte — read a byte nybble by nybble.

SYNOPSIS

```
byte = ReadExpansionByte( board, offset )
D0                        A0     D0
```

FUNCTION

ReadExpansionByte reads a byte from a new-style expansion board. These boards have their readable data organized as a series of nybbles in memory. This routine reads two nybbles and returns the byte value.

In general, this routine will only be called by ReadExpansionRom.

The offset is a byte offset into a ExpansionRom structure. The actual memory address read will be four times larger. The macros EROFFSET and ECOFFSET are provided to help get these offsets from C.

INPUTS

board — a pointer to the base of a new style expansion board. offset — a logical offset from the board base

RESULTS

byte — a byte of data from the expansion board, or -1 if there was an error reading from the board.

EXAMPLES

```
byte = ReadExpansionByte( cd->BoardAddr, EROFFSET( er_Type )
); ints = ReadExpansionByte( cd->BoardAddr, ECOFFSET
( ec_Interrupt ) );
```

EXCEPTIONS

SEE ALSO

WriteExpansionByte, ReadExpansionRom

BUGS

NAME

ReadExpansionRom — read a board's configuration ROM space

SYNOPSIS

```
error = ReadExpansionRom( board, configDev )
D0                        A0     A1
```

FUNCTION

ReadExpansionRom reads a the ROM portion of an expansion device in to cd_Rom portion of a ConfigDev structure. This routine knows how to detect whether or not there is actually a board there.

In addition, the Rom portion of a new style expansion board is encoded in ones-complement format (except for the first two nybbles — the er_Type field). ReadExpansionRom knows about this and uncomplements the appropriate fields.

INPUTS

board — a pointer to the base of a new style expansion board.
configDev — the ConfigDev structure that will be read in.
offset — a logical offset from the configdev base

RESULTS

error — If the board address does not contain a valid new style expansion board, then error will be non-zero.

EXAMPLES

```
configDev = AllocConfigDev();
if( ! configDev ) panic();

error = ReadExpansionBoard( board, configDev );
if( ! error ) [
        configDev->cd_BoardAddr = board;
        ConfigBoard( configDev );
]
```

EXCEPTIONS

SEE ALSO

ReadExpansionByte, WriteExpansionByte

BUGS

SYSTEM/LIBRARIES/ EXPANSION/RELEASE CONFIGBINDING

NAME

ReleaseConfigBinding — allow others to bind to drivers

SYNOPSIS

ReleaseConfigBinding( )

FUNCTION

This call should be used when you are done binding drivers to ConfigDev entries. It releases the SignalSemaphore; this allows others to bind their drivers to ConfigDev structures.

INPUTS

RESULTS

EXAMPLES

EXCEPTIONS

SEE ALSO

ObtainConfigBinding

BUGS

SYSTEM/LIBRARIES/ EXPANSION/ REMCONFIGDEV

NAME

RemConfigDev — remove a ConfigDev structure from the system

SYNOPSIS

```
RemConfigDev( configDev )
              A0
```

FUNCTION

This routine removes the specified ConfigDev structure from the list of Configuration Devices in the system.

INPUTS

configDev — a valid ConfigDev structure.

RESULTS

EXCEPTIONS

SEE ALSO

AddConfigDev

BUGS

SYSTEM/LIBRARIES/ EXPANSION/ SETCURRENTBINDING

NAME

SetCurrentBinding — sets static board configuration area

SYNOPSIS

SetCurrentBinding(currentBinding, size)
A0 D0:16

FUNCTION

This function records the contents of the "currentBinding" structure in a private place. It may be read via GetCurrentBinding( ). This is really a kludge, but it is the only way to pass extra arguments to a newly configured device.

A CurrentBinding structure has the name of the currently loaded file, the product string that was associated with this driver, and a pointer to the head of a singly linked list of ConfigDev structures (linked through the cd_NextCD field).

Many devices may not need this information; they have hard coded into themselves their manufacture number. It is recommended that you at least check that you can deal with the product code in the linked ConfigDev structures.

INPUTS

currentBinding — a pointer to a CurrentBinding structure

size — the size of the user's binddriver structure. No more than this much data will be copied. If size is larger than the library's ideal CurrentBinding size, then the structure will be null padded.

RESULTS

EXAMPLES

EXCEPTIONS

SEE ALSO

GetCurrentBinding

BUGS

SYSTEM/LIBRARIES/ EXPANSION/ WRITEEXPANSIONBYTE

NAME

WriteExpansionByte — write a byte nybble by nybble.

SYNOPSIS

```
error = WriteExpansionByte( board, offset, byte )
D0                          A0     D0      D1
```

FUNCTION

WriteExpansionByte write a byte to a new-style expansion board. These boards have their writeable data organized as a series of nybbles in memory. This routine writes two nybbles in a very carefull manner to work with all types of new expansion boards.

To make certain types of board less expensive, an expansion board's write registers may be organized as either a byte-wide or nybble-wide register. If it is nybble-wide then it must latch the less significant nybble until the more significant nybble is written. This allows the following algorithm to work with either type of board:

write the low order nybble to bits D15-D12 of byte (offset*4) + 2

write the entire byte to bits D15-D8 of byte (offset*4)

The offset is a byte offset into a ExpansionRom structure. The actual memory address read will be four times larger. The macros EROFFSET and ECOFFSET are provided to help get these offsets from C.

INPUTS

board — a pointer to the base of a new style expansion board.
offset — a logical offset from the configdev base
byte — the byte of data to be written to the expansion board.

RESULTS

error — the routine will return a zero on success, non-zero if there was a problem.

EXAMPLES

```
err = WriteExpansionByte( cd->BoardAddr, ECOFFSET
( ec_Shutup ), 0 );
err = WriteExpansionByte( cd->BoardAddr, ECOFFSET
( ec_Interrupt ), 1 );
```

EXCEPTIONS

SEE ALSO

ReadExpansionByte, ReadExpansionRom

BUGS

100 Pin Expansion Signals on Amiga Computers

INTRODUCTION

This section details the signals found on the 100 pin standard Amiga expansion connector. The main point of this document is to discuss the signals found on the B2000 computer and how these differ from the similar signals found on A2000 computers and those of the original Zorro specification and A1000 computers. Anytime something is specified for the A2000, it is also true for the B2000 unless otherwise stated.

Changes from Previous Documents

We've attempted to keep the Expansion Bus pin specification as much the same as possible from machine to machine. However, especially concerning the changes from the original specification to the A2000 specifications, there were indeed some major changes made. Although these changes will affect relatively few boards, they're non-trivial for the boards that they do affect. In this case, we basically chose to sacrifice a small fraction of our compatibility for a reasonably large increase in the power of the Expansion Bus. If possible, add-on boards should be designed for the Expansion Bus. While the 86 pin slot is similar to the A1000 86 pin edge connector, it is intended for add-on processors, such as 68020 boards. Hard disk, memory, peripheral boards, etc. should work just fine in 100 pin expansion slots; the differences should only affect some coprocessor/turbo boards. Also note that the autoconfiguration should be done in the 100 pin slots.

Most of the Expansion Bus signals are buffered (the ZORRO detail will of course depend on the design; the characteristics assumed here will be present if the Commodore-Amiga design specifications are followed). This is an important point to keep in mind, for buffered signals should be specifically considered in any timing analysis, while unbuffered signals should be considered specifically in any loading analysis. Buffered signals are typically either inputs or some synchronous bidirectionals; outputs and asynchronous bidirectionals can't easily be buffered.

Definition of Terms

Several terms are used in the following text, and an understanding of them is required to speak proper Amiga-ese. A *PIC*, or Plug In Card, is a device that plugs into an expansion slot and follows the auto-configuration protocol. Nothing should plug into a 100 pin slot that doesn't follow this protocol. The term *slot* refers to a physical plug-in location, either the Coprocessor Slot or one of the five available Expansion Slots. The terms *100 Pin Slot* and *Expansion Slot* are considered synonyms, and describe one of the five 100 pin Expansion Slots. The *Expansion Bus* is the processor bus that is in common be-

tween all Expansion Slots. The terms *86 Pin Slot, Coprocessor Slot,* and *Local Slot* are considered synonyms, and pertain to the 86 pin Coprocessor Slot in the A2000 and B2000. The terms *86 Pin Edge* and *Expansion Edge* are considered synonyms, and pertain to the 86 pin Expansion Edge in the A1000 and A500. The *Local Bus* is the processor bus directly connected to the 68000 processor and the Coprocessor Slot or Expansion Edge; both the Coprocessor Slot and Expansion Edge are considered Local Bus Ports. Each different implementation of a hardware design is termed an *Instance* of that design; thus, the A2000's Expansion Bus, the B2000's Expansion Bus, and all third party ZORRO backplanes for the A1000 or A500 are instances of the Expansion Bus.

Along with an understanding of Amiga bus terms, a familiarity with Motorola's 68000 processor and its characteristic names and related terms will also be very useful in understanding this section.

POWER CONNECTIONS

The Expansion Bus provides several different voltages designed to supply expansion devices. The A2000 power supply is a "switching" power supply, currently rated at 200 watts, which supplies the main board and all other expansion ports, as well as the Expansion Bus.

Digital Ground (Ground)

Digital supply ground used by all expansion cards as the return path for all expansion supplies. This is found on all instances of the Expansion Bus. See the Table at the end of this section for pin assignments.

Main Supply (+5V)

Main power supply for all expansion cards, and is capable of sourcing large currents; each Expansion Slot can draw up to 2.0 Amps of +5, and a single Slot can draw as much as 4 Amps if necessary, for devices such as 8 megabyte RAM cards. The maximum supply current for the entire A2000 system is 20 Amps on the +5 supply. All ports open to the outside of the box have their own, separate +5V supply that's short protected, thus no loads external to the A2000 box need be considered. This supply is found on all instances of the Expansion Bus, though the available currents may vary. Pins: 5, 6.

Negative Supply (−5V)

Negative version of the main supply, for small current loads only; there's a total of 0.3 Amp for the entire A2000 system. Found on all instances of the Expansion Bus, though the available currents may vary. Pin: 8.

High Voltage Supply (+12V)

Higher voltage supply, useful for communications cards and other devices requiring greater than digital voltage levels. This is intended for small loading only; there's a total of 8 Amps for the entire A2000 system, much of which is normally devoted to floppy and hard disk drive motors. Found on all instances of the Expansion Bus, though the available currents may vary. Pin: 10.

Negative High Supply (−12V)

Negative version of the high voltage supply, also commonly used in communications applications, and similarly intended for small loads only; there is a total of 0.3 Amp for the entire A2000 system. This pin is an extension of the original Zorro specification, and is found in all A2000 machines. Pin: 20.

CLOCK SIGNALS

The Expansion Bus provides clock signals for expansion boards. They are generally used to allow clocked logic to be used in designs instead of delay lines. See p. 39 for bus loading specs.

/C1 Clock

This is a 3.58 MHz clock synched to the falling edge of the 7.16 MHz system clock. Also known as /CCK in some places. Pin 16.

/C3 Clock

This is a 3.58 MHz clock synched to the rising edge of the 7.16 MHz system clock. Also known as /CCKQ in some places. Pin 14.

CDAC Clock

This is a 7.16 MHz clock that leads the 7.16 MHz system clock by 70ns (90 degrees). Pin 15.

E Clock

This is the 68000 generated "E" clock, used for 6800 family peripherals driven by "E" and 6502 peripherals driven by PHI2. This clock is six 7.16 MHz clocks high, four clocks low, as per the 68000 spec. Pin 50.

7MHZ Clock

This is the 7.16 MHz system clock. On A2000/B2000 design has true 7MHz which is actually in common with the 68000's 7MHz input. On the original ZORRO bus specification this was the EQU7MHz signal, a 7M equivalent made using the relationship EQU7MHz = /C1 XNOR /C3. Because of this, there may be some timing differences in this signal among different vendors of ZORRO expansion boards and between these ZORRO boards and the A2000/B2000 system. It is possible to create an EQU7MHz clock on a ZORRO board that is nearly identical to the internal version, as on an A2000 the signal is created using exactly this aforementioned relationship. Pin 92.

DDRESSING AND ONTROL SIGNALS

These signals are various items used for the addressing of devices on the bus by the 68000 and any DMA devices. Most of these signals are buffered versions of similar 68000 signals, and are bidirectionally buffered to allow any DMA device on the bus to drive the 68000 local bus when such a device is a bus master.

ead Enable (READ)

Read enable for the bus, which is a buffered version of the 68000's R/W output. Read asserted indicates a read or internal cycle, read negated indicates a write cycle. Pin 68.

ddress Bus (A1-A23)

This is a buffered version of the 68000's address bus, providing 16 megabytes of address space, though only 8 megabytes of this address space is available to expansion bus devices. Expansion boards should only respond to address ranges assigned them during configuration; otherwise, addressing conflicts between multiple boards will arise. See Appendix for pin list.

ddress Strobe (/AS)

The falling edge of this strobe indicates that addresses are valid, the rising edge signals the end of an Expansion Bus memory cycle. This is a buffered version of the 68000 /AS signal. Found on pin 74.

ata Bus (D0-D15)

This is a buffered version of the 68000's data bus, providing 16 bits of data accessible by word or either byte. Note that the data bus is enabled by /AS asserted, so the data bus is not expected to have any significant hold time beyond /AS negated, so during write cycles in most design applications /AS should not be used to latch data. During read cycles, the enabling of the data bus is delayed to give the collision detection circuitry time to detect any collisions before data is enabled, thus avoiding any fights among the data drivers of multiple PICs. See Appendix for pin list.

Data Strobes (/LDS, /UDS)

These are buffered versions of the 68000's upper and lower data strobes. The strobes fall on data valid during transfer; the lower strobe being used for the lower byte (even byte address), the upper strobe being used for the upper byte (odd byte address). These are considered by the data bus buffers during read cycles, in case the cycle actually turns out to be a read-modify-write cycle. They're ignored during write cycles, since they can become valid quite late in the cycle, and a late enable would require unnecessarily fast data handling in certain PIC applications. Pins: 70, 72.

Valid Memory Address (/VMA)

Unbuffered output from the 68000 indicating a valid address for 6800 style peripheral devices, in response to a /VPA input. Pin 51.

Valid Peripheral Address (/VPA)

Unbuffered input to the 68000 indicating the address has selected a 6800 or 6502 style peripheral, so the 6800 style peripheral access should take place. Pin 48.

Data Transfer Acknowledge (/DTACK)

This signal is logically associated with the 68000's Data Transfer Acknowledge input. Normally in the Amiga system, Amiga system logic creates /DTACK for a simple, no-wait state cycle (this may be varied by the custom chips). Therefore, this signal is treated as an output to the Expansion and Coprocessor Slots, for most situations. Any slow device on the bus that needs to control /DTACK may do so by negating XRDY to hold off /DTACK or asserting /OVR very quickly to tri-state /DTACK. Note that depending upon when /AS is asserted by a bus master when accessing the CHIP memory, one of two possible cycles may result. If /AS is asserted during C1 low, C3 low, the bus cycle is considered "in-sync," and will proceed, with /DTACK driven as for a normal, 4 tick clock cycle. If, instead, /AS is asserted during C1 high, C3 high, the bus cycle is considered "out of sync" and the internally generated /DTACK will be held off, causing a wait state that's designed to "sync-up" the DMA cycle with the custom chip's memory cycle. This signal is on pin 66.

Processor Status (FC0-FC2)

These signals are the buffered versions of the 68000 Processor Status outputs, which can be used by bus devices to determine the internal state of the 68000 any time /AS is asserted. Pins 31, 33, 35.

Bus Error (/BERR)

This is an input that goes directly to the 68000. It is used to indicate the occurrence of some kind of bus error. Any expansion card capable of detecting a bus error relating directly to that card can assert /BERR when that bus error condition is detected. At other times, the card must monitor /BERR and be prepared to tri-state all of its on-bus output buffers whenever this signal is asserted. Since any number of devices may assert /BERR, and all bus cards must monitor it, any device that drives /BERR must drive with an open collector or similar device capable of sinking at least 12ma, and any device that monitors /BERR should place as little load on it as possible (1 "F" type load or less, per board, is suggested). This signal is connected to a low valued on-board pullup resistor, and shouldn't need any more pulling up. Pin 46.

System Reset (/RST, /BUSRST)

Pin 53 of the bus contains the /RST signal, pin 94 contains the /BUSRST signal. Both of these reflect system reset, however, the /RST signal is bidirectional, unbuffered, and in common with the original 68000 reset signal. It should only be used on boards that are capable of resetting the system. The /BUSRST signal is a buffered output-only version of the reset signal that should be used as the normal reset input to boards not concerned with resetting the system on their own. The /RST signal is connected to a medium valued on-board pullup resistor and shouldn't need any more pulling up.

System Halt (/HLT)

This is the 68000's processor halt signal, tied directly to the 68000. It is connected to a medium valued on-board pullup resistor and shouldn't need any more pulling up. This signal, when driven by a PIC, will halt and tri-state the 68000 at the end of the current bus cycle. If driven by the 68000, it indicates detection of a double bus fault. Pin 55.

System Interrupts

Six of the 68000 interrupts are available on the Expansion Bus, and these are labelled as /INT2, /INT6, /EINT1, /EINT4, /EINT5, /EINT7. The interrupt structure of the original ZORRO specification has been slightly changed for the A2000/B2000. This change affects the availability of decoded interrupt inputs and multiplexed interrupt inputs. Specifically, the 68000 accepts 7 levels of interrupt that are presented to it as 8 possible values priority encoded into 3 multiplexed inputs. The original ZORRO specification called for decoded interrupt inputs on pin 19 for interrupt level 2 (/INT2), and on pin 22 for interrupt level 6 (/INT6). These are the same interrupts used by the Amiga internal system chips and encoded by the Paula chip. The interrupts could be used by external devices by wired ORing interrupt requests into one of these available interrupts. The original ZORRO

bus also provides the encoded interrupt lines /IPL0, /IPL1, and /IPL2 on bus pins 40, 42, and 44 respectively. These are useless as inputs, but as outputs are required by any Coprocessor or alternate processor that needs to monitor system interrupts. In the A2000/B2000 scheme, coprocessors sit in the Coprocessor Slot which allows them full control of the system. The encoded interrupt lines have been replaced with decoded interrupt lines that may be freely used as inputs; interrupt levels 7 (/EINT7), 5 (/EINT5), and 4 (/EINT4) are available now on bus pins 40, 42, and 44 respectively, and the level 1 interrupt (/EINT1) is available on bus pin 96 (which is left open in the ZORRO specification). See Appendix for pin list.

Override (/OVR)

The /OVR, or Override, signal is a special Amiga expansion signal that can serve two purposes. The signal can basically turn off the onboard decoding of system memory ranges, including those used by the Amiga custom chips. As a result of this, it can also turn off internally generated things, like /DTACK.

The timing in the A500 and B2000, based on the Gary chip (not the PALs of the older machines) effectively prohibits the use of OVR* for the area outside of $200000 to $9FFFFF. Due to the buffering delays of the Expansion Bus, this signal should never be used for overlay on a PIC.

The other use of this signal is better supported. Asserting /OVR will tri-state the internally generated /DTACK signal, allowing a Coprocessor or Expansion device to create its own /DTACK. The same effect can be achieved for most applications by using XRDY to delay the motherboard's generation of /DTACK. Pin 17.

External Ready (XRDY)

This input provides a way for an external device to delay the motherboard generated /DTACK, for things like slow memory and I/O boards that need to add wait states. This signal should be negated very quickly, no later than 60ns from address valid (/AS asserted), in order for the motherboard circuitry to have enough time to prevent the normal assertion of /DTACK. XDRY should stay negated for as many wait states are required. Once XRDY is asserted, /DTACK completes the rest of the normal cycle. XRDY is a wired-OR input; it is pulled up by a resistor on the motherboard, and should be driven with an open collector or equivalent output. Pin 18.

SLOT CONTROL SIGNALS

This group of signals is responsible for the control of things that happen between Expansion Slots.

Slave (/SLAVEn)

Pin 9 is the SLAVEn signal, where "n" refers to the Expansion Slot number. Each Slot has its own SLAVE output, all of which go into the collision detect circuitry. Whenever a PIC is responding to a decoded address range, it must assert its SLAVE output within 35 ns. The SLAVE output must be negated at the end of a cycle within 50 ns. If a more than one SLAVE output occurs for the same address, or if a PIC asserts its SLAVE output for an address reserved by the local bus, a collision is registered and results in /BERR being asserted.

Configuration Chain (/CFGINn, /CFGOUTn)

Pins 11 and 12 are, respectively, the /CFGOUTn and /CFGINn signals, where "n" refers to the Expansion Slot number. Each Slot has its own version of each signal, which make up the configuration chain between Slots. Each subsequent /CFGIN is a result of all previous /CFGOUTs, going from slot 1 to slot 5 on the Expansion Bus. On the B2000, the 86 pin coprocessor has CONFIG priority 0, which chains directly into Expansion Slot 1. This enforces the order of autoconfiguration between slots. During the autoconfiguration process, an unconfigured PIC responds to the 64K address space starting at $E80000 if its CFGIN signal is asserted. All unconfigured PICs come up with CFGOUT negated. When configured, or told to "shut up", a PIC will assert is CFGOUT, which results in the CFGIN of the next slot to be asserted. On-board logic automatically passes on the state of the previous CFGOUT to the next CFGIN for any slot not occupied by a PIC, so there's no need to sequentially populate the Expansion Bus Slots.

Data Output Enable (DOE)

This signal is used by an expansion card to enable the buffers on the data bus. The signal's timing changes from read cycle to write cycle. Pin 93.

DMA CONTROL SIGNALS

There are various signals on the Expansion Bus that coordinate the arbitration of DMAs that may be requested by devices on the Expansion Bus.

PIC is DMA Owner (/OWN)

Asserted by Expansion Bus DMA device when it becomes bus master. This output is to be treated as a wired-OR output between all Expansion Slots, any of which may have a PIC signalling bus mastership. Thus, this should be driven with an open-collector or similar output by any PIC using it. Found on pin 7.

Slot Specific Bus Arbitration (/BRn, /BGn)

Pins 60 and 64 are, respectively, the /BRn and /BGn signals, where "n" refers to the Expansion Slot number. Each Slot has its own version of each signal. The Bus Request and Bus Grant from each board go to some prioritization circuitry, and then to the 68000. Slot 1 has the highest priority, Slot 5 the lowest, out of the Expansion Slots. On a B2000, the Coprocessor Slot is included in this priority chain when its not acting as a coprocessor, and it acts as priority level 0, right before that of slot 1. Note that along with the request prioritization logic, the bus requests are clocked by the rising edge of the 7M clock, and its a very good idea for any PIC requesting the bus to similarly clock its Bus Request output. This design prohibits any astable or race conditions that can occur when two PICs desire to own the bus asynchronously. Found on pins 60, 64, respectively.

Bus Grant Acknowledge (/BGACK)

This is the unbuffered 68000 /BGACK signal. Any PIC that receives a bus grant from the 68000 should assert this signal as long as the DMA continues, releasing it once the DMA request is finished. This signal should never be asserted until the Bus Grant has been received, AS is negated, DTACK is negated, and BGACK itself is negated, indicating that all other potential bus masters have relinquished the bus. This output is driven as a wired-OR output, so all devices driving it must drive it with an open collector or equivalent device. Pin 62.

Processor Bus Grant (/BG, /GBG)

The A1000 and A2000 systems receive the the /BG (bus grant) signal from the 68000 directly, unchanged, in addition to the slot specific /BGn signals. This was actually a late change to the original ZORRO specification, so it may not be on every A1000 ZORRO expansion box. This has changed slightly on the B2000 system as part of the coprocessor interface. The B2000's bus pin 95 is /GBG, Generic Bus Grant. When the 68000 is in charge, /GBG is essentially a buffered /BG. When the coprocessor is in charge, /GBG is a buffered /CBG. This allows all cards in the expansion bus to function without concern as to which processor is actually controlling the bus.

RESERVED PINS

Pins 96, 97, and 98 have been left open for future expansion.

100 PIN CONNECTOR PINOUTS

There are three instances of the Expansion Bus (so far), the original A1000/ZORRO specification, and the A2000 enhancement to this original spec, and the B2000 (A2000-CR) specification. The ZORRO specification is treated as a single instance for the purposes of this chart, even though there are several different ZORRO bus implementations from several different hardware manufacturers.

PIN	ZORRO	A2000	B2000	Buffered?	Function
1	X	X	X	N/A	Ground
2	X	X	X	N/A	Ground
3	X	X	X	N/A	Ground
4	X	X	X	N/A	Ground
5	X	X	X	N/A	+5VDC
6	X	X	X	N/A	+5VDC
7	X	X	X	N/A	/OWN
8	X	X	X	N/A	-5VDC
9	X	X	X	N/A	/SLAVEn
10	X	X	X	N/A	+12VDC
11	X	X	X	N/A	/CFGOUTn
12	X	X	X	N/A	/CFGINn
13	X	X	X	N/A	Ground
14	X	X	X	Yes	/C3 Clock
15	X	X	X	Yes	CDAC Clock
16	X	X	X	Yes	/C1 Clock
17	X	X	X	No	/OVR
18	X	X	X	No	XRDY
19	X	X	X	No	/INT2
20	X			N/A	No Connect
		X	X	N/A	-12VDC
21	X	X	X	Yes	A5
22	X	X	X	No	/INT6
23	X	X	X	Yes	A6
24	X	X	X	Yes	A4
25	X	X	X	N/A	Ground
26	X	X	X	Yes	A3
27	X	X	X	Yes	A2
28	X	X	X	Yes	A7
29	X	X	X	Yes	A1
30	X	X	X	Yes	A8
31	X	X	X	Yes	FC0
32	X	X	X	Yes	A9
33	X	X	X	Yes	FC1
34	X	X	X	Yes	A10
35	X	X	X	Yes	FC2
36	X	X	X	Yes	A11
37	X	X	X	N/A	Ground
38	X	X	X	Yes	A12
39	X	X	X	Yes	A13
40	X			No	/IPL0
		X	X	No	/EINT7

PIN	ZORRO	A2000	B2000	Buffered?	Function
41	X	X	X	Yes	A14
42	X		X	Yes	/IPL1
		X	X	Yes	/EINT5
43	X	X	X	Yes	A15
44	X		X	No	/IPL2
		X	X	No	/EINT4
45	X	X	X	Yes	A16
46	X	X	X	No	/BEER
47	X	X	X	Yes	A17
48	X	X	X	No	/VPA
49	X	X	X	N/A	Ground
50	X	X	X	No	E Clock
51	X	X	X	N/A	/VMA
52	X	X	X	Yes	A18
53	X	X	X	No	/RST
54	X	X	X	Yes	A19
55	X	X	X	No	/HLT
56	X	X	X	Yes	A20
57	X	X	X	Yes	A22
58	X	X	X	Yes	A21
59	X	X	X	Yes	A23
60	X	X	X	N/A	/BRn
61	X	X	X	N/A	Ground
62	X	X	X	No	/BGACK
63	X	X	X	Yes	D15
64	X	X	X	N/A	/BGn
65	X	X	X	Yes	D14
66	X	X	X	No	/DTACK
67	X	X	X	Yes	D13
68	X	X	X	Yes	READ
69	X	X	X	Yes	D12
70	X	X	X	Yes	/LDS
71	X	X	X	Yes	D11
72	X	X	X	Yes	/UDS
73	X	X	X	N/A	Ground
74	X	X	X	Yes	/AS
75	X	X	X	Yes	D0
76	X	X	X	Yes	D10
77	X	X	X	Yes	D1
78	X	X	X	Yes	D9
79	X	X	X	Yes	D2
80	X	X	X	Yes	D8
81	X	X	X	Yes	D3
82	X	X	X	Yes	D7
83	X	X	X	Yes	D4
84	X	X	X	Yes	D6
85	X	X	X	N/A	Ground
86	X	X	X	Yes	D5
87	X	X	X	N/A	Ground
88	X	X	X	N/A	Ground

PIN	ZORRO	A2000	B2000	Buffered?	Function
89	X	X	X	N/A	Ground
90	X	X	X	N/A	Ground
91	X	X	X	N/A	Ground
92	X			N/A	EQU7MHz
		X	X	No	7MHz
93	X	X	X	N/A	DOE
94	X	X	X	Yes	/BUSRST
95	X	X		No	/BG
			X	Yes	/GBG
96				N/A	No Connect
		X	X	No	/EINT1
97	X	X	X	N/A	No Connect
98	X	X	X	N/A	No Connect
99	X	X	X	N/A	Ground
100	X	X	X	N/A	Ground

Coprocessor Expansion and 86 Pin Signals

INTRODUCTION

This section details the signals found on the various types of 86 pin expansion connectors on different Amiga computers, especially the signals found on the B2000 computer's 86 pin Coprocessor Slot, and how these differ from the similar signals found on A2000 computers and those of the original A1000 computers. This paper also explains the Coprocessor Slot's autoconfiguration and DMA protocols and how they fix the problems introduced in the A2000 Coprocessor Slot.

Changes from Previous Documents

We've kept the 86 pin specification on the B2000 as similar to those available on the A2000, A1000 and A500, wherever possible. However, some major changes were absolutely required. With the design of the A2000, the function of the 86 pin slot had shifted from a general expansion connector to expansion specifically intended for coprocessors and similar devices. Thus, while the A500's and A1000's 86 pin connectors have to support both some kind of coprocessor expansion and the normal ZORRO expansion, the A2000 machines can optimize each slot for its purpose if required (or if necessary, which is more the case).

The 86 pin connector on the A500 and A1000 becomes something of an advantage, because of the fact that all expansion must be done externally. When a coprocessor device, something that needs to completely replace the 68000 in all forms of bus access and operation (like a 68020 accelerator card) is added, it can physically sit between the computer motherboard and the 100 pin expansion box, thus allowing the device to completely replace the action of the motherboard's processor from the point of view of the expansion box. A machine with both slots on the motherboard must provide some facility to logically insert the 86 pin slot in front of the 100 pin slot for certain applications.

In the A2000, the Coprocessor Slot signals that control DMA can be used to insert the coprocessor in the place of the normal 68000 via the standard 68000 DMA request protocol. This, however, isn't a totally transparent replacement; the action of the coprocessor taking control over the local bus from the 68000, in the A2000, can block other DMA events coming over from the 100 Expansion Bus. For total control of the Expansion Bus on the A2000, the 68000 could be physically removed from the motherboard, but that would result in the "coprocessor" being a complete "replacement" processor, with no swapping between the two permissible. The B2000 solves these problems with a higher-level DMA protocol between the main and coprocessor devices.

COPROCESSOR SLOT SIGNALS

The Coprocessor Slot signals discussed below apply for most of the machines, though in some cases the item mentioned exists on only some of the machines; these are specified. Most of these signals are directly in common with the 68000, or directly a part of the 68000 local bus, instead of being buffered as on the Expansion Bus. No signal on a Coprocessor card should load the Local Bus with more than one "F" series standard load.

POWER CONNECTIONS

The Coprocessor Slot provides several different voltages designed to supply Coprocessor devices. The A2000 power supply is currently rated at 200 Watts, which supplies the main board and all other expansion ports, as well as the Coprocessor Slot.

Digital Ground (Ground)

Digital supply ground used by all expansion cards as the return path for all expansion supplies. This is found on all instances of the Local Bus ports. See p. 98 for pin assignments.

Main Supply (+5V)

Main power supply the Coprocessor slot, and can supply up to 2.0 Amps of +5VDC on the A2000. The maximum supply current for the entire A2000 system is 20 Amps for all devices inside the A2000 that use +5V, including the motherboard. The corresponding pins on the Expansion Edge of the A1000 can source only 1 Amp, and even less on the A500. Pins: 5, 6.

Negative Supply (−5V)

Negative version of the main supply, for small current loads only; there's a total of 0.3 Amp for the entire A2000 system.
This pin is similar to what's available on the A1000 and A500, though these other instances will have different currents available. Pin: 8.

High Voltage Supply (+12V)

Higher voltage supply, useful for communications cards and other devices requiring greater than digital voltage levels. This is intended for small loading only; there's a total of 8 Amps for the entire A2000 system, much of which is normally devoted to floppy and hard disk drive motors. Found on all instances of Local Bus Ports, though the available currents may vary. Pin: 10.

CLOCK SIGNALS

There are various system clocks available at all Local Bus Ports, useful in designing synchronous Coprocessor systems. Loading on these clocks should be watched very carefully on all types of Amiga computers.

/C1 Clock

This is a 3.58 MHz clock synched to the falling edge of the 7.16 MHz system clock. Also known as /CCK in some places. Pin 16.

/C3 Clock

This is a 3.58 MHz clock synched to the rising edge of the 7.16 MHz system clock. Also known as /CCKQ in some places. Pin 14.

CDAC Clock

This is a 7.16 MHz clock that leads the 7.16 MHz system clock by about 70ns (90 degrees). Pin 15.

E Clock

This is the 68000 generated "E" clock, used for 6800 family peripherals driven by "E" and 6502 peripherals driven by PHI2. This clock is six 7.16 MHz clocks high, four clocks low, as per the 68000 spec. This clock is always generated by the 68000, regardless of the state of the bus and the Coprocessor; this fact should be considered by the Coprocessor implementor when designing any Coprocessor /VMA logic. Pin 50.

7MHz Clock

This is the 7.16 MHz system clock. This is available only on the B2000 at this pin, and is in common with the 68000's clock input. This pin, pin 7, is unused on all other Local Bus Port instances. Many applications that run on systems without the 7MHz clock create a 7MHz equivalent clock, using the relationship 7MHzEQU = /C1 XNOR /C3; care must be taken in considering any additional delays that this equivalent clock causes on systems other that the B2000.

28MHz Clock

This is the 28.64 MHz fundamental clock used to derive all other system clocks under normal operation. There's no guaranteed phase relationship between this clock and the system clocks. When the system is being driven by an external clock source via XCLK and /XCLKEN, this clock will essentially be completely asynchronous to the system clocks. It is provided mainly to provide a fast clock for fast coprocessors. This is pin 9 on the Coprocessor Slot, and is an unused pin on the Expansion Edge of the A500 and A1000.

ADDRESSING AND CONTROL SIGNALS

These signals are various items used for the addressing of resources on a coprocessor card by the 68000 and any DMA devices, and for 24 by 16 bit addressing of other system resources by a coprocessor device (which may easily have more potential). Most of these signals are directly in common with 68000 signals.

Read-Write (R/W)

The 68000's R/W output. When driven high it indicates a read or internal cycle, when driven low it indicates a write cycle. When the coprocessor takes over it drives this line; the 68000's output will tri-state. Pin 68.

Address Bus (A1-A23)

This directly connects to the 68000's address bus, providing 16 megabytes of address space with 23 bits of address for a 16 bit data bus. The 68000 is capable of driving only this much address space. Thus, any resources on a coprocessor board must map somewhere into the 68000 memory space. The best thing to do with any such memory is allow it to be autoconfigured by the 1.2 OS; this will place it somewhere in the 8 megabyte space starting at $200000 (the A2000 doesn't support autoconfiguration from the Coprocessor Slot, the B2000 does). Any resources intended specifically for the coprocessor only can be located above the 68000's 16 megabyte space if the coprocessor hardware permits that extended addressing. All board and Expansion bus resources will normally map into the first 16 megabytes of the address space of a coprocessor board. See p. 98 for pin list.

Address Strobe (/AS)

The falling edge of this strobe indicates that addresses are valid, the rising edge signals the end of the memory cycle. This is in common with the 68000 /AS signal. The coprocessor drives this signal when it takes over; the 68000's will tri-state. Found on pin 74.

Data Bus (D0-D15)

This is directly connected to the 68000's data bus, providing 16 bits of data accessible by word or either byte. Any coprocessor handling words larger than 16 bits must either step down to 16 bits on its own or provide circuitry to convert the 16 bit word size of the main board and Expansion Bus to the natural size of such a coprocessor, when accessing main board resources. See p. 98 for pin list.

Data Strobes (/LDS, /UDS)

These are the 68000's upper and lower data strobes. The strobes fall on data valid during transfer; the lower strobe being used for the lower byte (even byte address), the upper strobe being used for the upper byte (odd byte address). Like /AS, these must be driven by the Coprocessor as it assumes control, as the 68000 pins will tri-state. Pins: 70, 72.

Valid Memory Address (/VMA)

Output from the 68000 indicating a valid address for 6800 style peripheral devices, in response to a /VPA input. This output goes tri-state when the Coprocessor takes over from the 68000, and as such must be re-created by the coprocessor in response to a VPA signal from somewhere on the motherboard. Pin 51.

Valid Peripheral Address (/VPA)

Input to the 68000 indicating the address has selected a 6800 or 6502 style peripheral, so the 6800 style peripheral access should take place. When the 68000 has given up the bus to the Coprocessor, this input is ignored and must be handled by the Coprocessor board. Pin 48.

Data Transfer Acknowledge (/DTACK)

This signal is the 68000's Data Transfer Acknowledge input, though it's being driven on the motherboard under most conditions. Normally in the Amiga system, Amiga system logic creates /DTACK for a simple, no-wait state cycle (this may be varied by the custom chips). Therefore, this signal is treated as an output to the Expansion and Coprocessor Slots, for most situations. Any slow device on the bus that needs to control /DTACK may do so by negating XRDY to hold off /DTACK or asserting /OVR very quickly to tri-state /DTACK. Any coprocessor must be able to support this action by Expansion boards as well. Note that depending upon when /AS is asserted by a bus master when accessing the CHIP memory, one of two possible cycles may result. If /AS is asserted during C1 low, C3 low, the bus cycle is considered "in-sync," and will proceed, with /DTACK driven as for a normal 4 tick clock cycle. If instead, /AS is asserted during C1 high, C3 high, the bus cycle is considered "out of sync" and the internally generated /DTACK will be held off, causing a wait state that's designed to "sync-up" the DMA cycle with the custom chip's memory cycle. Of course, when a coprocessor is accessing any of its on-board resources, the designer can implement any reasonable data transfer scheme that comes to mind. This signal is on pin 66.

Processor Status (FC0-FC2)

These signals are the 68000 Processor Status outputs, which can be used by bus devices to determine the internal state of the 68000 any time /AS is asserted. When a coprocessor is in charge, it must drive these pins in a way compatible with how the 68000 does it. The different 68000 status codes can be found in any 68000 spec sheet. Pins 31, 33, 35.

Bus Error (/BERR)

This is an input that goes directly to the 68000. Its used to indicate the occurrence of some kind of bus error. Any Expansion Card capable of detecting a bus error relating directly to that card can assert /BERR when that bus error condition is detected. At other times, the card must monitor /BERR and be prepared to tri-state all of its on-bus output buffers whenever this signal is asserted. The Coprocessor card won't have to tri-state on /BERR, but it must note it and provide some way of handling the occurrence (the 68000 under normal Amiga OS control merely signals a Guru Error based on the Bus Error Exception). Since any number of devices may assert /BERR, and nearly everything in the system must monitor it, any device that drives /BERR must drive with an open collector or similar device capable of sinking at least 12ma, and any device that monitors /BERR should place as little load on it as possible (1 "F" type load or less, per board, is suggested). This signal is connected to a low valued on-board pullup resistor, and shouldn't need any more pulling up. Pin 46.

System Reset (/RST)

Pin 53 of the bus contains the /RST signal which is in common with the original 68000 reset signal. The /RST signal is bidirectional, and the 68000 tri-states it when the coprocessor takes over. It is only necessary for the processor to output this signal if it needs to reset the system under program control. The /RST signal is connected to a medium valued on-board pullup resistor and shouldn't need any more pulling up. The coprocessor must monitor this signal and respond to it appropriately; this may mean a complete reset, but it doesn't have to. The Coprocessor can also assert this line if a system reset is desired.

System Halt (/HLT)

This is the 68000's processor halt signal, tied directly to the 68000. It is connected to a medium valued on-board pullup resistor and shouldn't need any more pulling up. This signal, when asserted, will halt and tri-state the 68000 at the end of the current bus cycle. If driven by the 68000, it indicates detection of a double bus fault. For a complete system reset, the 68000 looks for both the /RST and /HLT lines to be asserted. The Coprocessor should handle this signal in a similar fashion. Pin 55.

Decoded Interrupts

Two of the 68000 non-encoded interrupt inputs are available at the Coprocessor slot, on pin 19 for interrupt level 2 (/INT2) and on pin 22 for interrupt level 6 (/INT6). These are the same interrupts used by the Amiga internal system chips and encoded by the Paula chip. They can be used by a Coprocessor board by driving them to generate 68000 interrupts when the 68000 is in charge, though generally they don't do much when the Coprocessor is in charge.

Encoded Interrupts (/IPL0-/IPL2)

The Coprocessor Slot provides the encoded interrupt lines /IPL0, /IPL1, and /IPL2 on bus pins 40, 42, and 44 respectively, which are the normal encoded interrupt inputs to the 68000. Nothing on the Coprocessor slot can drive these lines, but they must be monitored by any Coprocessor or alternate processor that needs to be able to respond to any system interrupts when acting as the bus master.

Override (/OVR)

The /OVR, or Override, signal is a special Amiga expansion signal that can serve two purposes. The signal can basically turn off the on-board decoding of system memory ranges. As a result of this, it can also turn off internally generated things, like /DTACK.

The timing in the A500 and B2000, based on the Gary chip (not the PALs of the older machines) effectively prohibits the use of OVR* for the area outside of $200000 to $9FFFFF.

The other use of this signal is better supported. Asserting /OVR will tri-state the internally generated /DTACK signal, allowing a Coprocessor or Expansion device to create its own /DTACK. The same effect can be achieved for most applications by using XRDY to delay the motherboard's generation of /DTACK. Pin 17.

External Ready (XRDY)

This input provides a way for an external device to delay the motherboard generated /DTACK, for things like slow memory and I/O boards that need to add wait states. This signal should be negated very quickly, no later than 60ns from address valid (/AS asserted), in order for the motherboard circuitry to have enough time to prevent the normal assertion of /DTACK. XDRY should stay negated for as many wait states as required. Once XRDY is asserted, /DTACK completes the rest of the normal cycle. XRDY is a wired-OR input; it is pulled up by a resistor on the motherboard, and should be driven with an open collector or equivalent output. Pin 18.

Configuration Chain (/COPCFG)

Pins 11 and 12 are basically the configuration IN and configuration OUT signals. Pin 12, the configuration IN input, is grounded on all versions of the Local Bus Ports, indicating that this Slot is the first in any configuration chain and may proceed with configuration. On the A500, A1000, and A2000, the configuration OUT signal, pin 12, is a no-connect. Because of this, its impossible to normally autoconfigure any device in the Coprocessor slot of an A2000. On the B2000, pin 11 is a true configuration OUT signal, which becomes the configuration IN input to the first Expansion Slot. This, the coprocessor slot is configured first on the B2000. A note of caution here, though. All normal Expansion Bus devices assert their /SLAVE output whenever they respond to an address. This SLAVE output allows the collision detect circuitry to determine if multiple devices are responding to the same address. When a collision is detected this way, the /BERR signal is asserted, causing all PICs to tri-state, and saving both these PICs and the Expansion Bus drivers from any potentially destructive buffer fights. While the Coprocessor slot on the B2000 can be automatically configured, it can't assert a SLAVE signal for collision detect. Thus, designers must be very careful with any autoconfiguring resources on a Coprocessor card.

During the autoconfiguration process, first the Coprocessor card, then all an unconfigured PICs in turn, respond to the 64K address space starting at $E80000 as their respective CFGIN signals are asserted. All unconfigured PICs come up with CFGOUT negated. When configured, or told to "shut up", the Coprocessor Card or any PIC should assert CFGOUT, which results in the CFGIN of the next slot to be asserted. On-board logic automatically passes on the state of the previous CFGOUT to the next CFGIN for any slot not occupied by a PIC, so there's no need to sequentially populate the Expansion Bus Slots and no need to have the Coprocessor Card do any autoconfiguring if real autoconfiguration isn't necessary.

DMA AND COPROCESSOR SIGNALS

This will be covered in more detail in the next section, but this section covers the basic signals involved in DMAs and the Coprocessor interface.

Bus Request (/BR, /CBR)

All instances of Local Expansion Ports have a Bus Request to 68000 of some kind. In the A2000, as in the A500 and A1000, this is directly connected to the 68000's /BR input, which is considered a wired-OR input; all devices driving this input must technically drive it with an open collector or equivalent driver. In actuality, the A500 and A1000 don't use this at all internally, so a standard driver may be used if necessary. The A2000's /BR input is shared by the /BR output of the DMA arbitration logic, so this will be necessary on an A2000 Coprocessor Slot device. The B2000 has in place of the 68000's /BR line a special bus request all its own, /CBR. In both cases, the signal is an input to the 68000 used to request mastership of the Local Bus. The signal is found on pin 60.

Bus Grant (/BG, /CBG)

All instances of Local Expansion Ports have a Bus Grant of some kind from the 68000. In the A2000, as in the A500 and A1000, this is directly connected to the 68000's /BG output. In the B2000, a Coprocessor specific Bus Grant signal, /CBG, is in its place. In either case, the signal is asserted by the 68000 in response to a Bus Request. This indicates to the device in the Coprocessor slot that the 68000 will fully relinquish the bus at the end of this cycle. A /BG received on the Coprocessor Slot in an A2000 could be a Grant given in response to an Expansion Bus DMA request as well as one in response to the Coprocessor Slot DMA request. On the B2000, /CBG will only be asserted if the Coprocessor Slot is granted the bus. This signal is found on pin 64.

Bus Grant Acknowledge (/BGACK)

This is the 68000's /BGACK, or Bus Grant Acknowledge, signal. Any device that receives a bus grant from the 68000 should assert this signal as long as the DMA continues, releasing it once the DMA request is finished. This signal should never be asserted until the specific Bus Grant has been received, /AS is negated, /DTACK is negated, and /BGACK itself is negated, indicating that all other potential bus masters have relinquished the bus. This output is driven as a wired-OR output, so all devices driving it must drive it with an open collector or equivalent device. Pin 62.

Coprocessor Grant Acknowledge (/BOSS)

This signal exists only on the B2000, on pin 20. That pin is unused on both the A2000 and the A500. Originally, this pin was called /PALOPE on the A1000, and was part of the planned ROM expansion method. This is currently obsolete; the method of ROM expansion was changed to work without the need for such a signal. On the B2000, the /BOSS signal is driven by a Coprocessor instead of /BGACK when the Coprocessor wishes the DMA access granted it to be a true Coprocessor access, not a simple DMA. This is all explained in the following section on the B2000 coprocessor interface.

THE B2000 COPROCESSOR INTERFACE

The B2000 computer implements an extended version of the A2000's Coprocessor Slot, designed to make the swapping of main processors under program control much more powerful and transparent to the rest of the B2000 system. There are things that can be done from the B2000 Coprocessor slot that can't be done from the A2000's Coprocessor Slot, so this is an important consideration to anyone designing a Coprocessor device of some kind.

Normal 68000 DMA Architecture

The 68000 supports hardware signals designed to permit a simple DMA protocol. This protocol allows multiple devices to take control of the 68000's data, address, and control buses. When a device of some kind desires direct access to the 68000's bus, it asserts the /BR (Bus Request) input of the 68000. Once /BR is asserted, the 68000 will complete whatever operation it's doing to the point it can cleanly relinquish its bus. At this point, it will assert its /BG (Bus Grant) output, telling the device requesting DMA that it's just about ready to shut down. The requesting device then issues /BGACK (Bus Grant Acknowledge) as soon as the 68000 is completely off the bus (DTACK and /AS are negated). When the DMAing device is done with the bus, it releases /DTACK and /BR, and the 68000 will then release /BG.

Where the 68000 DMA Protocol Fails

The above protocol, as implemented in the 68000, is sufficient for many types of DMA operation, especially for simple things in which there are single DMA devices on the bus. What this doesn't easily account for are multiple DMA devices. While the /BR and /BGACK inputs to the 68000 can be wire-ORed to support several devices, there are still problems with this scheme. Should multiple devices request DMA at the same time, the 68000 will see nothing different than if only one device is requesting DMA. While careful monitoring of the /BGACK by responding potential bus masters can solve some of the problems, there are much cleaner approaches to this problem.

One such solution is implemented in the ZORRO and A2000/B2000 Expansion Buses. Each slot on the Expansion Bus has its own private Bus Request and Bus Grant. Each Bus Request signal is considered by a priority encoding and latching circuit. The result is that if simultaneous Bus Requests come in from Expansion Slots, only the Slot given higher priority will actually get a Bus Grant. Any Bus Requests that come in while another DMA is in effect are held off until the 68000's/BG line has been negated for at least one tick. this circuitry, part of the original ZORRO specification, eliminates the problems that can occur with various DMA devices all competing for the Expansion and Local Buses.

The B2000 Coprocessor Solution

The B2000 hardware has implemented a more sophisticated Coprocessor system that removes these problems. The B2000 Coprocessor Slot has a signal called /CBR (Coprocessor Bus Request) as a replacement for /BR, a signal called /CBG (Coprocessor Bus Grant) as a replacement for /BG, and one additional signal, /BOSS, which is also known as Coprocessor Grant Acknowledge.

Under the B2000 system, there are essentially two ways a Coprocessor device can receive a Local Bus mastership. Both start in the same way. To request the bus, the Coprocessor asserts /CBR. Instead of going directly to the 68000, this signal is prioritized and latched along with any Expansion Slot /BR signals. The /CBR signal has the highest DMA priority. Assuming no other DMAs are currently active, the 68000 issues a Bus Grant via /BG, which will go to the prioritizer[1] and result in /CBG being asserted. At this point, all other DMA requests will be locked out; no other /BGs of any kind will be issued. Following the normal 68000 protocol, at this point, the Coprocessor will assert /BGACK when the 68000 is off the bus, and will have bus access as before. And as before, it is holding off any further DMAs from the Expansion Bus (which may be what was wanted). This type of DMA access is very similar to what a normal DMA device from the Expansion Bus would achieve.

There is another way to take over the Bus. This starts in the same manner as before, with a /CBR resulting in a /CBG. Once the Coprocessor has received its Bus Grant, however, it does something different. It asserts the /BOSS signal instead of /BGACK. This has several immediate effects. First of all, the 68000 sees /BOSS as the same thing as /BGACK, so it stays off the bus just as if /BGACK had been asserted. Next, the data direction of /CBR and /CBG change on the Coprocessor Bus. The /CBR signal is now an output from the bus control logic, the prioritized and latched combination of all the /BR signals from the Expansion Bus. The /CBG signal is now an input going into the bus control logic that will be passed on to the Expansion Bus in response to an Expansion Bus /BR. The bus control logic also holds /BR to the 68000 in a low state. The data direction of /CBR and /CBG changes with a change in /BOSS, so the lines that alternately drive /CBR and /CBG on a Coprocessor card should be enabled and disabled with the assertion of /BOSS.

Anyway, what all this means is that, in asserting /BOSS instead of /BGACK, the Coprocessor has the bus, the 68000 is in tri-state, and any of the Expansion Slots may initiate a DMA of the Coprocessor at any time, directly, according to the normal /BR → /BG → /BGACK protocol of the 68000. The Coprocessor can allow the 68000 back on the bus by negating the /BOSS line. Thus, the Coprocessor can be a real Coprocessor, functioning as the equivalent of the 68000 for all things as far as the whole Amiga system is concerned.

---

[1]The B2000 system does all of its DMA prioritization via the "Buster" custom bus controller chip.

86 PIN CONNECTOR PINOUTS

Here are the four instances of the 86 pin Local Bus, the A500 and A1000 Edge connectors, used for all kinds of expansion on those machines, and the A2000 and B2000 Coprocessor slots.

PIN	A500	A1000	A2000	B2000	Function
1	X	X	X	X	Ground
2	X	X	X	X	Ground
3	X	X	X	X	Ground
4	X	X	X	X	Ground
5	X	X	X	X	+5VDC
6	X	X	X	X	+5VDC
7	X	X	X	X	No Connect
8	X	X	X	X	-5VDC
9	X	X			No Connect
			X	X	28MHz Clock
10	X	X	X	X	-12VDC
11	X	X	X		No Connect
				X	/COPCFG (Configuration Out)
12	X	X	X	X	CONFIG IN, Grounded
13	X	X	X	X	Ground
14	X	X	X	X	/C3 Clock
15	X	X	X	X	CDAC Clock
16	X	X	X	X	/C1 Clock
17	X	X	X	X	OVR
18	X	X	X	X	RDY
19	X	X	X	X	/INT2
20		X			/PALOPE
	X		X		No Connect
				X	/BOSS
21	X	X	X	X	A5
22	X	X	X	X	/INT6
23	X	X	X	X	A6
24	X	X	X	X	A4
25	X	X	X	X	Ground
26	X	X	X	X	A3
27	X	X	X	X	A2
28	X	X	X	X	A7
29	X	X	X	X	A1
30	X	X	X	X	A8
31	X	X	X	X	FC0
32	X	X	X	X	A9
33	X	X	X	X	FC1
34	X	X	X	X	A10
35	X	X	X	X	FC2
36	X	X	X	X	A11
37	X	X	X	X	Ground
38	X	X	X	X	A12

PIN	A500	A1000	A2000	B2000	Function
39	X	X	X	X	A13
40	X	X	X	X	/IPL0
41	X	X	X	X	A14
42	X	X	X	X	/IPL1
43	X	X	X	X	A15
44	X	X	X	X	/IPL2
45	X	X	X	X	A16
46	X	X	X	X	/BEER
47	X	X	X	X	A17
48	X	X	X	X	/VPA
49	X	X	X	X	Ground
50	X	X	X	X	E Clock
51	X	X	X	X	/VMA
52	X	X	X	X	A18
53	X	X	X	X	/RST
54	X	X	X	X	A19
55	X	X	X	X	/HLT
56	X	X	X	X	A20
57	X	X	X	X	A22
58	X	X	X	X	A21
59	X	X	X	X	A23
60	X	X	X		/BR
				X	/CBR
61	X	X	X	X	Ground
62	X	X	X	X	/BGACK
63	X	X	X	X	D15
64	X	X	X		/BG
				X	/CBG
65	X	X	X	X	D14
66	X	X	X	X	/DTACK
67	X	X	X	X	D13
68	X	X	X	X	R/W
69	X	X	X	X	D12
70	X	X	X	X	/LDS
71	X	X	X	X	D11
72	X	X	X	X	/UDS
73	X	X	X	X	Ground
74	X	X	X	X	/AS
75	X	X	X	X	D0
76	X	X	X	X	D10
77	X	X	X	X	D1
78	X	X	X	X	D9
79	X	X	X	X	D2
80	X	X	X	X	D8
81	X	X	X	X	D3
82	X	X	X	X	D7

PIN	A500	A1000	A2000	B2000	Function
83	X	X	X	X	D4
84	X	X	X	X	D6
85	X	X	X	X	Ground
86	X	X	X	X	D5

High Voltage Supply (+12V)

Higher voltage supply, intended for small loading only; there's a total of 8 Amps for the entire A2000 system, much of which is normally devoted to floppy and hard disk drive motors. Pin: 10.

CLOCK SIGNALS

These are various clock signals useful for synchronous timing of video peripherals.

/C1 Clock

For NTSC, this is a 3.58 MHz clock that's synched to the falling edge of the 7.16 MHz system clock. Also known as /CCK in some places. Pin 34. For PAL, these frequencies are 3.55 MHz and 7.09 MHz respectively.

/C4 Clock

For NTSC, this is a 3.58 MHz clock that's synched to the rising edge of the 7.16 MHz CDAC clock. Pin 19. Again, for PAL, these frequencies are 3.55 MHz and 7.09 MHz respectively.

External Clock (XCLK, / XCLKEN)

The video slot provides for an external system clock, generally used to cause the entire A2000 system to become synchronized to something external. This should be something very close to the 28.64 MHz clock normally used to drive the system; the value used for XCLK can be a somewhat higher frequency, although anything too high will cause memory and other system timings to break down. XCLK will only be engaged as the system clock when /XCLKEN is asserted. XCLK is found on pin 33, /XCLKEN is on pin 16. There is no fixed phase relationship between XCLK and internal clocks and video outputs. Video interfaces must synchronize to the output clocks/ video.

VIDEO SIGNALS

The main point of this slot is access to the video signals generated by the Amiga's custom video chips. Most of these are also found on the 23 pin external video connector.

Analog Video

This is the analog RGB output, which consists of Red, Green, and Blue signals, each of which generates a 0.7V p-p, 47 Ohm terminated analog output. Found, respectively, on pins 7, 11, and 15.

Digital Video

These signals serve as digital output, suitable for use with an IBM or Commodore 128 style 4 bit digital color or monochrome monitor or similar output device. On the B2000, these (in conjunction with

The Amiga 2000 Video Slot

INTRODUCTION

This document details the signals found on the internal video slot of the Amiga 2000 (A2000), and the additional component of this slot as implemented on the B2000 model. The A2000 video connector is a 36 pin edge connector, mechanically similar to the slot extension connector of an IBM PC-AT. The B2000 adds a second 36 pin connector, directly in front of the first one, that supplies additional audio/video information. Where possible, a device should use only the first slot, thus maintaining compatibility with both A2000 and B2000. Of course, there are quite a few things that can't be accomplished with the A2000 connector alone.

ORIGINAL A2000 SLOT

The original A2000 video slot was designed to provide the functionality of the 23 pin external video connector in a form that could internally house video boards such as modulators, genlocks, etc.

POWER CONNECTIONS

The Video Slot provides several different voltages designed to supply Video devices. The A2000 power supply is currently rated at 200 Watts, which supplies the main board and all other expansion ports as well as the Video Slot.

Video Ground

Video supply ground used by all video devices and the internal video circuitry. Currently on the B2000, the Video and Digital grounds are common signals, while on the A2000 these are distinct. This is available on pins 9, 12, 13, 17, 20, 21, 24, 32.

Main Supply (+5V)

Main digital level power supply for the Video Slot. This can supply large currents, on the order of 2 Amps or so for the Video Slot. The maximum supply current for the entire A2000 system is 20 Amps for all devices inside the A2000 that use +5V, including the motherboard. Pins: 6, 8.

Negative Supply (−5V)

Negative version of the main supply, for small current loads only; there's a total of 0.3 Amp for the entire A2000 system. Pin: 31.

other signals found on the second video connector) provide access to the full 12 bits of digital video output produced on the motherboard by the Denise chip (4 bits each of R, G, and B). Each of these outputs is 47 Ohm terminated. The pin assignments are Digital Red (R3) on pin 29, Digital Green (G3) on pin 27, Digital Blue (B3) on pin 25, and Digital Intensity (B0) on pin 23.

Separate Sync (/HSYNC, /VSYNC)

These are the separate, bidirectional, 47 Ohm terminated video frame synchronization clocks. The horizontal sync, /HSYNC, is on pin 22; the vertical sync, /VSYNC, is on pin 26. As the names imply, these sync signals are active low.

Composite Sync (/CSYNC, COMP SYNC)

Two versions of a composite synchronization signal are available. Pin 14, /CSYNC, is an unterminated digital level composite sync; pin 28, COMP SYNC, is a buffered TTL version of the combined synchronization clocks.

Burst

NTSC/PAL colorburst. Pin 18. To obtain the correct PAL colorburst signal, the video plug-in card must multiply this signal by 1.25 (i.e., 3.55*1.25 = 4.433 MHz).

Pixel Switch (/PIXELSW)

Background color indicator (color 0), on a pixel by pixel basis. 47 Ohm terminated, /PIXELSW, pin 30.

AUDIO SIGNALS

Along with access to video signals, audio signals are available at the Video Slot. The audio signals are the Left and Right audio channels, on pins 3 and 4 respectively.

RESERVED FOR EXPANSION

The original Video Slot has pins 1, 2, 5, 35, and 36 reserved for future expansion.

B2000 EXTENDED VIDEO SLOT

The B2000 Extended Video Slot was designed to provide nearly every internal video signal available, plus additional audio signals and some control lines too. This slot allows much more complex and powerful devices to be placed in the video slot.

POWER CONNECTIONS

The Extended Video Slot provides several different voltages designed to supply Video devices. The A2000 power supply is currently rated at 200 Watts, which supplies the main board and all other expansion ports as well as the Video Slot.

Digital/Video Ground (GROUND)

These pins provide additional grounding for digital or video based devices. Pins 1, 5, 9, 12, 22, and 32.

Audio Ground

These pins provide grounding in common with the separate on-board audio ground. Pins 34, 36.

CLOCK SIGNALS

These are various clock signals useful for synchronous timing of video peripherals.

CDAC Clock

For NTSC, this is a 7.16 MHz clock that leads the 7.16 MHz system clock by about 70ns (90 degrees). Pin 15. For a PAL system, this is 7.09 MHz.

/C3 Clock

For NTSC, this is a 3.58 MHz clock that's synched to the rising edge of the 7.16 MHz system clock. Also known as /CCKQ in some places. Pin 17. For a PAL system, this is 3.55 MHz.

Timer Time Base (TBASE)

This is the real time clock time-base input, either 50Hz or 60Hz, depending on the country involved and the setting of the Time Base Jumper. The jumper can select either line frequency or vertical synchronization as the clock's time base. Pin 14.

VIDEO SIGNALS

The main point of this slot is access to more of the video signals generated by the Amiga's custom video chips. Most of the signals available here aren't available on any external port.

Composite Video

This is the analog level monochrome Composite Video signal also available on the Composite Video jack of the B2000. Pin 13.

other signals found on the second video connector) provide access to the full 12 bits of digital video output produced on the motherboard by the Denise chip (4 bits each of R, G, and B). Each of these outputs is 47 Ohm terminated. The pin assignments are Digital Red (R3) on pin 29, Digital Green (G3) on pin 27, Digital Blue (B3) on pin 25, and Digital Intensity (B0) on pin 23.

Separate Sync (/HSYNC, /VSYNC)

These are the separate, bidirectional, 47 Ohm terminated video frame synchronization clocks. The horizontal sync, /HSYNC, is on pin 22; the vertical sync, /VSYNC, is on pin 26. As the names imply, these sync signals are active low.

Composite Sync (/CSYNC, COMP SYNC)

Two versions of a composite synchronization signal are available. Pin 14, /CSYNC, is an unterminated digital level composite sync; pin 28, COMP SYNC, is a buffered TTL version of the combined synchronization clocks.

Burst

NTSC/PAL colorburst. Pin 18. To obtain the correct PAL colorburst signal, the video plug-in card must multiply this signal by 1.25 (i.e., 3.55*1.25 = 4.433 MHz).

Pixel Switch (/PIXELSW)

Background color indicator (color 0), on a pixel by pixel basis. 47 Ohm terminated, /PIXELSW, pin 30.

AUDIO SIGNALS

Along with access to video signals, audio signals are available at the Video Slot. The audio signals are the Left and Right audio channels, on pins 3 and 4 respectively.

RESERVED FOR EXPANSION

The original Video Slot has pins 1, 2, 5, 35, and 36 reserved for future expansion.

B2000 EXTENDED VIDEO SLOT

The B2000 Extended Video Slot was designed to provide nearly every internal video signal available, plus additional audio signals and some control lines too. This slot allows much more complex and powerful devices to be placed in the video slot.

Digital Video

The remaining 8 bits of digital video are available on this connector. The signals are Red 0-2 (pins 2, 3, 4), Green 0-2 (pins 6, 7, 8), and Blue 1-2 (pins 10 and 11). The timing of the digital video is not tightly specified. Developers wishing to use this should contact Commodore for further details.

LIGHT PEN (/LPEN)

This is an input to the Agnus light pen input. This signal should go low in response to the lighting of a pixel on a video display monitor. The Agnus chip latches the raster position that was in effect when the /LPEN signal goes low, so an application can follow the position of a light pen on the screen. Pin 19.

PORT CONNECTIONS

Most of the signals from the bidirectional parallel port (printer port) are available on this connector as well, along with a few others.

8 Bit Parallel Port (PD0-PD7)

The 8 bit bidirectional parallel port most commonly used to drive a Centronics interface printer externally is accessable here. It can be used to control various aspects of a complex video interface device. The port lines PD0-PD7 are on pins 23 to 30 of this connector.

Parallel Port Handshake (/ACK)

This is the acknowledge (/ACK) input, the same as the acknowledge input to the parallel port. Driving this with an output from a Video Card can cause a level 2 interrupt to occur through the 8520 CIA device this is connected to, based on the programming of an 8520 register. On pin 20.

Other Port Lines (BUSY, POUT, SEL)

Connector pins 18 (BUSY) and 16 (POUT) are general purpose I/O signals that together can also function as a synchronous serial data port driven by an 8520 CIA device. In normal printer use, the BUSY signal is used to indicate printer buffer full to the Amiga, while POUT is used to indicate the printer paper is out. For serial port usage, BUSY is the serial clock, POUT is the serial data line. These should be driven with open collector devices if the Video Card uses them as inputs to the 8520. The SEL signal, on pin 21, is a general purpose I/O port, usually used as a device select signal on the parallel port.

The expanded B2000 video slot is a 36 pin edge connector, the same type as used on the 16 bit IBM style bus extension.

PIN	Signal	PIN	Signal
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	Composite Video	14	TBASE
15	CDAC Clock	16	POUT
17	/C3 Clock	18	BUSY
19	/LPEN	20	/ACK
21	SEL	22	Ground
23	PD0	24	PD1
25	PD2	26	PD3
27	PD4	28	PD5
28	PD6	30	PD7
31	/LED	32	Ground
33	Raw Audio Left	34	Audio Ground
35	Raw Audio Right	36	Audio Ground

Description of PC/XT Emulator for AMIGA 2000

AMIGA ACCESS: Amiga Interface Offset Address = Base Addr.

Base Addr. + (00000 – 1FFFF) : Byte Access
Base Addr. + (20000 – 3FFFF) : Word Access
Base Addr. + (40000 – 5FFFF) : Graphic Access
Base Addr. + (60000 – 7FFFF) : I/O Register Access

INTERFACE MEMORY MAP:

Interface Offset Address	Size	Usage
00000 . . . 0FFFF	64K	DISK BUFFER RAM
10000 . . . 17FFF	32K	COLOR VIDEO RAM
18000 . . . 1BFFF	16K	PARAMETER RAM
1C000 . . . 1DFFF	8K	MONO VIDEO RAM
1E000 . . . 1FFFF	8K	IO-PAGE

Kinds of memory access on the following pages:
B = Byte access
G = Graphic access
W = Word access

(*) selectable by BIT 5 and 6 of the MODE REGISTER

BIT 5 = SEL1
BIT 6 = SEL2

PC MEMORY AND I/O MAP:

PC Address	Range	Size	Usage	kind of access	Amiga Interface Offset Address
0000 ...	03FF	1K	IO-PAGE	B	1E000 ... 1FFFF
				W	3E000 ... 3FFFF
				G	5E000 ... 5FFFF
					7E000 ... 7FFFF
A0000 ...	AFFFF	64K	DISK BUFFER RAM (*)	B	00000 ... 0FFFF
			(*)	W	20000 ... 2FFFF
			(*)	G	40000 ... 4FFFF
B0000 ...	B1FFF	8K	MONO VIDEO RAM	B	1C000 ... 1DFFF
				W	3C000 ... 3DFFF
				G	5C000 ... 5DFFF
B8000 ...	BFFFF	32K	COLOR VIDEO RAM	B	10000 ... 17FFF
				W	30000 ... 37FFF
				G	50000 ... 57FFF
E0000 ...	EFFFF	64K	DISK BUFFER RAM (*)	B	00000 ... 0FFFF
			(*)	W	20000 ... 2FFFF
			(*)	G	40000 ... 4FFFF
D0000 ...	DFFFF	64K	DISK BUFFER RAM (*)	B	00000 ... 0FFFF
			(*)	W	20000 ... 2FFFF
			(*)	G	40000 ... 4FFFF
F0000 ...	F3FFF	16K	PARAMETER RAM	B	18000 ... 1BFFF
				W	38000 ... 3BFFF
				G	58000 ... 5BFFF

AMIGA MEMORY MAP:

Amiga Interface Offset Address	PC Address Range	Size	Usage	kind of access
00000 ... 0FFFF	A0000 ... AFFFF	64K	DISK BUFFER RAM (*)	B
00000 ... 0FFFF	D0000 ... DFFFF	64K	DISK BUFFER RAM (*)	B
00000 ... 0FFFF	E0000 ... EFFFF	64K	DISK BUFFER RAM (*)	B
10000 ... 17FFF	B8000 ... BFFFF	32K	COLOR VIDEO RAM	B
18000 ... 1BFFF	F0000 ... F3FFF	16K	PARAMETER RAM	B
1C000 ... 1DFFF	B0000 ... B1FFF	8K	MONO VIDEO RAM	B
1E000 ... 1FFFF	0000 ... 03FF	1K	IO-PAGE	B
20000 ... 2FFFF	A0000 ... AFFFF	64K	DISK BUFFER RAM (*)	W
20000 ... 2FFFF	D0000 ... DFFFF	64K	DISK BUFFER RAM (*)	W
20000 ... 2FFFF	E0000 ... EFFFF	64K	DISK BUFFER RAM (*)	W
30000 ... 37FFF	B8000 ... BFFFF	32K	COLOR VIDEO RAM	W
38000 ... 3BFFF	F0000 ... F3FFF	16K	PARAMETER RAM	W
3C000 ... 3DFFF	B0000 ... B1FFF	8K	MONO VIDEO RAM	W
3E000 ... 3FFFF	0000 ... 03FF	1K	IO-PAGE	W
40000 ... 4FFFF	A0000 ... AFFFF	64K	DISK BUFFER RAM (*)	G

40000 . . . 4FFFF	D0000 . . . DFFFF	64K	DISK BUFFER RAM (*)	G
40000 . . . 4FFFF	E0000 . . . EFFFF	64K	DISK BUFFER RAM (*)	G
50000 . . . 57FFF	B8000 . . . BFFFF	32K	COLOR VIDEO RAM	G
58000 . . . 5BFFF	F0000 . . . F3FFF	16K	PARAMETER RAM	G
5C000 . . . 5DFFF	B0000 . . . B1FFF	8K	MONO VIDEO RAM	G
5E000 . . . 5FFFF	0000 . . . 03FF	1K	IO-PAGE	G
7E000 . . . 7FFFF	0000 . . . 03FF	1K	IO-PAGE	

AT MEMORY and I/O MAP:

PC Address	Range	Size	Usage	kind of access	Amiga Interface Offset Address
0000 . . .	03FF	1K	IO-PAGE	B	1E000 . . . 1FFFF
				W	3E000 . . . 3FFFF
				G	5E000 . . . 5FFFF
					7E000 . . . 7FFFF
A0000 . . .	AFFFF	64K	DISK BUFFER RAM (*)	B	00000 . . . 0FFFF
			(*)	W	20000 . . . 2FFFF
			(*)	G	40000 . . . 4FFFF
B0000	B1FFF	8K	MONO VIDEO RAM	B	1C000 . . . 1DFFF
				W	3C000 . . . 3DFFF
				G	5C000 . . . 5DFFF
B8000 . . .	BFFFF	32K	COLOR VIDEO RAM	B	10000 . . . 17FFF
				W	30000 . . . 37FFF
				G	50000 . . . 57FFF
D0000 . . .	D3FFF	16K	PARAMETER RAM	B	18000 . . . 1BFFF
				W	38000 . . . 3BFFF
				G	58000 . . . 5BFFF
D4000 . . .	DFFFF	64K	DISK BUFFER RAM (*)	B	04000 . . . 0FFFF
			(*)	W	24000 . . . 2FFFF
			(*)	G	44000 . . . 4FFFF

AMIGA MEMORY MAP:

Amiga Interface Offset Address	AT Address Range	Size	Usage	kind of access
00000 . . . 0FFFF	A0000 . . . AFFFF	64K	DISK BUFFER RAM (*)	B
00000 . . . 03FFF	CAN NOT BE ACCESSED BY THE AT			
04000 . . . 0FFFF	D4000 . . . DFFFF	48K	DISK BUFFER RAM (*)	B
10000 . . . 17FFF	B8000 . . . BFFFF	32K	COLOR VIDEO RAM	B
18000 . . . 1BFFF	D0000 . . . D3FFF	16K	PARAMETER RAM	B
1C000 . . . 1DFFF	B0000 . . . B1FFF	8K	MONO VIDEO RAM	B
1E000 . . . 1FFFF	0000 . . . 03FF	1K	IO-PAGE	B
20000 . . . 2FFFF	A0000 . . . AFFFF	64K	DISK BUFFER RAM (*)	W
20000 . . . 23FFF	CAN NOT BE ACCESSED BY THE AT			

AMIGA MEMORY MAP:

Amiga Interface Offset Address	AT Address Range	Size	Usage	kind of access
24000 ... 2FFFF	D4000 ... DFFFF	48K	DISK BUFFER RAM (*)	W
30000 ... 37FFF	B8000 ... BFFFF	32K	COLOR VIDEO RAM	W
38000 ... 3BFFF	D0000 ... D3FFF	16K	PARAMETER RAM	W
3C000 ... 3DFFF	B0000 ... B1FFF	8K	MONO VIDEO RAM	W
3E000 ... 3FFFF	0000 ... 03FF	1K	IO-PAGE	W
40000 ... 4FFFF	A0000 ... AFFFF	64K	DISK BUFFER RAM (*)	G
40000 ... 43FFF	CAN NOT BE ACCESSED BY THE AT			
44000 ... 4FFFF	D4000 ... DFFFF	48K	DISK BUFFER (*)	G
50000 ... 57FFF	B8000 ... BFFFF	32K	COLOR VIDEO RAM	G
58000 ... 5BFFF	D0000 ... D3FFF	16K	PARAMETER RAM	G
5C000 ... 5DFFF	B0000 ... B1FFF	8K	MONO VIDEO RAM	G
5E000 ... 5FFFF	0000 ... 03FF	1K	IO-PAGE	G
7E000 ... 7FFFF	0000 ... 03FF	1K	IO-PAGE	

PC/AT I/O REGISTER MAP

PC/AT I/O Address	Usage			Offset Address INTERFACE / AMIGA	
60	KEYBOARD DATA		(W)	1E41F	7E41F
61	SYSTEM REGISTER		(W)	1E05F	7E05F
62	SYSTEM STATUS		(W)	1E03F	7E03F
2F8	COM2 TRANSMIT DATA	(DLAB = 0)	(W)	1E07D	7E07D
2F8	" RECEIVE DATA	(DLAB = 0)	(R)	1E09D	7E09D
2F8	" RESET IRQ3_b	(DLAB = 0)	(R)	1E09D	7E09D
2F9	" INTERRUPT CONTROL	(DLAB = 0)	(W)	1E0BD	7E0BD
2F9	" INTERRUPT CONTROL	(DLAB = 0)	(R)	1E0DD	7E0DD
2F8	" DIVISOR LATCH (LSB)	(DLAB = 1)	(R/W)	1E07F	7E07F
2F8	" RESET IRQ3_b	(DLAB = 1)	(R)	1E07F	7E07F
2F9	" DIVISOR LATCH (MSB)	(DLAB = 1)	(R/W)	1E09F	7E09F
2FA	COM2 INTERRUPT ACKN		(R)	1E0FF	7E0FF
2FA	DUMMY		(W)	1E01F	7E01F
2FB	" LINE CONTROL	(DLAB = BIT 7)	(W)	1E11F	7E11F
2FB	DUMMY		(R)	1E01F	7E01F
2FC	" MODEM CONTROL		(W)	1E13F	7E13F
2FC	DUMMY		(R)	1E01F	7E01F
2FD	" LINE STATUS		(R)	1E15F	7E15F
2FD	DUMMY		(W)	1E01F	7E01F
2FE	" MODEM STATUS		(R)	1E17F	7E17F
2FE	DUMMY		(W)	1E01F	7E01F
2FF	DUMMY		(R/W)	1E01F	7E01F
378	LPT1 PRINTER DATA		(R/W)	1E19F	7E19F
379	" STATUS		(R)	1E1BF	7E1BF

PC/AT I/O Address	Usage			Offset Address INTERFACE / Amiga		
379	"	RESET IRQ7		(R)	1E1BF	7E1BF
379	"	INTERRUPT CONTROL		(W)	1E19F	7E19F
		BIT 6 = 0 : ON				
		BIT 6 = 0 : OFF				
37A	"	CONTROL		(W)	1E1DF	7E1DF
37A	"	CONTROL		(R)	1E19F	7E19F
3B0	MONO	CRT ADDRESS INDEX REGISTER		(W)	1E1FF	7E1FF
3B0	"	RESET IRQ3_a		(R)	1E01F	7E01F
3B2	"	CRT ADDRESS INDEX REGISTER		(W)	1E1FF	7E1FF
3B2		DUMMY		(R)	1E01F	7E01F
3B4	"	CRT ADDRESS INDEX REGISTER		(W)	1E1FF	7E1FF
3B4		DUMMY		(R)	1E01F	7E01F
3B6	"	CRT ADDRESS INDEX REGISTER		(W)	1E1FF	7E1FF
3B6		DUMMY		(R)	1E01F	7E01F
3B1	"	CRT DATA	REGISTER	(R/W)		s.b.
3B3	"	CRT DATA	REGISTER	(R/W)		s.b.
3B5	"	CRT DATA	REGISTER	(R/W)		s.b.
3B7	"	CRT DATA	REGISTER	(R/W)		s.b
			LAST WRITE ON INDEX = 00		1E2A1	7E2A1
			LAST WRITE ON INDEX = 01		1E2A3	7E2A3
			LAST WRITE ON INDEX = 02		1E2A5	7E2A5
			LAST WRITE ON INDEX = 03		1E2A7	7E2A7
			LAST WRITE ON INDEX = 04		1E2A9	7E2A9
			LAST WRITE ON INDEX = 05		1E2AB	7E2AB
			LAST WRITE ON INDEX = 06		1E2AD	7E2AD
			LAST WRITE ON INDEX = 07		1E2AF	7E2AF
			LAST WRITE ON INDEX = 08		1E2B1	7E2B1
			LAST WRITE ON INDEX = 09		1E2B3	7E2B3
			LAST WRITE ON INDEX = 0A		1E2B5	7E2B5
			LAST WRITE ON INDEX = 0B		1E2B7	7E2B7
			LAST WRITE ON INDEX = 0C		1E2B9	7E2B9
			LAST WRITE ON INDEX = 0D		1E2BB	7E2BB
			LAST WRITE ON INDEX = 0E		1E2BD	7E2BD
			LAST WRITE ON INDEX = 0F		1E2BF	7E2BF
3B8	MONO	CONTROL REGISTER		(W)	1E2FF	7E2FF
3BA	MONO	STATUS REGISTER		(R)	——	——
	BIT 0	:	H-SYNC (18KHz)			
	BIT 3	:	V-SYNC (50 Hz)			
3BA	DUMMY			(W)	1E01F	7E01F
3BB	DUMMY			(R/W)	1E01F	7E01F
3BC	DUMMY			(R/W)	1E01F	7E01F
3BD	DUMMY			(R/W)	1E01F	7E01F
3BE	DUMMY			(R/W)	1E01F	7E01F
3BF	DUMMY			(R/W)	1E01F	7E01F

PC/AT I/O Address	Usage			Offset Address INTERFACE/	Amiga
3D0	COLOR	CRT ADDRESS INDEX REGISTER	(W)	1E21F	7E21F
3D0		DUMMY	(R)	1E01F	7E01F
3D2	"	CRT ADDRESS INDEX REGISTER	(W)	1E21F	7E21F
3D2		DUMMY	(R)	1E01F	7E01F
3D4	"	CRT ADDRESS INDEX REGISTER	(W)	1E21F	7E21F
3D4		DUMMY	(R)	1E01F	7E01F
3D6	"	CRT ADDRESS INDEX REGISTER	(W)	1E21F	7E21F
3D6		DUMMY	(R)	1E01F	7E01F
3D1	"	CRT DATA REGISTER	(R/W)		s.b.
3D3	"	CRT DATA REGISTER	(R/W)		s.b.
3D5	"	CRT DATA REGISTER	(R/W)		s.b.
3D7	"	CRT DATA REGISTER	(R/W)		s.b
		LAST WRITE ON INDEX = 00		1E2C1	7E2C1
		LAST WRITE ON INDEX = 01		1E2C3	7E2C3
		LAST WRITE ON INDEX = 02		1E2C5	7E2C5
		LAST WRITE ON INDEX = 03		1E2C7	7E2C7
		LAST WRITE ON INDEX = 04		1E2C9	7E2C9
		LAST WRITE ON INDEX = 05		1E2CB	7E2CB
		LAST WRITE ON INDEX = 06		1E2CD	7E2CD
		LAST WRITE ON INDEX = 07		1E2CF	7E2CF
		LAST WRITE ON INDEX = 08		1E2D1	7E2D1
		LAST WRITE ON INDEX = 09		1E2D3	7E2D3
		LAST WRITE ON INDEX = 0A		1E2D5	7E2D5
		LAST WRITE ON INDEX = 0B		1E2D7	7E2D7
		LAST WRITE ON INDEX = 0C		1E2D9	7E2D9
		LAST WRITE ON INDEX = 0D		1E2DB	7E2DB
		LAST WRITE ON INDEX = 0E		1E2DD	7E2DD
		LAST WRITE ON INDEX = 0F		1E2DF	7E2DF
3D8	COLOR	CONTROL REGISTER	(W)	1E23F	7E23F
3D8	DUMMY		(R)	1E01F	7E01F
3D9	COLOR	SELECT REGISTER	(W)	1E25F	7E25F
3D9	DUMMY		(R)	1E01F	7E01F
3DA	COLOR	STATUS REGISTER	(R)	—	—
		BIT 0 : H-SYNC (18KHz)			
		BIT 3 : V-SYNC (50 Hz)			
3DA	DUMMY		(W)	1E01F	7E01F
3DD	DISPLAY SYSTEM REGISTER		(W)	1E29F	7E29F
3DD	DUMMY		(R)	1E01F	7E01F
3DE	DUMMY		(R/W)	1E01F	7E01F
3DF	DUMMY		(R/W)	1E01F	7E01F

AMIGA I/O MEMORY MAP (REGISTER DESCRIPTION)

AMIGA	Register	Interface / Memory		Offset Address INTERFACE	/AMIGA
AMIGA	INTERRUPT STATUS	read register		1FF1	7FF1
PC	INTERRUPT STATUS	read register		1FFF3	7FFF3
NEGATE PC RESET		read register		1FFF5	7FFF5
MODE REGISTER		read register /	write register	1FFF7	7FFF7
INTERRUPT MASK		read memory /	write register	1FFF9	7FFF9
PC INTERRUPT CONTROL		read memory /	write register	1FFFB	7FFFB
CONTROL REGISTER		read memory /	write register	1FFFD	7FFFD
KEYBOARD REGISTER		read memory /	write register	1FFFF	7FFFF

PC SIDE

System Status Register:

How to Enable/Disable Interrupts from Amiga to PC

A write access to this register (i/o location 62 hex) forces a /SYSINT interrupt on the AMIGA side

A write access to bit 6 of i/o location 379 hex enables/disables the AMIGA forced interrupts IRQ1 (keyboard), IRQ3 (serial interface COM2) and IRQ7 (parallel interface LPT1) as follows:

D6	Function
0	interrupts enabled
1	interrupts disabled

Note:

The access to i/o location 379 hex is enabled if PARON is high. That is, the AMIGA has to write a "1" to MODE REGISTER bit 1. (See 'Amiga Mode Register.')

The following initialization routine must be used to allow an external printer card on the pc side:

```
AMIGA: set MODE REGISTER bit 1 to "1"  ; switch parallel
                                       ; interface on
PC   : set i/o location 379 hex bit 6 to "0"; keyboard and
                                       ; serial interr. off
AMIGA: set MODE REGISTER bit 1 to "0"  ; switch parallel
                                       ; interface off
```

Now the keyboard and the serial interface emulation is enabled, the parallel interface emulation is disabled.

How to Clear an Asserted Interrupt Signal

INT	Negation
IRQ3_a	Read to i/o location 3b0 hex
IRQ3_b	Read com2 register 2F8 hex
IRQ7	Read line printer status register 379 hex

AMIGA SIDE

All registers on the memory locations 1FFF0 TO 1FFFF are only accessable from the AMIGA side.

Amiga Interrupt Status Register (R) (1FFF1 / 17FFF1)

Reading this register returns the interrupt events caused on PC accesses as follows:

Bit no.	Function	
0	Mono Video Ram	(/MINT)
1	Color Video Ram	(/GINT)
2	Mono CRT	(/CRT1INT)
3	Color CRT	(/CRT2INT)
4	Keyboard Register	(/ENBKB)
5	LPT1 Control Reg	(/LPT1INT)
6	COM2 Data Reg	(/COM2INT)
7	see PC System Status Res	(/SYSINT)

The event was valid if the bit is set to "1". After reading the register all bits turns to "0" automatically and the interrupt flag will be negated.

PC Interrupt Status Register (R) (1FFF3 / 7FFF3)

Reading this register returns the pending PC interrupts on the lower nibble. The PC interrupt is asserted as shown by the corresponding bit in the table.

Bit no.	Function	Asserted if
0	IRQ1 (Keyboard interrupt)	1
1	IRQ3_a	0
2	IRQ3_b	0
3	IRQ7	0
4-7	NOT USED, always HIGH	

Bit 1 and bit 2 (IRQ3_a and IRQ3_b) are externally "ORed" to IRQ3

Negate PC Reset (R) (1FFF5 / 7FFF5)

A read access to this register negates the PC reset line and allows the PC to start the boot procedure. On power-on the PC reset line is asserted (default).

Mode Register (R/W) (1FFF7 / 7FFF7)

Reading this register returns system configuration information

Bit no.	Name	Function
0	SERON	serial interface enabled
1	PARON	parallel interface enabled
2	KEYON	keyboard interface enabled
3	MON	monochrome display emulation enabled
4	COLOR	color display emulation enabled
5	SEL1	select the PC/AT memory bank, s.b.
6	SEL2	select the PC/AT memory bank, s.b.
7	PC/AT	LOW = AT mode HIGH = PC mode

Writing to this register sets system configuration information

Bit no.	Name	Function
0	SERON	switch serial interface on
1	PARON	switch parallel interface on
2	KEYON	switch keyboard interface on
3	MON	enable monochrome display emulation
4	COLOR	enable color display emulation
5	SEL1	PC/AT memory bank select, s.b.
6	SEL2	PC/AT memory bank select, s.b.
7	/STOPCLK	LOW = disable the clock for video retrace and keyboard HIGH = enable the clock for video retrace and keyboard

SEL2	SEL1	PC memory	AT memory
0	0	not used	not used
0	1	A0000 – AFFFF HEX	A0000 – AFFFF HEX
1	0	D0000 – DFFFF HEX	D4000 – DFFFF HEX
1	1	E0000 – EFFFF HEX	not used

Interrupt Mask Register (R/W) (1FFF9 / 7FFF9)

You can mask each PC interrupt event separately by writing a "1" to the corresponding bit as shown below.

Bit no.	Maskable Event (cmp. to Amiga interrupt status reg.)
0	/MINT
1	/GINT
2	/CRT1INT
3	/CRT2INT
4	ENKBKB
5	/LPT1INT
6	/COM2INT
7	/SYSINT

PC Interrupt Control Register (R/W) (1FFFB / 7FFFB)

A PC interrupt can be forced by writing a "0" to the corresponding bit of the lower nibble except the keyboard interrupt, which can be asserted by writing a "1", as shown below:

Bit no.	Asserted PC interrupt level
0	KBSTART (start keyboard shift-register)
1	IRQ3_a (forces interrupt IRQ3)
2	IRQ3_b (forces interrupt IRQ3)
3	IRQ7

Bit 1 and bit 2 (IRQ3_a and IRQ_b) are externally "ORed" to IRQ3

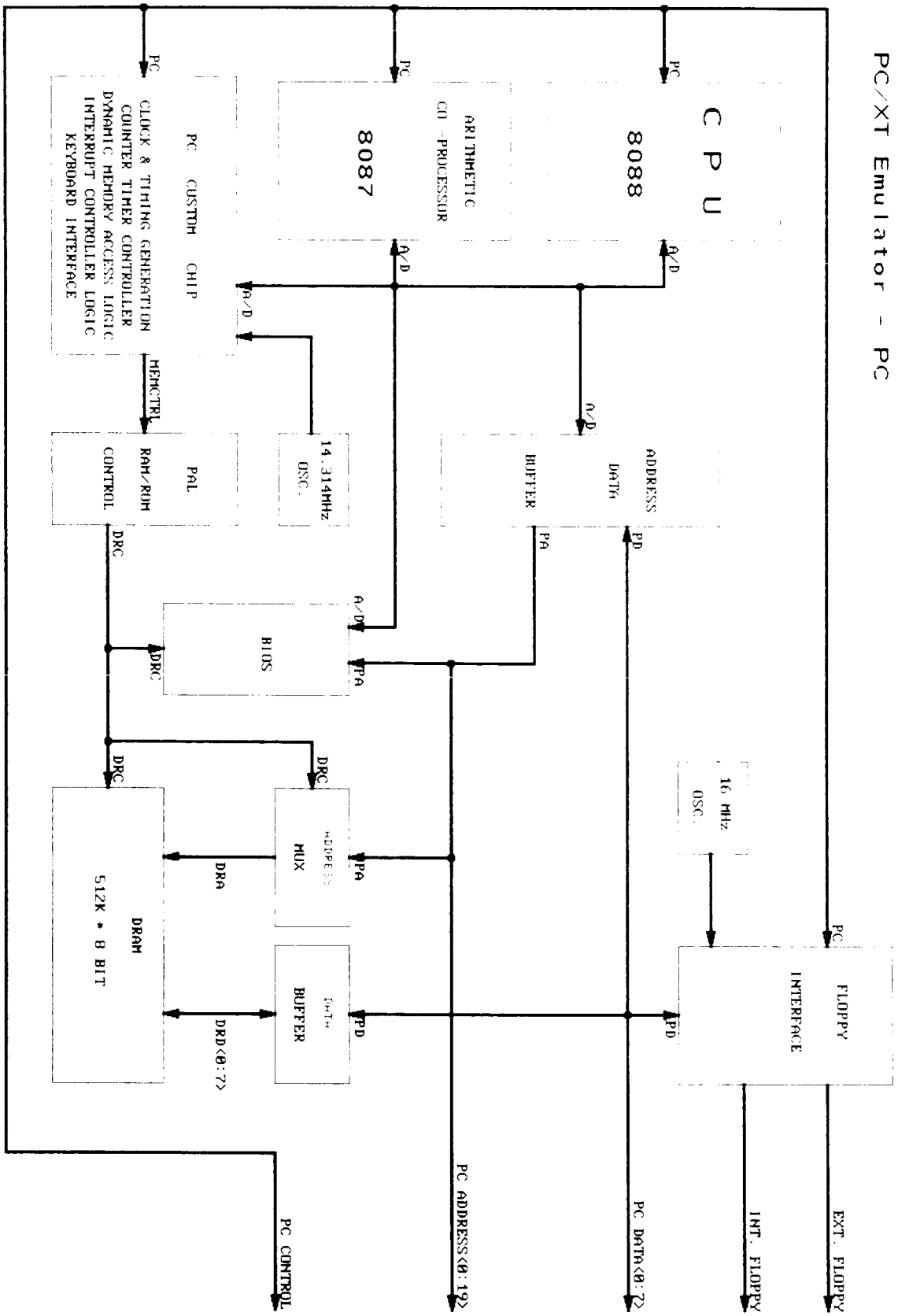
Control Register (R/W) (1FFFD / 7FFFD)

All control function will be done by writing a "0" to the corresponding bit. Only bits 0 to 4 are used.

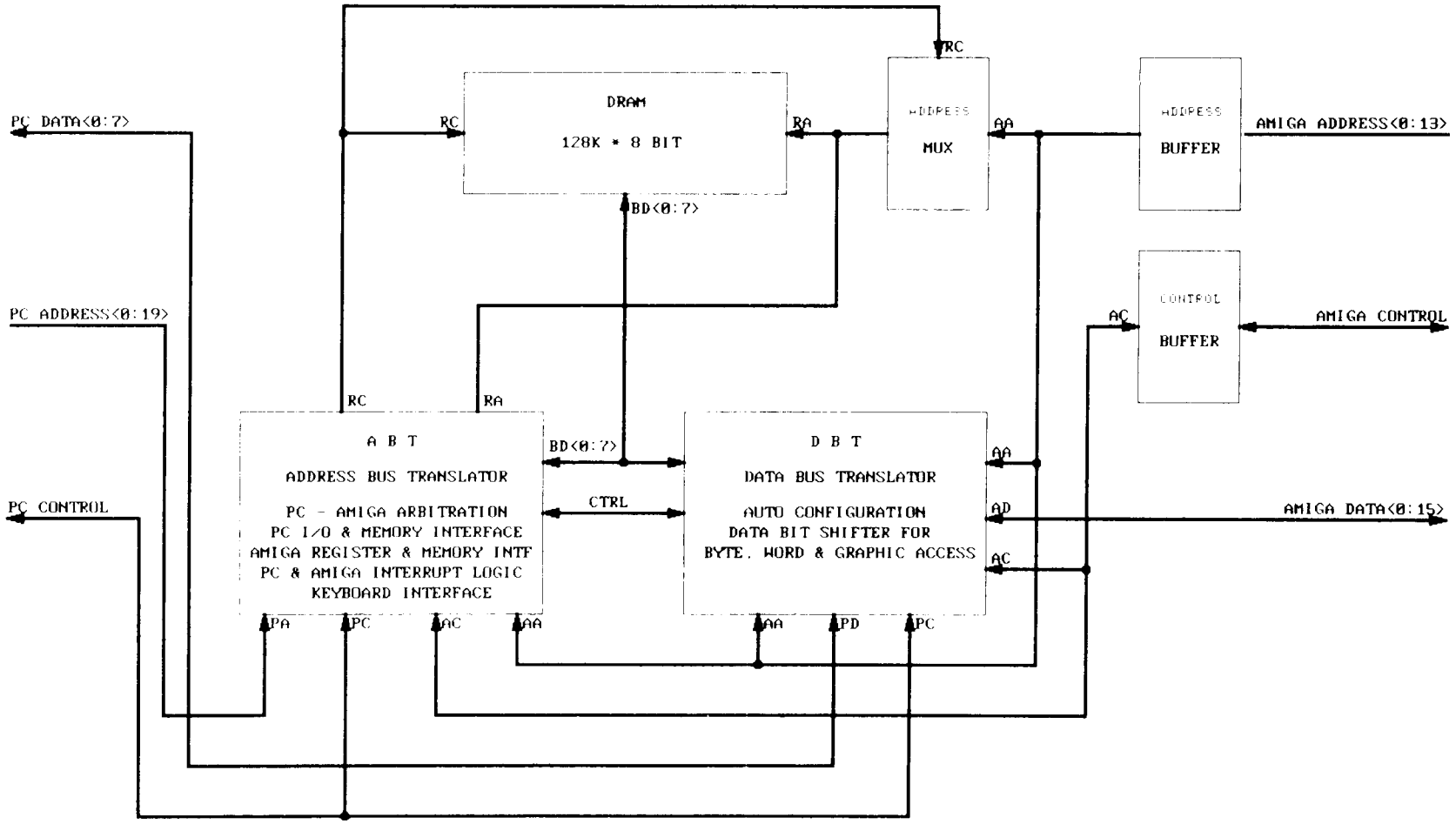
Bit no.	usage
0	general interrupt enable to the AMIGA
1	general interrupt disable to the AMIGA (default)
2	assert the PC reset line
3	negate all PC interrupt levels except the keyboard interrupt
4	reset line printer BUSY (port 379 hex bit 7) The line printer BUSY bit will be set by writing a "1" to bit 0 of port 37A hex from PC side.

Keyboard Register (R/W) (1FFFF / 7FFFF)

Keyboard emulation is done by writing a character to this register and then asserting a "1" to bit 0 of the PC INTERRUPT CONTROL REGISTER.


PC/XT Emulator - PC
C P U
8088
8087
ARITHMETIC
CO-PROCESSOR
PC CUSTOM CHIP
CLOCK & TIMING GENERATION
COUNTER TIMER CONTROLLER
DYNAMIC MEMORY ACCESS LOGIC
INTERRUPT CONTROLLER LOGIC
KEYBOARD INTERFACE
MEMCTRL
PAL
RAM/ROM
CONTROL
14.314MHz
OSC.
BUFFER
ADDRESS
DATA
BIOS
DRC
DRA
DRD<0:7>
DRAM
512K * 8 BIT
MUX
BUFFER
16 MHz
OSC.
FLOPPY
INTERFACE
INT. FLOPPY
EXT. FLOPPY
PC CONTROL
PC ADDRESS<0:19>
PC DATA<0:7>
A/D
PA
PD
PC

PC/XT Emulator - interface



BIOS Entry Points

VIDEO ENTRY POINT VIA S/W INT 10H

SET VIDEO MODE (AH = 00H)

INPUT: AL = VIDEO MODE (0-7)
- 0: 40 × 25 alpha b/w
- 1: 40 × 25 alpha 16 colors
- 2: 80 × 25 alpha b/w
- 3: 80 × 25 alpha 16 colors
- 4: 320 × 200 graphics 4 colors
- 5: 320 × 200 graphics b/w
- 6: 640 × 200 graphics monochrome
- 7: 80 × 25 alpha monochrome

SET CURSOR TYPE (AH = 01H)

INPUT: CH = START LINE OF CURSOR (BITS 0-4)
CURSOR CONTROL OPERATION (BITS 5-6)
- 00 = NON-BLINK
- 01 = DON'T DISPLAY CURSOR
- 10 = BLINK @ 1/16 FIELD RATE
- 11 = BLINK @ 1/32 FIELD RATE

CL = END LINE OF CURSOR (BITS 0-4)

SET CURSOR POSITION (AH = 02H)

INPUT: BH = Page # if CRT mode is 0 -> 3
(0 if graphics or monochrome)
DH = Row # of cursor
DL = Column # of cursor

OUTPUT: None

READ CURSOR POSITION

INPUT: BH = ACTIVE DISPLAY PAGE
(ignored and set to 0 if graphics or monochrome mode)

RETURNED: DH = ROW LOCATION OF CURSOR
DL = COLUMN LOCATION OF CURSOR
CX = CURSOR TYPE

OUTPUT: AX = Undefined (however, we return it unchanged)

READ LIGHT PEN (AH = 04H)

INPUT: None

OUTPUT: AH = 0 if light pen not triggered, 1 if it is
DH = Character row of light pen
DL = Character column of light pen
CH = Pixel row
BX = Pixel column, best estimate

SELECT ACTIVE DISPLAY PAGE (AH = 05H)

INPUT: AL = NEW ACTIVE DISPLAY PAGE

OUTPUT: AX = (?)

SCROLL ACTIVE PAGE UP (AH = 06H)

INPUT: AL = LINES TO SCROLL (CLEAR WINDOW IF 0)
BH = ATTRIBUTE FOR BLANK LINE(S)
CH, CL = ROW/COLUMN OF UPPER LEFT CORNER OF WINDOW
DH, DL = ROW/COLUMN OF LOWER RIGHT CORNER OF WINDOW

OUTPUT: None

SCROLL ACTIVE PAGE DOWN (AH = 07H)

INPUT: AL = LINES TO SCROLL (CLEAR WINDOW IF 0)
BH = ATTRIBUTE FOR BLANK LINE(S)
CH, CL = ROW/COLUMN OF UPPER LEFT CORNER OF WINDOW
DH, DL = ROW/COLUMN OF LOWER RIGHT CORNER OF WINDOW

OUTPUT: None

READ CHAR & ATTRIBUTE AT CURSOR POSITION (AH = 08H)

INPUT: BH = ACTIVE DISPLAY PAGE

OUTPUT: AL = CHARACTER
AH = ATTRIBUTE
(not defined for graphics, however we return an ORing of any and all color bits set as the attribute, a reasonable compromise)

GRAPHICS MODE READ:

OUTPUT: AL = CHAR READ (if recognized, else 0)
AH = ATTRIBUTE (COLOR) (If recognized, else 0)
All characters above 80h are recognized if the RAM font vector is other than 0, else not

WRITE CHAR & ATTRIBUTE AT CURSOR POSITION (AH = 09H)

INPUT: BH = ACTIVE DISPLAY PAGE
CX = NUMBER OF TIMES TO WRITE CHARACTER
AL = CHAR TO WRITE
BL = CHARACTER ATTRIBUTE

OUTPUT: None

WRITE CHAR AT CURSOR POSITION (AH = OAH)

INPUT: BH = ACTIVE DISPLAY PAGE
CX = NUMBER OF TIMES TO WRITE CHARACTER
AL = CHAR TO WRITE
BL = Character Attribute if in a graphics mode otherwise ignored

OUTPUT: None

SET COLOR PALETTE (AH = OBH)

INPUT: BH = 0 FOR BACKGROUND COLOR IN BL
1 FOR COLOR SET NUMBER IN BL
BL = BITS 0-4 IF BH = 0
0 FOR COLOR SET GREEN/RED/YELLOW
F BH = 1
1 FOR COLOR SET CYAN/MAGENTA/WHITE
IF BH = 1

WRITE DOT (AH = OCH)

INPUT: DX = ROW NUMBER (MODE DEPENDENT)
CX = COLUMN NUMBER (MODE DEPENDENT)
AL = COLOR VALUE

OUTPUT: AH = ?

READ DOT (AH = ODH)

INPUT: DX = ROW NUMBER (MODE DEPENDENT)
CX = COLUMN NUMBER (MODE DEPENDENT)

OUTPUT: AH = ?
AL = COLOR VALUE

WRITE TELETYPE (AH = OEH)

INPUT AL = CHARACTER TO BE WRITTEN
BL = FOREGROUND COLOR OF CHAR (USED ONLY IN GRAPHICS MODE)
BH = REQUESTED DISPLAY PAGE (REALLY IS IGNORED)

OUTPUT: None

READ CURRENT VIDEO STATE (AH = OFH)

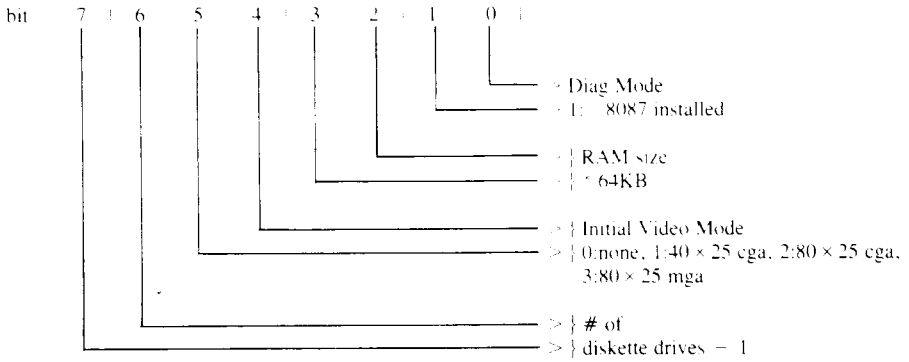
INPUT: DS = ROM data segment

OUTPUT: AH = NUMBER OF SCREEN COLUMNS
AL = CURRENT VIDEO MODE
BH = ACTIVE DISPLAY PAGE

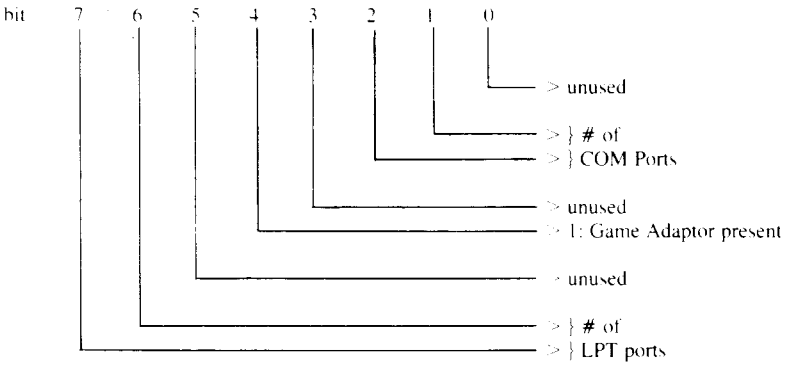
EQUIPMENT CHECK VIA S/W INT 11H

OUTPUT: AX = Equipment Flags

Bits of AL:



Bits of AH:



MEMORY SIZE CHECK VIA S/W INT 12H

OUTPUT: AX = Total Memory size in Kilobytes

DISKDSR ENTRY POINT VIA S/W INT 13H

RESET DISK SUBSYSTEM (AH = 00H)

OUTPUT: AH = DISK STATUS

READ DISK STATUS (AH = 01H)

OUTPUT: AH & AL = DISK STATUS

READ SECTOR(S) (AH = 02H)
WRITE SECTOR(S) (AH = 03H)
VERIFY SECTOR(S) (AH = 04H)

INPUT: DL = DRIVE NUMBER (0-3)
DH = HEAD NUMBER (0-1)
CH = TRACK NUMBER (0-39)
CL = SECTOR NUMBER (1-8)
AL = NUMBER OF SECTORS TO READ, WRITE OR VERIFY (1-8)
ES:BX = BUFFER ADDRESS

OUTPUT: AH = DISK STATUS
AL = 0

FORMAT TRACK (AH = 05H)

INPUT: DL = DRIVE NUMBER (0-3)
DH = HEAD NUMBER (0-1)
CH = TRACK NUMBER (0-39)
AL = # of sectors to format to see if we have a DMA boundary error
ES:BX = BUFFER ADDRESS 4-BYTE TRACK INFO FIELDS (C,H,R,N):
C = TRACK NUMBER
H = HEAD NUMBER
R = SECTOR NUMBER
N = BYTES/SECTOR (00 = 128, 01 = 256, 10 = 512, 11 = 1024)

OUTPUT: AH = DISK STATUS

DISK STATUS RETURNED IN AH (IF CF = 1)

01H – Illegal Command
02H – Address Mark not Found
03H – Write Protect Error
04H – Sector not found
06H – No Diskette
08H – DMA Overrun
09H – DMA Boundary Violation
10H – CRC Error
20H – FDC Error
40H – Seek Error
80H – Timeout

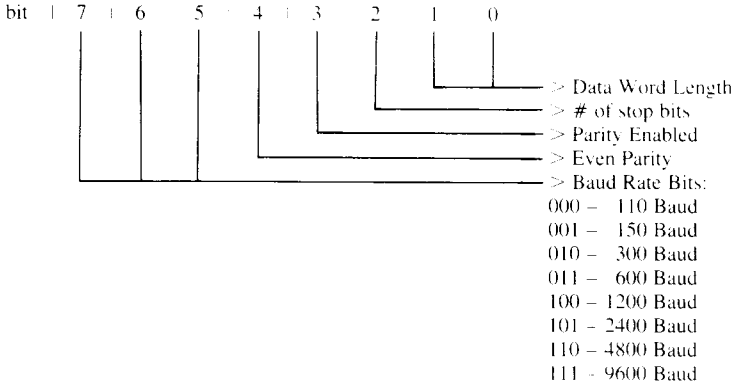
EIA DSR ENTRY POINT VIA S/W INT 14H

INITIALIZE COMM PORT (AH = 00H)

INPUT: DX = Modem Control Register port
AL = Baud Rate and UART control parameters
BH = 0, upper bits of baud rate index

OUTPUT: AH = Line Status
AL = Modem Status
Serial Port Control bits in AL Register

Bits of AL on Entry:



TRANSMIT A CHAR (AH = 01H)

INPUT: DX = Index into device table
AL = Character to transmit
CX = Timeout value
BX = 0, used as timeout counter

OUTPUT: AH = Line Status

RECEIVE A CHAR (AH = 02H)

INPUT: DX = Index into device table
CX = Timeout value
BX = 0, used as timeout counter

OUTPUT: AH = Line Status (error bits only, = 0 if OK)
AL = Received Character

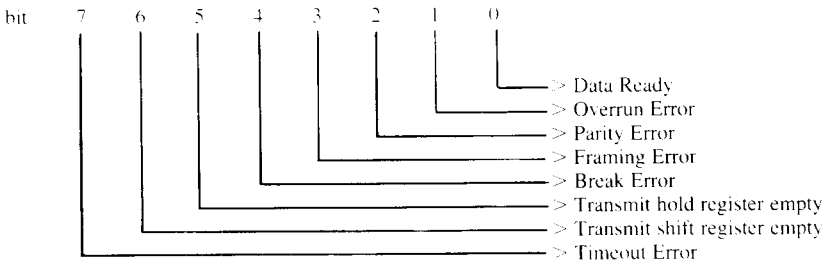
RETURN SERIAL PORT STATUS (AH = 03H)

INPUT: DX = Modem Control Register port

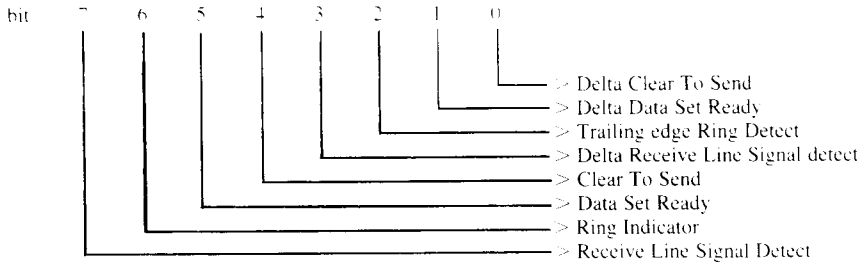
OUTPUT: AH = Line Status
AL = Modem Status

Serial Port Status bits returned in AX Register:

AH Register:



AH Register:



KYBDSR ENTRY POINT VIA S/W INT 16H

READ KEYBOARD INPUT (AH = 00H)

INPUT: DS = ROM data segment (0040h)

OUTPUT: AL = ASCII CHARACTER
AH = SCAN CODE

READ KEYBOARD STATUS (AH = 01H)

INPUT: DS = ROM data segment (0040h)

OUTPUT: AL = ASCII CHARACTER
AH = SCAN CODE
Z FLAG = 1 if no character available
Z FLAG = 0 if character available

READ SHIFT STATUS (AH = 02H)

INPUT: DS = ROM data segment (0040h)

OUTPUT: AL = SHIFT STATUS BYTE

CASSETTE INT 15H DSR

OUTPUT: AH = 86h, Error code, Carry set.
Interrupts off

KYBDSR ENTRY POINT VIA S/W INT 16H

READ KEYBOARD INPUT (AH = 00H)

INPUT: DS = ROM data segment (0040h)

OUTPUT: AL = ASCII CHARACTER
AH = SCAN CODE

READ KEYBOARD STATUS (AH = 01H)

INPUT: DS = ROM data segment (0040h)

OUTPUT: AL = ASCII CHARACTER
AH = SCAN CODE
Z FLAG = 1 if no character available
Z FLAG = 0 if character available

READ SHIFT STATUS (AH = 02H)

INPUT: DS = ROM data segment (0040h)

OUTPUT: AL = SHIFT STATUS BYTE

PRINTER ENTRY POINT VIA S/W INT 17H

PRINT CHARACTER (AH = 00H)

INPUT: AL = Character to output
DX = Index to Printer table port + 1
(the Status port)
CX = Timeout value

OUTPUT: AH = Printer Status

INITIALIZE PRINTER (AH = 01H)

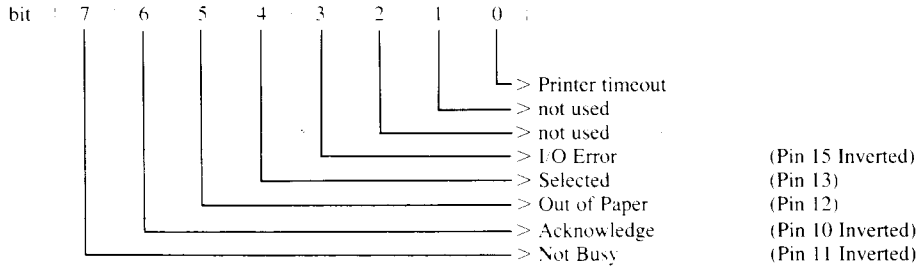
INPUT: DX = Printer Status Port address = (Printer Table contents + 1)

OUTPUT: AH = Printer Status

RETURN PRINTER STATUS (AH = 02H)

INPUT: DX = Index to Printer table port + 1 (the Status port)
CH must have correct value of timeout flag

OUTPUT: AH = Printer Status:



Notes: Pins #s are those on a 25 pin D connector

ROM BASIC ENTRY VIA S/W INT 18H

Not able to boot diskette, go to Monitor, error message or a ROM BASIC

BOOT FROM DISKETTE VIA S/W INT 19H

*** B O O T D I S K E T T E ***

If boot attempt fails:

Fall through to user routine INT 18h, which might be a monitor or a ROM BASIC, an error message etc.
Should INT 18h return, which is unlikely, we'll return to caller of INT 19h.

TIMER DEVICE SERVICE ROUTINE — INT 1Ah

READ CLOCK (AH = 00h)

INPUT: DS = ROM data segment (0040h)

OUTPUT: AL = 24-Hour Rollover flag
CX = High word of Clock Count
DX = Low word of Clock Count

SET CLOCK (AH = 01h)

INPUT: DS = ROM data segment (0040h)
CX = High word of Clock Count
DX = Low word of Clock Count

OUTPUT: AH = 0

USER SUPPLIED KEYBOARD BREAK ROUTINE — INT 1Bh

When a CTRL + ScrLock is detected, this INT is issued. A USER Break Routine may be invoked here. Note that this function is used by MSDOS.

USER SUPPLIED TIMER INTERVAL TICK — INT 1Ch

This interrupt is called internally after each timer interrupt (18.2 Hz). It is initialized to point to a dummy 1RET instruction.

CRT CONTROLLER PARAMETERS — DWORD POINTER AT INT IDH VECTOR

Default Video CRT Parameter Block

DEFAULT DISK PARAMETER BLOCK — DWORD POINTER AT INT 1EH VECTOR

Default Diskette Parameter Block

EXTENDED GRAPHIC CHARACTER SET — DWORD POINTER AT INT 1FH VECTOR

Character Generator ROM used in Graphic mode for characters 80H to OFFh.

Janus.Library

PREFACE

This is a brief description of the janus code. This code supports low level access to the "janus" system — the link between a PC and an Amiga.

THE PUBLIC ROUTINES

Contents

- AllocJanusMem
- CheckJanusInt
- CleanupJanusSig
- FreeJanusMem
- GetJanusStart
- GetParamOffset
- JBCopy
- JanusLock
- JanusMemBase
- JanusMemToOffset
- JanusMemType
- JanusUnLock
- SendJanusInt
- SetJanusEnable
- SetJanusHandler
- SetJanusRequest
- SetParamOffset
- SetupJanusSig

Descriptions

The code is packaged as a library (specifically "janus.library"), which is loaded during Autoconfig procedure.

All routines that return a value return it in D0. There is a link library for C routines, "jlib.lib".

oldHandler = SetJanusHandler(jintnum, intserver)
D0 A1

This routine sets up an interrupt handler for a particular janus interrupt. The old interrupt is returned. A null means that there is no interrupt handler. If there is no interrupt handler then interrupts not will be processed for that jintnum.

oldEnable = SetJanusEnable(jintnum, newvalue)
D0 D1

Each jintnum may be individually enabled or disabled (this is in addition to the control of setting the interrupt handler to NULL). If the interrupt is disabled then requests that are received will not generate interrupts. These requests may be detected via SetJanusRequest.

If newvalue is 0 then the interrupt is disabled. If it is 1 then the interrupt is enabled. All other values are reserved.

This routine will generate an interrupt if it an interrupt is enabled that has a pending request. This does not currently happen until the next hardware interrupt occurs.

oldRequest = SetJanusRequest(jintnum, newvalue)
D0 D1

This routine sets or clears an interrupt request for jintnum. If newvalue is zero then the request is cleared. If newvalue is one then the request is set. In either case the old value of the request is returned.

Setting a request will generate an interrupt (if it is enabled). This does not currently happen.

SendJanusInt(jintnum)
D0

This call is useful for "system" requests — e.g. those requests not directly defined by the hardware. The call marks the request in the system interrupt area and then posts a hardware interrupt to the pc.

CheckJanusInt(jintnum)
D0

This call returns the status byte from the interrupt area. It can be used to tell if the pc has noticed the interrupt yet. A value of JNOINT ($ff) means no interrupt is pending (which probably means that the pc has already processed it). JSENDINT ($7f) means that the interrupt is pending. Anything else should be treated with suspicion.

ptr = AllocJanusMem(size, type)
D0 D1

This routine allocates memory from the parameter or buffer memory free pools, and returns a 68000 addressable pointer to the memory. It allocates "size" bytes, or returns NULL if there was not enough memory.

The type field is used to determine which pool of memory is desired. It should be either MEMF_PARAMETER or MEMF_BUFFER. In addition, you may specify what sort of memory access the pointer should refer to. The four choices are MEM_BYTEACCESS, MEM_WORD-

ACCESS, MEM_GRAPHICACCESS, or MEM_IOACCESS. See the hardware description for the meaning of these access methods if you do not already know.

FreeJanusMem(ptr, size)
A1 D0

The specified memory is returned to the free pool. Some modest error checking is done, and the system will Alert if there is a problem.

ptr = JanusMemBase(type)
D0

The base of the memory referred to by the type specifier is returned. See AllocJanusMem for a (very) brief description of type.

type = JanusMemType(ptr)
D0

The type of the pointer is returned. "Unpredictable results" will occur if ptr points to neither buffer memory nor parameter memory.

offset = JanusMemToOffset(ptr)
D0

If ptr points to buffer or parameter memory, the offset from the start of that memory to ptr will be returned. This is the value that should be fed to SetParamOffset( ) if this is a parameter block.

offset = GetParamOffset(jintnum)
D0

The parameter offset for interrupt jintnum is returned. The system does not interpret this number, but by convention a $ffff means that no parameter block has been set up.

oldOffset = SetParamOffset(jintnum, offset)
D0 D1

The parameter offset for jintnum is set to the bottom sixteen bits of offset. The previous offset is returned. The system does not interpret this number, but by convention a $ffff means that no parameter block had previously been set up.

ptr = GetJanusStart( )

The base of the janus board is returned.

setupSig = SetupJanusSig
(jintnum, signum, paramsize, paramtype)
D0 D1 D2 D3

This routine does the "standard" things that most users of the janus system would want. It is conceivable that most people who use the janus board will use only this routine and CleanupJanusSig( ).

The main purpose is to set up an interrupt handler for your interrupt, and translate this into an exec signal that will be sent to your task. This allows you to ignore all the complexities of writing interrupt code.

You specify the jintnum to use as the interrupt number and the signal number (signum) to be sent to you. Jintnum should (for now) be gotten via the include file services.[hi]. Signum will most often be gotten via AllocSignal(-1), which allocates an unused signal.

In addition to setting up a way to get interrupts, the call can set up a parameter area. It will allocate paramsize bytes of type paramtype, and set up the parameter area to point to them.

There is some error checking done while all this is going on. If signum is -1 the call fails (-1 is the error return from AllocSignal...). If there is already an interrupt handler then the call fails. If paramsize is non-zero and there is already a parameter area the call fails. If it cannot allocate enough memory the call fails.

The call returns a NULL if it fails. If it succeeds then a pointer to a SetupSig structure is returned. This structure is defined in setupsig.[hi].

CleanupJanusSig(setsupSig)
A0

This routine undoes everything that SetupJanusSig does.

JanusLock(ptr)
A0

Gain a janus lock (e.g. a lock on a memory list). You must not keep this lock for a long time — keep it just long enough to manipulate the data structure associated with the lock, and don't go to sleep.

JanusUnLock(ptr)
A0

Release a janus lock.

JBCopy(source, designation, length)
A0 A1 D0

Copy arbitratily aligned memory as efficiently as possible with the processor.

INCLUDE FILES

janus.[hi]:
gives interface to janus.library. All definitions in this file are amiga specific. The most useful thing in this file are the definitions for janus memory allocation types

janusreg.[hi]:
hardware constants. Most people should not need this. If you do, we need to hide more information.

janusvar.[hi]:
the shared data structure between the amiga and the pc. Once again, you should not need direct access to these routines. We have tried to provide interface routines to do all the normal things.

i86block.i:
command blocks for calling pc's interrupt's directly and for the hard disk.

services.[hi]:
hard coded constants for interrupt numbers. Eventually these numbers will be gotten at run time, but for now they are constants. These numbers correspond to the "jintnum" parameters below.

setupsig.[hi]:
data structure for SetupJanusSig( ) call.

LISTINGS

```
i86block.i — interface definitions between amiga and
             commodore-pc
Copyright © 1986, Commodore-Amiga Inc., All rights reserved
```

```
        IFND                    JANUS_I86BLOCK_I
JANUS_I86BLOCK_I                SET 1

; All registers in this section are arranged to be read and written
; from the WordAccessOffset area of the shared memory. If you really
; need to use the ByteAccessArea, all the words will need to be byte
; swapped.
```

```
; Syscall86 — how the 8086/8088 wants its parameter block arranged:

STRUCTURE Syscall86,0
    UWORD   s86_AX
    UWORD   s86_BX
    UWORD   s86_CX
    UWORD   s86_DX
    UWORD   s86_SI
    UWORD   s86_DS
    UWORD   s86_DI
    UWORD   s86_ES
    UWORD   s86_BP
    UWORD   s86_PSW
    UWORD   s86_INT             ; 8086 int # that will be
                                  called
    LABEL   Syscall86_SIZEOF

; Syscall68 — the way the 68000 wants its parameters arranged:

STRUCTURE Syscall68,0
    ULONG   s68_D0
    ULONG   s68_D1
    ULONG   s68_D2
    ULONG   s68_D3
    ULONG   s68_D4
    ULONG   s68_D5
    ULONG   s68_D6
    ULONG   s68_D7
    ULONG   s68_A0
    ULONG   s68_A1
    ULONG   s68_A2
    ULONG   s68_A3
    ULONG   s68_A4
    ULONG   s68_A5
    ULONG   s68_A6
    ULONG   s68_PC              ; pc to start execution from
    ULONG   s68_ArgStack        ; array to be pushed onto
                                  stack
    ULONG   s68_ArgLength       ; number of bytes to be
                                  pushed (must be even)
    ULONG   s68_MinStack        ; minimum necessary stack
                                  (0 = use default)
    ULONG   s68_CCR             ; condition code register
    ULONG   s68_Process         ; ptr to process for this
                                  block.
    UWORD   s68_Command         ; special commands: see
                                  below
    UWORD   s68_Status          ;
    UWORD   s68_SigNum          ; internal use: signal to
                                  wake up process
```

```
LABEL       Syscall68_SIZEOF

S68COM_DOCALL       EQU 0       ; normal case _ jsr to specified Program cntr
S68COM_REMPROC      EQU 1       ; kill process
S68COM_CRPROC       EQU 2       ; create the process, but do not call anything

; Disk request structure for raw amiga access to 8086's disk
; goes directly to PC BIOS (via PC int 13 scheduler):

STRUCTURE DskAbsReq,0
  UWORD   dar_FktCode           ; bios function code (see ibm tech ref)
  UWORD   dar_Count             ; sector count
  UWORD   dar_Track             ; cylinder #
  UWORD   dar_Sector            ; sector #
  UWORD   dar_Drive             ; drive
  UWORD   dar_Head              ; head
  UWORD   dar_Offset            ; offset of buffer in MEMF_BUFFER memory
  UWORD   dar_Status            ; return status
  LABEL   DskAbsReq_SIZEOF

; Definition of an AMIGA disk partition. returned by info function:

STRUCTURE DskPartition,0
  UWORD   dp_Next               ; 8088 ptr to next part. 0 -> end of list
  UWORD   dp_BaseCyl            ; cyl # where partition starts
  UWORD   dp_EndCyl             ; last cyclinder # of this partition
  UWORD   dp_DrvNum             ; DOS drive number (80H, 81H, ...)
  UWORD   dp_NumHeads           ; number of heads for this drive
  UWORD   dp_NumSecs            ; number of sectors per track for this drive
  LABEL   DskPartition_SIZEOF

; Disk request structure for higher level Amiga disk request to 8086:

STRUCTURE AmigaDskReq,0
  UWORD   adr_Fnctn             ; function code (see below)
  UWORD   adr_Part              ; partition number (0 is first partition)
  ULONG   adr_Offset            ; byte offset into partition
  ULONG   adr_Count             ; number of bytes to transfer
```

```
    UWORD   adr_BufferAddr          ; offset into MEMF-
                                      _BUFFER memory for
                                      buffer
    UWORD   adr_Err                 ; return code, 0 if all OK
LABEL       AmigaDskReq_SIZEOF

; Function codes for AmigaDskReq adr_Fnctn word:

ADR_FNCTN_INIT      EQU     0   ; given nothings, sets adr_
                                  Part to # partitions
ADR_FNCTN_READ      EQU     1   ; given partition, offset,
                                  count, buffer
ADR_FNCTN_WRITE     EQU     2   ; given partition, offset,
                                  count, buffer
ADR_FNCTN_SEEK      EQU     3   ; given partition, offset
ADR_FNCTN_INFO      EQU     4   ; given part, buff adr, cnt,
                                  copys in a DskPartition
                                  structure, cnt set to actual
                                  number of bytes copied.

; Error codes for adr_Err, returned in low byte:

ADR_ERR_OK          EQU     0   ; no error
ADR_ERR_OFFSET      EQU     1   ; offset not on sector
                                  boundary
ADR_ERR_COUNT       EQU     2   ; dsk_count not a multiple
                                  of sector size
ADR_ERR_PART        EQU     3   ; partition does not exist
ADR_ERR_FNCT        EQU     4   ; illegal function code
ADR_ERR_EOF         EQU     5   ; offset past end of
                                  partition
ADR_ERR_MULPL       EQU     6   ; multiple calls while
                                  pending service

; Error condition from IBM-PC BIOS, returned in high byte:

ADR_ERR_SENSE_FAIL          EQU $ff
ADR_ERR_UNDEF_ERR           EQU $bb
ADR_ERR_TIME_OUT            EQU $80
ADR_ERR_BAD_SEEK            EQU $40
ADR_ERR_BAD_CNTRLR          EQU $20
ADR_ERR_DATA_CORRECTED      EQU $11 ; data corrected
ADR_ERR_BAD_ECC             EQU $10
ADR_ERR_BAD_TRACK           EQU $0b
ADR_ERR_DMA_BOUNDARY        EQU $09
ADR_ERR_INIT_FAIL           EQU $07
ADR_ERR_BAD_RESET           EQU $05
ADR_ERR_RECRD_NOT_FOUND     EQU $04
ADR_ERR_BAD_ADDR_MARK       EQU $02
ADR_ERR_BAD_CMD             EQU $01

ENDC    !JANUS_I86BLOCK_I
```

```
janus.i — software conventions for janus.i
Copyright © 1986, Commodore-Amiga Inc., All rights reserved
```

```
IFND    EXEC_TYPES_I
INCLUDE "exec/types.i"
ENDC    EXEC_TYPES_I

IFND    EXEC_LIBRARIES_I
        INCLUDE "exec/libraries.i"
ENDC    EXEC_LIBRARIES_I

IFND    EXEC_INTERRUPTS_I
INCLUDE "exec/interrupts.i"
ENDC    EXEC_INTERRUPTS_I

; JanusResource — an entity which keeps track of the reset state of
  the 8088. If this resource does not exist, it is assumed the 8088 can
  be reset.

STRUCTURE JanusResource,LN_SIZE
    APTR    jr_BoardAddress         ; address of JANUS board
    UBYTE   jr_Reset                ; non_zero indicates 8088
                                      is held reset
    LABEL   JanusResource_SIZEOF

; As a coding convenience, we assume a maximum of 32 handlers.
; People should avoid using this in their code, because we want to
; be able to relax this constraint in the future. All the standard
; commands' syntactically support any number of interrupts, but
; the internals are limited to 32.

MAXHANDLER      EQU     32

; JanusAmiga — amiga specific data structures for janus project:

STRUCTURE JanusAmiga,LIB_SIZE
ULONG   ja_IntReq               ; software copy of out-
                                  standing requests
ULONG   ja_IntEna               ; software copy of enabled
                                  interrupts
APTR    ja_ParamMem             ; ptr to (word arranged)
                                  param mem
APTR    ja_IoBase               ; ptr to base of io register
                                  region
APTR    ja_ExpanBase            ; ptr to start of shared
                                  memory
APTR    ja_ExecBase             ; ptr to exec library
APTR    ja_DOSBase              ; ptr to DOS library
APTR    ja_SegList              ; holds a pointer to our
                                  code segment
```

```
APTR        ja_IntHandlers              ; base of array of int server
                                          ptrs
STRUCT      ja_IntServer,IS_SIZE        ; INTB_PORTS server
STRUCT      ja_ReadHandler,IS_SIZE      ; JSERV_READAMIGA
                                          handler
LABEL       JanusAmiga_SIZEOF

; Hide a byte quantity in the lib_pad field
ja_SpuriousMask   EQU   LIB_pad

; Magic constants for memory allocation:

MEM_TYPEMASK        EQU  $00ff  ; 8 memory areas
BITDEF      MEM,PARAMETER,0     ; parameter memory
BITDEF      MEM,BUFFER,1        ; buffer memory
MEM_ACCESSMASK      EQU  $3000  ; bits that participate in
                                  access types
MEM_BYTEACCESS      EQU  $0000  ; return base suitable for byte
                                  access
MEM_WORDACCESS      EQU  $1000  ; return base suitable for
                                  word access
MEM_GRAPHICACCESS EQU  $2000    ; return base suitable for
                                  graphic access
MEM_IOACCESS        EQU  $3000  ; return base suitable for
                                  io access
TYPEACCESSTOADDR    EQU  5      ; # of bits to change access
                                  mask into addr

; Macro to lock access to janus data structures from PC side:
LOCK        MACRO       ; ( 1 — effective address of lock byte )
begin@
            tas         1
            beq.s       exit@
            nop
            nop
            bra.s       begin@
exit@:
            endm

UNLOCK      MACRO       ; ( 1 — effective address of lock byte )
            move.b      #0,1
            ENDM

JANUSNAME   MACRO
            dc.b        'janus.library',0
            ENDM
```

janusreg.i — janus hardware registers (from amiga point of view)
Copyright © 1986, Commodore-Amiga Inc., All rights reserved.

```
; Hardware interrupt bits (all bits are active low)
BITDEF   JINT,MINT,0      ; mono video ram written to
BITDEF   JINT,GINT,1      ; color video ram written to
BITDEF   JINT,CRT1INT,2   ; mono video control registers changed
BITDEF   JINT,CRT2INT,3   ; color video control registers changed
BITDEF   JINT,ENBKB,4     ; keyboard ready for next character
BITDEF   JINT,LPT1INT,5   ; parallel control register
BITDEF   JINT,COM2INT,6   ; serial control register
BITDEF   JINT,SYSINT,7    ; software int request

; The Amiga side of the Bridgeboard has four sections of its address
; space. Three of these parts are different arrangements of the same
; memory. The fourth part has the specific amiga accessible I/O
; registers (jio--??). The other three parts all contain the same
; data, but the data is arranged in different ways: Byte Access
; lets the 68k read byte streams written by the 8088, Word Access
; lets the 68k read word streams written by the 8088, and Graphic
; Access lets the 68k read medium res graphics memory in a more
; efficient manner (the pc uses packed two bit pixels; graphic
; access rearranges these data bits into two bytes, one for each bit
; plane).

ByteAccessOffset        EQU $00000
WordAccessOffset        EQU $20000
GraphicAccessOffset     EQU $40000
IoAccessOffset          EQU $60000

; Within each bank of memory are several sub regions. These are the
; definitions for the sub regions:

BufferOffset            EQU $00000
ColorOffset             EQU $10000
ParameterOffset         EQU $18000
MonoVideoOffset         EQU $1c000
IoRegOffset             EQU $1e000

BufferSize              EQU $10000
ParameterSize           EQU $04000

These are the definitions for the io registers. All the registers are
byte wide and the address are for Byte Access addresses:

jio-KeyboardData        EQU  $061f ; data that keyboard will read
jio_SystemStatus        EQU  $003f ; pc only register

jio_NmiEnable           EQU $005f  ; pc only register

jio_Com2XmitData        EQU $007d
jio_Com2ReceiveData     EQU $009d
jio_Com2IntEnableW      EQU $00bd
```

```
jio_Com2IntEnableR    EQU $00dd
jio_Com2DivisorLSB    EQU $007f
jio_Com2DivisorMSB    EQU $009f
jio_Com2IntID         EQU $00ff
jio_Com2LineCntrl     EQU $011f
jio_Com2ModemCntrl    EQU $013f
jio_Com2LineStatus    EQU $015f
jio_Com2ModemStatus EQU $017f

jio_Lpt1Data          EQU $019f  ; data byte
jio_Lpt1Status        EQU $01bf  ; see equates below
jio_Lpt1Control       EQU $01df  ; see equates below

jio_MonoAddressInd    EQU $01ff  ; current index into crt data regs
jio_MonoData          EQU $02a1  ; every other byte for 16 registers
jio_MonoControlReg    EQU $02ff

jio_ColorAddressInd   EQU $021f  ; current index into crt data regs
jio_ColorData         EQU $02c1  ; every other byte for 16 registers
jio_ColorControlReg   EQU $023f
jio_ColorSelectReg    EQU $025f
jio_ColorStatusReg    EQU $027f

jio_DisplaySystemReg  EQU $029f

jio_IntReq            EQU $1ff1  ; read clears, pc -> amiga ints
jio_PcIntReq          EQU $1ff3  ; r/o, amiga -> pc ints
jio_ReleasePcReset    EQU $1ff5  ; r/o, strobe release pc's reset
jio_RamSize           EQU $1ff7  ; r/o, give ram addresses
jio_IntEna            EQU $1ff9  ; r/w, enables pc int lines
jio_PcIntGen          EQU $1ffb  ; w/o, bit = = 0 -> cause pc int
jio_Control           EQU $1ffd  ; w/o, random control lines
jio_RamBaseAddr       EQU $1fff  ; r/w, sets expansion ram base
                                    address
```

; Now the magic bits in each register (and boy, are there a lot of them!)

```
; Bits for Lpt1Status register
BITDEF    JPCLS,STROBE,0
BITDEF    JPCLS,AUTOFEED,1
BITDEF    JPCLS,INIT,2
BITDEF    JPCLS,SELECTIN,3
BITDEF    JPCLS,IRQENABLE,4   ; active 1

; Bits for Lpt1Control register
BITDEF
JPCLC,ERROR,3
BITDEF
JPCLC,SELECT,4
```

```
BITDEF      JPCLC,NOPAPER,5
BITDEF    JPCLC,ACK,6
BITDEF    JPCLC,BUSY,7

; Bits for PcIntReq, PcIntGen registers
BITDEF    JPCINT,IRQ1,0     ; active high
BITDEF    JPCINT,IRQ3,1     ; active low
BITDEF    JPCINT,IRQ4,2     ; active low
BITDEF    JPCINT,IRQ7,3     ; active low

; PC side interrupts
JPCKEYINT     EQU $ff     ; keycode available
JPCSENDINT    EQU $fc     ; system request
JPCLPT1INT    EQU $f6     ; printer acknowledge

; Bits for RamSize
BITDEF      JRAM,EXISTS,0     ; unset if there is any ram at all
BITDEF      JRAM,2MEG,1       ; set if 2 meg, clear if 1/2 meg

; Bits for control register
BITDEF  JCNTRL,ENABLEINT,0    ; enable amiga interrupts
BITDEF  JCNTRL,DISABLEINT,1   ; disable amiga interrupts
BITDEF  JCNTRL,RESETPC,2      ; reset the pc. remember to strobe
                              ;   ReleasePcReset afterwards
BITDEF  JCNTRL,CLRPCINT,3     ; turn off all amiga->pc ints (except
                              ;   keyboard

; Constants for sizes of various janus regions
JANUSTOTALSIZE    EQU 512*1024    ; 1/2 megabyte
JANUSBANKSIZE     EQU 128*1024    ; 128K per memory bank
JANUSNUMBANKS     EQU 4           ; four memory banks
JANUSBANKMASK     EQU $60000      ; mask bits for bank region
```

janusvar.i—the software data structure for the janus board
Copyright © 1986, Commodore-Amiga Inc., All rights reserved

```
; All bytes described here are described in the byte order of the
; 8088. Note that words and longwords in these structures will be
; accessed from the word access space to preserve the byte order in
; a word — the 8088 will access longwords by reversing the words :
; like a 68000 access to the word access memory

; JanusMemHead — a data structure roughly analogous to an exec
; mem chunk. It is used to keep track of memory used between the
; 8088 and the 68000.
```

```
STRUCTURE JanusMemHead,0
        UBYTE   jmh_Lock            ; lock byte between processors
        UBYTE   jmh_pad0
        APTR    jmh_68000Base       ; rptr's are relative to this
        UWORD   jmh_8088Segment     ; segment base for 8088
        RPTR    jmh_First           ; offset to first free chunk
        RPTR    jmh_Max             ; max allowable index
        UWORD   jmh_Free            ; total number of free bytes -1
        LABEL   JanusMemHead_SIZEOF

STRUCTURE JanusMemChunk,0
        RPTR    jmc_Next    ; rptr to next free chunk
        UWORD   jmc_Size    ; size of chunk -1
        LABEL   JanusMemChunk_SIZEOF

STRUCTURE JanusBase,0
        UBYTE   jb_Lock         ; also used to handshake at 8088 reset
        UBYTE   jb_8088Go       ; unlocked to signal 8088 to initialize
        STRUCT  jb_ParamMem,JanusMemHead_SIZEOF
        STRUCT  jb_BufferMem,JanusMemHead_SIZEOF
        RPTR    jb_Interrupts
        RPTR    jb_Parameters
        UWORD   jb_NumInterrupts
        LABEL   JanusBase_SIZEOF

;------ constant to set to indicate a pending software interrupt
JSETINT         EQU     $7f

        FUNCDEF SetJanusHandler
        FUNCDEF SetJanusEnable
        FUNCDEF SetJanusRequest
        FUNCDEF SendJanusInt
        FUNCDEF CheckJanusInt
        FUNCDEF AllocJanusMem
        FUNCDEF FreeJanusMem
        FUNCDEF JanusMemBase
        FUNCDEF JanusMemType
        FUNCDEF JanusMemToOffset
        FUNCDEF GetParamOffset
        FUNCDEF SetParamOffset
        FUNCDEF GetJanusStart
        FUNCDEF SetupJanusSig
        FUNCDEF CleanupJanusSig
        FUNCDEF JanusLock
        FUNCDEF JanusUnLock
        FUNCDEF JBCopy
```

```
memrw.i—parameter area definition for access to other
        processors mem
Copyright © 1986, Commodore-Amiga Inc., All rights reserved
```

```
    IFND        JANUS_MEMRW_I
JANUS_MEMRW_I   SET 1

; this is the parameter block for the JSERV_READPC and JSERV_
; READAMIGA; services — read and/or write the other processors
; memory.

STRUCTURE MemReadWrite,0
UWORD  mrw_Command      ; see below for list of commands
UWORD  mrw_Count        ; number of bytes to transfer
ULONG  mrw_Address      ; local address to access. This is
                        ;   a machine pointer for the 68000, and
                        ;   a segment/offset pair for the 808x.
                        ;   The address is arranged so the native
                        ;   processor may read it directly.
UWORD  mrw_Buffer       ; The offset in buffer memory for the
                        ;   other buffer.
UWORD  mrw_Status       ; See below for status.
LABEL  MemReadWrite_SIZEOF

; Command definitions:
MRWC_NOP        EQU 0 ; do nothing — return OK status
                      code
MRWC_READ       EQU 1 ; xfer from address to buffer
MRWC_WRITE      EQU 2 ; xfer from buffer to address
MRWC_READIO     EQU 3 ; only on 808x — read from IO
                      space
MRWC_WRITEIO    EQU 4 ; only on 808x — write to IO space
MRWC_WRITEREAD  EQU 5 ; write from buffer, then read back

; Status definitions:
MRWS_INPROGRESS  EQU $ffff   ; we've noticed command and
                             are working on it
MRWS_OK          EQU $0000   ; completed OK
MRWS_ACCESSERR   EQU $0001   ; some sort of protection
                             violation
MRWS_BADCMD      EQU $0002   ; command that the server
                             doesn't understand

        ENDC

    IFND        JANUS_SERVICES_I
JANUS_SERVICES_I    EQU     1
```

```
memrw.i—parameter area definition for access to other
        processors mem
Copyright © 1986, Commodore-Amiga Inc., All rights reserved
```

```
; this is the table of hard coded services. Other services may exist
; that are dynamically allocated via AllocJanusService.

; Service numbers constrained by hardware:
JSERV_MINT          EQU 0   ; monochrome display written to
JSERV_GINT          EQU 1   ; color display written to
JSERV_CRT1INT       EQU 2   ; mono display's control registers
                              changed
JSERV_CRT2INT       EQU 3   ; color display's control registers
                              changed
JSERV_ENBKB         EQU 4   ; keyboard ready for next character
JSERV_LPT1INT       EQU 5   ; parallel control register
JSERV_COM2INT       EQU 6   ; serial control register
; hard coded service numbers
JSERV_PCBOOTED      EQU 7   ; PC is ready to service soft
                              interrupts

JSERV_SCROLL        EQU 8   ; PC is scrolling its screen
JSERV_HARDDISK      EQU 9   ; Amiga reading PC hard disk
JSERV_READAMIGA     EQU 10  ; PC reading Amiga mem
JSERV_READPC        EQU 11  ; Amiga reading PC mem
JSERV_AMIGACALL     EQU 12  ; PC executing Amiga subroutine
JSERV_PCCALL        EQU 13  ; Amiga causing PC interrupt
JSERV_NEWASERV      EQU 14  ; PC initiating Amiga side of a new
                              service
JSERV_NEWPCSERV EQU 15  ; Amiga initiating PC side of a new
                              service

                ENDC    JANUS_SERVICES_I

        IFND    JANUS_SETUPSIG_I
JANUS_SETUPSIG_I    EQU 1
```

```
setupsig.i—data structure for SetupJanusSig() routine
Copyright © 1986, Commodore-Amiga Inc., All rights reserved
```

```
                IFND        EXEC_TYPES_I
                INCLUDE     'exec/types.i'
                ENDC

                IFND        EXEC_INTERRUPTS_I
                INCLUDE     'exec/interrupts.i'
                ENDC
```

```
STRUCTURE SetupSig,IS_SIZE
        APTR        ss_TaskPtr
        ULONG       ss_SigMask
        APTR        ss_ParamPtr
        ULONG       ss_ParamSize
        UWORD       ss_JanusIntNum
        LABEL       SetupSig_SIZEOF

            ENDC
```

```
janus.h—software conventions for janus subsystem
Copyright © 1986, Commodore-Amiga Inc., All rights reserved
```

```
#ifndef     EXEC_TYPES_I
#include    "exec/types.h"
#endif      EXEC_TYPES_I

#ifndef     EXEC_LIBRARIES_I
#include    "exec/libraries.h"
#endif      EXEC_LIBRARIES_I

#ifndef     EXEC_INTERRUPTS_I
#include    "exec/interrupts.h"
#endif      EXEC_INTERRUPTS_I

/*
** As a coding convenience, we assume a maximum of 32 handlers.
** People should avoid using this in their code, because we want
** to be able to relax this constraint in the future. All the
** standard commands' syntactically support any number of
   interrupts,
** but the internals are limited to 32.
*/

#define MAXHANDLER 32

typedef     UWORD       RPTR;

/* JanusAmiga — amiga specific data structures for janus project */

struct JanusAmiga [
        struct Library ja_LibNode;
        ULONG   ja_IntReq;      /* software copy of outstanding
                                   requests */
        ULONG   ja_IntEna;      /* software copy of enabled
                                   interrupts */
        UBYTE   *ja_ParamMem;   /* ptr to (byte arranged) param
                                   mem */
```

```
        UBYTE   *ja_IoBase;         /* ptr to base of io register region */
        UBYTE   *ja_ExpanBase;      /* ptr to start of shared memory */
        APTR    ja_ExecBase;        /* ptr to exec library */
        APTR    ja_SegList;         /* ptr to loaded code */
        struct Interrupt **ja_IntHandlers;/* base of array of int
        handler ptrs */
        struct Interrupt ja_IntServer; /* INTB_PORTS server */
        struct Interrupt ja_ReadHandler;/* JSERV_READAMIGA
        handler */
];
/* hide a byte field in the lib_pad field */
#define   ja_SpurriousMask        lib_pad

/* magic constants for memory allocation */
#define MEM_TYPEMASK        0x00ff  /* 8 memory areas */
#define MEMB_PARAMETER      (0)     /* parameter memory */
#define MEMB_BUFFER         (1)     /* buffer memory */

#define MEMF_PARAMETER      (1<<0) /* parameter memory */
#define MEMF_BUFFER         (1<<1) /* buffer memory */

#define MEM_ACCESSMASK      0x3000  /* bits that participate in
                                       access types */
#define MEM_BYTEACCESS      0x0000  /* return base suitable for
                                       byte access */
#define MEM_WORDACCESS      0x1000  /* return base suitable for
                                       word access */
#define MEM_GRAPHICACCESS 0x2000  /* return base suitable for
                                       graphic access */
#define MEM_IOACCESS        0x3000  /* return base suitable for
                                       io access */

#define TYPEACCESSTOADDR  5         /* # of bits to turn access
                                       mask to addr */

#define JANUSNAME       "janus.library"
```

```
janusreg.h — janus hardware registers (from amiga point
             of view)
Copyright © 1986, Commodore Amiga Inc., All rights reserved
```

```
/* hardware interrupt bits all bits are active low */
#define JINTB_MINT      (0)     /* mono video ram written to */
#define JINTB_GINT      (1)     /* color video ram written to */
#define JINTB_CRT1INT   (2)     /* mono video control registers
                                   changed */
#define JINTB_CRT2INT   (3)     /* color video control registers
                                   changed */
#define JINTB_ENBKB     (4)     /* keyboard ready for next
                                   character */
```

```
#define JINTB_LPT1INT   (5)     /* parallel control register */
#define JINTB_COM2INT   (6)     /* serial control register */
#define JINTB_SYSINT    (7)     /* software int request */

#define JINTF_MINT      (1<<0)
#define JINTF_GINT      (1<<1)
#define JINTF_CRT1INT   (1<<2)
#define JINTF_CRT2INT   (1<<3)
#define JINTF_ENBKB     (1<<4)
#define JINTF_LPT1INT   (1<<5)
#define JINTF_COM2INT   (1<<6)
#define JINTF_SYSINT    (1<<7)

/*
** The amiga side of the janus board has four sections of its address space.
** Three of these parts are different arrangements of the same memory. The
** fourth part has the specific amiga accessible IO registers (jio—??).
** The other three parts all contain the same data, but the data is arranged
** in different ways: Byte Access lets the 68k read byte streams written
** by the 8088, Word Access lets the 68k read word streams written by the
** 8088, and Graphic Access lets the 68k read medium res graphics memory
** in a more efficient manner (the pc uses packed two bit pixels; graphic
* access rearranges these data bits into two bytes, one for each bit plane).
*/

#define ByteAccessOffset            0x00000
#define WordAccessOffset            0x20000
#define GraphicAccessOffset         0x40000
#define IoAccessOffset              0x60000

#define jio_IntReq          0x1ff1  /* read clears, pc -> amiga ints */
#define jio_PcIntReq        0x1ff3  /* r/o, amiga -> pc ints */
#define jio_ReleasePcReset  0x1ff5  /* r/o, strobe release pc's reset */
#define jio_RamSize         0x1ff7  /* r/o, give ram addresses */
#define jio_IntEna          0x1ff9  /* r/w, enables pc int lines */
#define jio_PcIntGen        0x1ffb  /* w/o, bit == 0 -> cause pc int */
#define jio_Control         0x1ffd  /* w/o, random control lines */
```

```
#define jio_RamBaseAddr   0x1fff      /* r/w, sets extra ram base
                                         address */

/* now the magic bits in each register (and boy, are there a lot of
them!) */

/* bits for PcIntReq, PcIntGen registers */
#define JPCINTB_IRQ1      (0)         /* active high */
#define JPCINTB_IRQ3      (1)         /* active low */
#define JPCINTB_IRQ4      (2)         /* active low */
#define JPCINTB_IRQ7      (3)         /* active low */

#define JPCINTF_IRQ1      (1<<0)
#define JPCINTF_IRQ3      (1<<1)
#define JPCINTF_IRQ4      (1<<2)
#define JPCINTF_IRQ7      (1<<3)

/* pc side interrupts */
#define JPCKEYINT         (0xff)      /* keycode available */
#define JPCSENDINT        (0xfc)      /* system request */
#define JPCLPT1INT        (0xf6)      /* parallel port acknowledge */

/* bits for RamSize */
#define JRAMB_EXISTS      (0)         /* set if there is any ram at
                                         all */
#define JRAMB_2MEG        (1)         /* set if 2 meg, clear if 1/2
                                         meg */

#define JRAMF_EXISTS      (1<<0)
#define JRAMF_2MEG        (1<<1)

/* bits for control register */
#define JCNTRLB_ENABLEINT   (0)       /* enable amiga interrupts */
#define JCNTRLB_DISABLEINT  (1)       /* disable amiga interrupts */
#define JCNTRLB_RESETPC     (2)       /* reset the pc. remember to
                                         strobe */
                                      /* ReleasePcReset afterwards */
#define JCNTRLB_CLRPCINT    (3)       /* turn off all amiga->pc ints */

/* constants for sizes of various janus regions */
#define JANUSTOTALSIZE    (512*1024)  /* 1/2 megabyte */
#define JANUSBANKSIZE     (128*1024)  /* 128K per memory
                                         bank */
#define JANUSNUMBANKS     (4)         /* four memory
                                         banks */
#define JANUSBANKMASK     (0x60000)   /* mask bits for bank
                                         region */
```

```
janus.h—the software data structures for the janus board
Copyright © 1986, Commodore Amiga Inc., All rights reserved
```

```
/* all bytes described here are described in the byte order of the
 * 8088. Note that words and longwords in these structures will be
 * accessed from the word access space to preserve the byte order in
 * a word — the 8088 will access longwords by reversing the words:
 * like a 68000 access to the word access memory.
 */
/* JanusMemHead — a data structure roughly analogous to an exec
  mem chunk.
 * It is used to keep track of memory used between the 8088 and the
  68000.
 */

struct JanusMemHead [
        UBYTE   jmh_Lock;           /* lock byte between
                                       processors */
        UBYTE   jmh_pad0;
        APTR    jmh_68000Base;      /* rptr's are relative to this */
        UWORD   jmh_8088Segment;/* segment base for 8088 */
        RPTR    jmh_First;          /* offset to first free chunk */
        RPTR    jmh_Max;            /* max allowable index */
        UWORD   jmh_Free;           /* total number of free
                                       bytes -1 */
];

/* JanusMemChunk — keep track of individualy freed chunks of
memory.
 * Memory Chunks are longword aligned in this memory.
 */

struct JanusMemChunk [
        RPTR    jmc_Next;           /* rptr to next free chunk */
        UWORD   jmc_Size;           /* size of chunk -1*/
];

#ifdef undef
this stuff is saved for future use, but is not yet thought out
/* JanusList — an RPTR/Exec style list header.
 */
struct JanusList [
        RPTR    jl—Head;
        RPTR    jl—Tail;
        RPTR    jl—TailPred;
        UBYTE   jl—Lock;            /* lock byte between
                                       processors
```

```
        */UBYTE  jl—pad0;
];

/* JanusNode — an RPTR/Exec style node.
 */
struct JanusReqList
        [RPTR    jn—Succ;
        RPTR     jn—Pred;
        RPTR     jn—Name;
        UWORD    jn—ReqIndex;      /* this' index into jb—
                                      CommRegs */
];

#endif undef

/* JanusBase — the master data table for the janus project. It is
   located
 * at the bottom of parameter memory.*/

struct JanusBase [
        UBYTE  jb—Lock;           /* lock byte between
                                     processors */
        UBYTE  jb—8088Go;
        struct   JanusMemHead     /* free mem pool for param
                 jb—ParamMem;       memory
        struct   JanusMemHead     /* free mem pool for buffer
                                     memory */
                 jb—BufferMem;
        RPTR   jb—Interrupts;     /* (UBYTE *) of request
                                     byte-pairs */
        RPTR   jb—Parameters;     /* array of ptrs to parameter
                                     areas */
        UWORD jb—NumInterrupts;   /* number of interrupts &
                                     parameters */];

/* constant to set to indicate a pending software interrupt
*/#define JSETINT      0x7f
```

```
memrw.i—parameter area definition for access to other
        processors mem
Copyright © 1986, Commodore-Amiga Inc., All rights reserved
```

```
#ifndef  JANUS_MEMRW_H
#define JANUS_MEMRW_H

/*
** this is the parameter block for the JSERV_READPC and JSERV_
** READAMIGA services — read and/or write the other processors
   memory.
*/
```

```
struct MemReadWrite [
        UWORD  mrw_Command;/* see below for list of commands */
        UWORD  mrw_Count;     /* number of bytes to transfer */
        ULONG  mrw_Address;   /* local address to access. This is */
                              /*   a machine pointer for the */
                              /*   68000, and a segment/offset */
                              /*   pair for the 808X. The ad- */
                              /*   dressed is arranged so the */
                              /*   native processor may read it */
                              /*   directly. */
        UWORD  mrw_Buffer;    /* The offset in buffer memory for */
                              /*   the other buffer. */
        UWORD  mrw_Status;    /* See below for status. */
];

/* command definitions */
#define MRWC_NOP          0     /* do nothing — return OK
                                   status code */
#define MRWC_READ         1     /* xfer from address to buffer */
#define MRWC_WRITE        2     /* xfer from buffer to address */
#define MRWC_READIO       3     /* only on 808x — read from
                                   IO space */
#define MRWC_WRITEIO      4     /* only on 808x — write to
                                   IO space */
#define MRWC_WRITEREAD 5        /* write from buffer, then read
                                   back */

/* status definitions */
#define MRWS_INPROGRESS$ffff  /* we've noticed cmd and are
                                   working on it */
#define MRWS_OK           $0000/* command completed OK */
#define MRWS_ACCESSERR    $0001/* some sort of protection
                                   violation */
#define MRWS_BADCMD       $0002/* command that the server
                                   doesn't understand */
```

services.h—define common service numbers between ibm-pc and amiga
Copyright © 1986, Commodore-Amiga Inc., All rights reserved

```
#ifndef  JANUS_SERVICES_H
#define JANUS_SERVICES_H

/**/
/* this is the table of hard coded services. Other services may exist
/* that are dynamically allocated.
/**/
```

```
/* service numbers constrained by hardware */
#define JSERV_MINT      0   /* monochrome display written to */
#define JSERV_GINT      1   /* color display written to */
#define JSERV_CRT1INT   2   /* mono display's control registers
                               changed */
#define JSERV_CRT2INT   3   /* color display's control registers
                               changed */
#define JSERV_ENBKB     4   /* keyboard ready for next
                               character */
#define JSERV_LPT1INT   5   /* parallel control register */
#define JSERV_COM2INT   6   /* serial control register */

/* hard coded service numbers */
#define JSERV_PCBOOTED      7   /* PC is ready to service soft
                                   interrupts */
#define JSERV_SCROLL        8   /* PC is scrolling its screen */
#define JSERV_HARDDISK      9   /* Amiga reading PC hard disk */
#define JSERV_READAMIGA     10  /* PC reading Amiga mem */
#define JSERV_READPC        11  /* Amiga reading PC mem */
#define JSERV_AMIGACALL     12  /* PC causing Amiga function call */
#define JSERV_PCCALL        13  /* Amiga causing PC interrupt */
#define JSERV_NEWASERV      14  /* PC initiating Amiga side of a
                                   new service */
#define JSERV_NEWPCSERV     15  /* Amiga initiating PC side of a
                                   new service */

#endif  !JANUS_SERVICES_H
#ifndef JANUS_SETUPSIG_H#define
```

setupsig.i—data structure for SetupJanusSig( ) routine
Copyright © 1986, Commodore-Amiga Inc., All rights reserved

```
#ifndef EXEC_TYPES_H
#include "exec/types.h"
#endif

#ifndef EXEC_INTERRUPTS_H
#include "exec/interrupts.h"
#endif

struct SetupSig [
        struct Interrupt ss_Interrupt;
        APTR     ss_TaskPtr;
        ULONG    ss_SigMask;
        APTR     ss_ParamPtr;
        ULONG    ss_ParamSize;
        UWORD    ss_JanusIntNum;
];
#endif
```

PC JANUS SERVICE

This service is called via INT JANUS.
AH contains a function code

J_GET_SERVICE
Gets a new Service Number
Expects:
nothing
Returns:
AL : New Service Number to use
−1 if no service available (J_NO_SERVICE)

J_GET_BASE
Gets Segments & offset of Janus Memory
Expects:
AL : Janus Service Number
Returns:
ES : Janus Parameter Segment
DI : Janus Parameter Offset (if defined),
else −1
DX : Janus Buffer Segment
AL : Status (J_OK, J_NO_SERVICE)

J_ALLOC_MEM
Allocates Janus Memory
Expects:
AL : Type of memory to allocate
BX : Number of Bytes to allocate
Returns:
BX : Offset of registered memory if success,
AL : Status (J_OK, J_NO_MEMORY)

J_FREE_MEM
Releases Janus Memory
Expects:
AL : Type of memory to free
BX : Offset of Memory to free
Returns:
Crash if offset/type was wrong (J_GOODBYE, later)

J_SET_PARAM
Set the default parameter memory pointer
Expects:
AL : Janus Service Number to support
BX : Default Offset of Param Memory to install
Returns:
AL : Status (J_OK, J_NO_SERVICE)

J_SET_SERVICE
Set an address for a far call for that service
Expects:
AL : Janus Service Number to support
ES:DX: Entry address for FAR call
Returns:
AL : Status (J_OK, J_NO_SERVICE)

J_STOP_SERVICE
Prevents AMIGA from using the far call (see above) for this function and releases this Service Number.
No memory is freed up.
No calls are accepted from either side anymore.
Expects:
AL : Number of Service to stop
Returns:
AL: Status (J_OK, J_NO_SERVICE)

J_CALL_AMIGA
Calls the requested function on AMIGA side.
Does not wait for the call to complete.
If J_SET_SERVICE defined, it is internally called on completion.
Expects:
AL : AMIGA Service to call
BX : New Parameter Memory offset to use, −1 : Use default offset
Returns:
AL: Status (J_PENDING, J_FINISHED, J_NO_SERVICE)

J_WAIT_AMIGA
Waits for a previous issued J_CALL_AMIGA to complete.
This function is used if no J_SET_SERVICE is defined.
Expects:
AL : Service Number to wait for
Returns:
AL : Status (J_FINISHED, J_NO_SERVICE)

J_CHECK_AMIGA
Checks completion status of a pending J_CALL_AMIGA
Expects:
AL : Service Number to check
Returns:
AL : Status (J_PENDING, J_FINISHED, J_NO_SERVICE)

This is the Interrupt we are using:

```
JANUS           equ     Obh
```

These are the function codes we know:

```
J_GET_SERVICE   equ     0
J_GET_BASE      equ     1
J_ALLOC_MEM     equ     2
```

```
J_FREE_MEM      equ     3
J_SET_PARAM     equ     4
J_SET_SERVICE   equ     5
J_STOP_SERVICE  equ     6
J_CALL_AMIGA    equ     7
J_WAIT_AMIGA    equ     8
J_CHECK_AMIGA   equ     9
Status Returns:
J_NO_SERVICE    equ     0ffh    ; no service available
J_PENDING       equ     0       ; after J_CALL_AMIGA and
                                  J_CHECK_AMIGA
J_FINISHED      equ     1       ; after J_CALL_AMIGA and
                                  J_CHECK_AMIGA
J_OK            equ     0       ; general good return
J_NO_MEMORY     equ     3       ; requested memory not available
J_ILL_FNCTN     equ     4       ; Illegal function code used in AH
```

Disk request structure for higher level Amiga file request from 8086:

```
AmigaDskReq     STRUC
adr_Fnctn       DW      ?       function code (see below)
adr_File        DW      ?       file number
adr_Offset_h    DW      ?       byte offset into file high
adr_Offset_l    DW      ?       byte offset into file low
adr_Count_h     DW      ?       number of bytes to transfer high
adr_Count_l     DW      ?       number of bytes to transfer low
adr_BufferAddr  DW      ?       offset into MEMF_BUFFER memory
                                for buffer
adr_Err         DW      ?       return code, 0 if all OK
AmigaDskReq     ENDS
```

Function codes for AmigaDskReq adr_Fnctn word

```
ADR_FNCTN_INIT          EQU     0       currently not used
ADR_FNCTN_READ          EQU     1       given file, offset, count, buffer
ADR_FNCTN_WRITE         EQU     2       given file, offset, count, buffer
ADR_FNCTN_SEEK          EQU     3       given file, offset
ADR_FNCTN_INFO          EQU     4       currently not used
ADR_FNCTN_OPEN_OLD      EQU     5       given ASCIIZ pathname in buffer
ADR_FNCTN_OPEN_NEW      EQU     6       given ASCIIZ pathname in buffer
ADR_FNCTN_CLOSE         EQU     7       given file
ADR_FNCTN_DELETE        EQU     8       given ASCIIZ pathname in buffer
```

Error codes for adr_Err, returned in low byte

```
ADR_ERR_OK              EQU     0       no error
ADR_ERR_OFFSET          EQU     1       not used
ADR_ERR_COUNT           EQU     2       not used
ADR_ERR_FILE            EQU     3       file does not exist
ADR_ERR_FNCT            EQU     4       illegal function code
ADR_ERR_EOF             EQU     5       offset past end of file
ADR_ERR_MULPL           EQU     6       not used
ADR_ERR_FILE_COUNT      EQU     7       too many open files
```

```
ADR_ERR_SEEK      EQU   8   seek error
ADR_ERR_READ      EQU   9   read went wrong
ADR_ERR_WRITE     EQU  10   write error
ADR_ERR_LOCKED    EQU  11   file is locked
```

Amiga Hard Disk/SCSI Controller

DESCRIPTION

The Amiga Hard Disk/SCSI Controller is an intelligent high performance controller designed to interface both ST506 hard disk drives and SCSI devices to the Amiga expansion bus architecture. A background command processor provides high level command interpretation minimizing Host intervention. Data is transferred to and from the Host via DMA (direct memory access) with FIFO allowing high data throughput while maintaining reasonable bus bandwidth for other bus controllers.

FEATURES

- Support for up to two ST506 hard disk drives
- Full SCSI with MacIntosh Plus compatibility
- High level command interpretation and exceptional handling performed by Z80 processor
- Support for up to 8 heads, 2048 cylinder with 512 bytes/sector
- Individually Programmable Drive Characteristics
- 1:1 sector interleave
- 32 bit ECC for data correction
- Multiple block transfers
- Full auto-config compatibility
- Real time data transfer rates of up to 800ns/byte via DMA

SPECIFICATIONS

Performance
Hard Disk (ST506)

Encoding method:	MFM
Cylinder per head:	Up to 2048
Sectors per track:	Up to 17
Sector length:	512
Heads:	8
Drive Selects:	2
Step Rate:	3.2 us to 6.5 ms
Data Transfer Rate:	5.0 Mbit/sec.
Write Precomp Time:	12 nanosec.
Sector Interleave:	1:1
Sector Interleave Across Heads:	1:2
Ecc Polynomial:	32 bits
Burst Error Correction:	11 bits

SCSI
ANSI X3T9.2 compatible
MacIntosh Plus compatible connector

Host Interface
Amiga expansion bus compatible
Full auto-config compatibility

Power Requirements
+5 Volts ±5%, 3 Amps. Max.

Environmental

Ambient Temperature:	0 - 55 Deg. C.
Relative Humidity:	20% - 80%

CONNECTOR PIN ASSIGNMENTS

The following tables list the pin assignments for the controller board.

Table 5-1 — Connectors J1 and J2
Disk Serial Data Pin Assignments

Ground Return	Signal Pin	Signal Name
2	1	Drive Selected
4	3	Reserved
6	5	Write Protected (J1 Only)
8	7	Reserved
10	9	Cartridge Changed (J1 Only)
12	11	Ground (GND)
	13	MFM Write Data +
	14	MFM Write Data-
16	15	Ground (GND)
	17	MFM Read Data +
	18	MFM Read Data-
20	19	Ground (GND)

Table 5-2 Connector J0
Disk Control Signal Pin Assignments

Ground Return	Signal Pin	Signal Name
1	2	Head Select 3
3	4	Head Select 2
5	6	Write Gate
7	8	Seek Complete
9	10	Track 00
11	12	Write Fault
13	14	Head Select 2
15	16	Reserved
17	18	Head Select 1
19	20	Index
21	22	Ready
23	24	Step
25	26	Drive Select 1
27	28	Drive Select 2
29	30	Reserved
31	32	Reserved
33	34	Direction In

Table 5-3
Connector CN1, SCSI

SCSI Connector (DB-25) Female

Pin	Name
1	REQ
2	MSG
3	I/O
4	RST
5	ACK
6	BSY
7	GROUND
8	DB0
9	GROUND
10	DB3
11	DB5
12	DB6
13	DB7
14	GROUND
15	C/D
16	GROUND
17	ATN
18	GROUND
19	SEL
20	DBP
21	DB1
22	DB2
23	DB4
24	GROUND
25	N.C.

Reference

- 8727 DMA Specification
- Amiga Expansion Architecture Manual
- Motorola 68000 Technical Manual
- Western Digital WD33C93 SCSI Chip Manual
- American National Standard Committee X3T9.2 SCSI Specification

FUNCTIONAL DESCRIPTION

The Amiga Hard Disk Controller basically consists of three main subsections:

- Host Interface
- ST506 Hard Disk Controller (HDC)
- SCSI Controller

Host Interface

The host interface is 68000 compatible with direct memory access and full auto-config capability. Data transfers to and from the host are usually made via DMA thereby allowing real time date transfer rates of 1.6us/byte for the ST506 interface and up to 800ns/byte for SCSI. Addressing for DMA operations is provided by three external address counters. Before any DMA operation can be performed each counter must be pre-set and thereafter will be incremented automatically. Information on initializing the DMA appears later in this section.

The DMA is a Commodore custom LSI chip (8727) with byte to word funneling and a built in 64 byte FIFO. The internal 64 byte FIFO permits real time data transfer to and from the host without holding the bus for an entire sector transfer. This provides very effective utilization of the bus. The average bus requirement for the transfer of an entire sector is 8.9μs once every 51.2μs. This amounts to only 17% over for CPU and other bus masters.

The interface logic also provides full auto-config and all I/O decode.

For electrical specification and detailed timings refer to Amiga expansion architecture manual.

ST506 Hard Disk Controller (HDC)

The ST506 Hard Disk controller is an intelligent background controller capable of high level command interpretation and support of up two ST506 hard disk units. This controller will be refered to in this document as the HDC or the Hard Disk Controller. The processor for the HDC is a Z80A CPU, with up to 8K of PROM for firmware and 1K of RAM for variable data. Collectively, the above components constitute the "intelligence" of the controller.

The design that has gone into this aspect of the controller has been to enhance performance and increase flexibility while reducing cost.

As a result, the majority of operations have been placed in firmware. The only functions performed by "hardware" are those that are too fast for the processor.

The Z80A CPU and its associated PROM and RAM collectively perform the following functions:

1. Power up initialization
2. Diagnostics
3. Error recovery
4. Error reporting
5. Error correction
6. Command processor
7. Disk select
8. Seek
9. Write precomp select, reduced write current
10. Head select
11. Mapping
12. Logical to physical address translation
    Physical to logical address translation

The DJC Custom Chip

The DJC is a custom LSI chip. It has been designed to handle all serial data, state machine and DMA functions as described below:

ERROR CORRECTION CODE

The error correction polynomial is a 32-bit code capable of correcting up to 11-bit burst errors.

In keeping with the overall design philosophy, the ECC circuitry generates the write syndrome and validates the read without requiring the processor to handle the data. Calculating this polynomial with the processor would seriously degrade the performance of the ST506 controller. Calculating the reverse polynomial to correct bad data is done by the processor. It is accomplished without any measurable effect on performance because the operation is only done after multiple retries and as such is seldom necessary.

HEADER VERIFICATION

Once a disk has been formatted, the DJC converts the desired record address on the disk. The conversion is done in terms of head, track and sector address, with a CRC code tested to further insure positional integrity. A comparison is then made of the header before a read or write function is performed.

TWO INDEX TIMEOUT

This function insures accurate control over the number of attempts to find a header (i.e., it is not "mislead" by counting false address marks).

MFM ENCODE

The DJC converts all parallel data to serial and then to MFM. This function is followed by Precomp, if selected.

Selectable Precomp

In Precomp, a "string" of pulses is analyzed to determine if they are arranged in the unique manner that could cause them to crowd once written on the disk. It also determines which way the crowding would distort the pulses when read. The write pulse stream is then shifted, early or late, to compensate for the crowding conditions, which normally occur on the innermost tracks of the drive.

Under the processor's control, the DJC precomps the disk MFM data by using external inductive delays. Precomp is selectable and is designed to shift the MFM data early or late by 12 nanoseconds to improve read margins.

The use of this feature should be performed in conjunction with the particular drive manufacturer's specification.

MFM Decode

Data received from a disk drive is MFM, a self-clocking serial data stream which contains a phase locked loop, lock detect, missing clock detect and the data seperator.

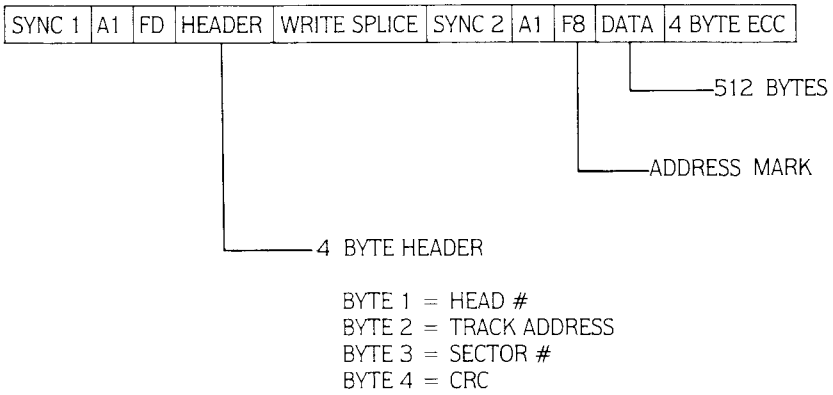
When the DJC asserts Read Gate, the 8465 data seperator will attempt to lock its phase locked loop on the read data. If this does not occur within 4.8 usec, the DJC will turn off Read Gate, causing the 8465 to be placed into the low track rate for increased stability.

The MFM data is now decoded into NRZ data and clock for the DJC. The 8465 decodes a missing clock bit and a hexidecimal A1, FD or an A1, F8 in the sync field. This data indicates the start of a valid header or data field. Receiving any other data causes the DJC to abort the read. Another read would be tried after resyncing the 8465 to 10 MHz.

Sector Format

Figure 5.1 describes the format of a typical sector.

Figure 5.1
Typical Sector Format



Note: 1. Address Mark is a Hex 1 with a missing clock pulse.
2. SYNC field 1 is comprised of 16 bytes of zeros.
3. SYNC field 2 is comprised of 15 bytes of zeros.

Error Recovery Philosophy

Extensive measures have been taken in the design of the controller to insure reliable data. Selectable precompensation circuitry and a sophisticated data seperator with two tracking rates are a few examples. Additional effort has been made to reduce the probability of miscorrection (of having bad data flagged as corrected) through design and options made available to the systems integrator.

In a write operation the controller only precomps the unique combinations of data that might cause crowding conditions on the disk. Shifting data early or late by 12 nsec is done to retain as much of the 50 nsec data window as is possible. This reduces the probability or errors occurring.

In a read operation the data seperator phase lock loop (PLL) provides two tracking rates, a high and a low, which allows for quick synchronization with the header address in the first case and stable data transfer in the second. The controller only contributes a maximum of 6 nsec (typically 3 nsec) of window error out of the allowable error window of 50 nsec. This allows the disk drive to have up to 44 nsec of jitter before error recovery/correction is needed.

The controller uses a 32-bit error correction code that enables an error correction span of up to 11 bits. This computer-generated code is considered superior to fire codes because it substantially reduces the chances of mis- correction while providing the full 11-bit correction span.

In data recovery and error correction the ECC syndrome must be stable in order to perform a correction. This insures that multiple attempts are made to recover marginal data before correction data is applied and further reduces the probability of miscorrection on long (greater than 12-bit) error bursts.

The significance of not correcting data unless the ECC syndrome is stable is that 1) noise induced errors are not corrected and 2) real errors are corrected quickly without wasting time on useless retries.

The user can improve data reliability by mapping tracks with flaws and by reducing the error correction span. The latter reduces the odds of mis-correction on large errors (greater than 12 bits) and provides for early detection of a degrading media. The controller can be programmed to report or not report "soft" errors, on reads that took multiple tries but did not need correction.

Monitoring soft errors is probably the best method of early detection. A correction span of seven (7) bits is thereby suggested as an optimum in data integrity. An alternate eleven (11) bit correction span could be used as a means to retrieve the data before the track is mapped.

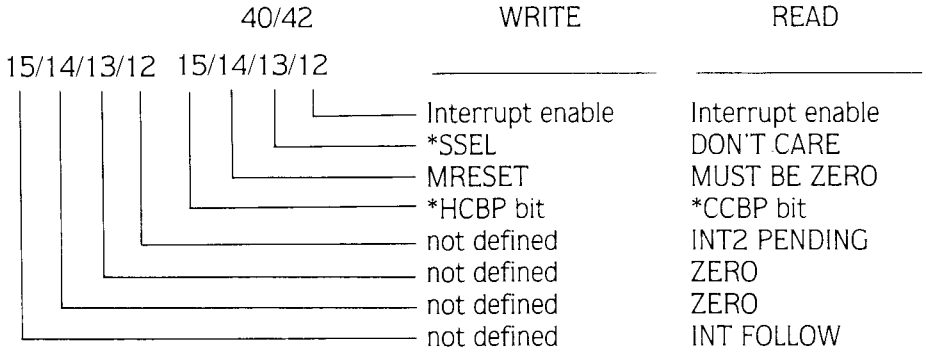
SCSI Controller

The SCSI controller uses the Western Digital WD33C93-SBIC which provides the actual interface to the SCSI connector and supports the full SCSI protocol minimizing host responsibilities. The WD33C93 is supported with a flexible architecture allowing either the 68000 (host) or the Z80A (board processor) to control the WD33C93 operations. Data transfer can be done via DMA or host I/O. For detailed information refer to Western Digital WD33C93 manual.

I/O DEFINITIONS

The following I/O addresses refer only to offset location since the actual board location in physical memory is configurable as described elsewhere in this manual. Refer to this manual for details on autoconfig I/O descriptions. I/O locations 0 hex through 42 hex are written out as nybbles or 4 data bits (AD12-AD15). I/O addresses 50H - 68H are unique to this board and are described later in this document.

Hex Location	Definitions
00/02	Boardtype and size
04/06	Product number
10/12 14/16	Mfg # high and low byte



* Signals unique to Amiga Hard Disk/SCSI Controller.

SSEL	Used to select SCSI controller or to ST605 controller. High = SCSI, low = ST506.
HCBP,CCBP	Host command block pointer and Controller command block pointer. Used to handshake address of Command block pointer to ST506
48H	Base address register

I/O addresses unique to board

50H	PROCC-Interrupt ST506 controller to process command. Write only. Data value written from host is XXX1 hex.
52H	WRCBP/INTACK - Multiplexed signal. WRCBP strobes the command block pointer register. IN-TACK clear INTP at end of command.

SCSI Controller

60H	CS - Chip select for the WD33C93 SCSI chip. Used to write to the internal address register and read from the internal status register.
62H	CS - Chip select for the WD33C93 SCSI chip. Used to write and read remaining Control registers in the WD33C93.
64H	SCSI PCSS - Used to initialize the 8727(DMA) in SCSI mode. Refer to section 5.0 for 8727 commands
68H	SCSI PCSD - Used to pas data to and from the 8727 in SCSI mode. Refer to section 5.0 for transfer procedures.

HOST INTERFACE PROTOCOL

Interface Protocol

The host interface is via a DMA controller. This DMA device is controlled by the Z80A on the disk controller board or 68000 (host). On the host side there are counters for the address bus that are preset before the beginning of each transfer. Three bytes must be written for the 23 address lines (A23-A1). The MSB (corresponding to A24) of the upper address latch is used to control the host R/W- line for DMA transfers. This line is set high to read from the host memory and low if a write is intended. The DMA logic, contained in one chip, can be configured to transfer a single word (2 bytes) or 256 words (512 bytes). Transfer are always on even byte boundaries.

The method of communicating to the DMA circuit is by two control lines PCSS- and PCSD-, controlled by the Z80 or 68000. PCSS- is always strobed first to strobe in the "state" on the data bus. The state will determine the function to be performed on the succeeding PCSD- strobes. Not all valid states need to be followed by a PSCD- strobe and for each state loaded, PCSD- can be strobed any number of times. When reading the host status for instance, the expected number of PCSD- strobes need not be given, but when writing to the DMA controller the correct number of PCSD- strobes must always be given.

DMA Commands

The valid commands, for DMA operations, are summarized in the table below. All data values are listed in hex.

Multiple states can be strobed into he DMA controller as long as no bus contention occurs. Notice that the state bits 4-0 are low in one position only for all the valid states. This implies that any state that does not require transfer of data by the following PCSD- can be combined and set simultaneously. Hence a single word transfer and start DMA cycle can be combined as DE. Some states are mutually exclusive such as F7 (transfer data to or from the FIFO) and EF (reading the DMA status). Similarly state D6 is illegal since word transfer and the FIFO path open will result in BUS contention. State FC is permitted as long as the same data is to be written in the DMA mid address latch and DMA low address counter. Other such valid states can be similarly derived.

Table 5-4. DMA States

Data Strobed by PCSS-	Brief Functional DESCRIPTION	Data Valid PCSD- (R/F)
FB 1111 1011	Load upper DMA address latch	F
FD 1111 1101	Load mid DMA address latch	F
FE 1111 1110	Load low DMA address latch; start DMA on rising edge of LDO; block mode XFER	F
F7 1111 1111	Open path to int. DMA FIFO (64 bytes)	R
EF 1110 1111	Read internal DMA status DB7 = 1 if no DMA or DMA cycle complete DB6 = 1 if byte avail. from or to FIFO DB5 = 1 if no FIFO overflow or underflow	R
9F 1001 1111	Force IREQ- to high impedance	X
BF 1011 1111	Command complete signal to host	X
DF 1101 1111	Set DMA into a single word transfer	X
7F 0111 1111	Reset DMA and clear FIFO followed by FF	X
FF 1111 1111	to ensure proper DMA reset.	X

Load Upper DMA Address Counter (FB)

The LD2 output of the DMA chip is set low on the rising edge of PCSS- and then set high on the falling edge of PCSD-. This loads the R/W- and the upper 7 address lines A23-A17 from the data bus into a counter on the rising edge of LD2. This 8 bit counter need not be reloaded if its contents are to remain unaltered in the succeeding operations.

Load Mid DMA Address Counter (FD)

Address lines A16-A9 are loaded into another counter in the same manner as above by the rising edge of LD1. This 8 bit counter also need not be reloaded if its contents are to remain unaltered in the succeeding operations.

Load Low DMA Address Counter (FE)

On the falling edge of PCSD-, LDO is set high to load the address lines A8-A1. The rising edge of LDO will start the DMA circuit. This also implies a block mode transfer operation, since bits 7-4 are all high. On power-up the DMA controller defaults to the block transfer mode. It should be noted that all three address counters mentioned above are cascaded allowing for the continues transfer of up to 64 Kbytes.

FIFO Access (F7)

This state opens a path to an internal FIFO that is 64 bytes in length. The falling edge of PCSD- will start to shift data out of the FIFO for a read or shift data into the FIFO on the rising edge of PCSD- if the R/W- was set low with LD2. The DMA will initiate host memory access, done a word at a time, whenever the FIFO is half full. A typical memory access without any wait states takes 4 cycles, each cycle being about 140 nS.

Read DMA Status (EF)

The host DMA status must be read before initiating any data transfer, since its FIFO can be shared by another device. At the end of every word or block transfer initiated by the hard disk controller, the status must be read to ensure successful data transfer completion. Status is not read after every word in a block transfer. After the last byte, in a block transfer, has been strobed into the DMA controller approximately 12 uS are needed to ensure that the DMA status lines are all high. To read the status, any number of PCSD- strobes may be used before initiating another DMA cycle. The DMA internal status available after the falling edge of PCSD- is interpreted as follows:

DATA BIT 7: This line will be high if no DMA was requested or a DMA cycle was completed. After completion of a word or a block transfer, this bit will be set high. A low indicates DMA busy status.

DATA BIT 6: This bit is high if a byte of data is available to be read from the FIFO, or if there is a byte to be written and the FIFO is not full. At the end of a block write operation to the disk, since there are no more bytes available, this bit is set low.

DATA BIT 5: This line is low if the FIFO overflowed or underflowed. This may occur during a disk transfer if the DMA circuit does not receive a bus acknowledge signal from another device on the 68000 motherboard, before the FIFO becomes full or empty. Under this condition the FIFO is cleared by the Z80, before any other data transfer can be initiated.

DATA BITS 4-0: These data lines will be logic zero.

Reset IREQ- (9F)

This state will force IREQ- line to high impedance. It is set low by the host.

Command Complete Acknowledge (BF)

This will cause the assertion of the host vectored interrupt line to its active low state to indicate the completion of a command by the HDC.

Word Transfer (DF)

This will set the internal DMA circuit into a single word tranfer. On completion of the word transfer, the DMA resets to a block transfer mode. Hence this state must be strobed for every word transfer desired.

Reset DMA (7F)

This state, followed by state 'FF', resets the DMA circuits and clears the FIFO. This state should be strobed on power-up and to clear any FIFO underflow or overflow conditions.

HOST/HDC COMMAND PROTOCOL

Commands are passed to the HDC through the DMA circuit. When the host requires a disk transfer a command block will be setup in the 68000 memory followed by the host asserting the IREQ- line low. The Z80 will then go through a sequence for each IREQ as discussed below:

Step 1: Setting Up The DMA Address

State FB is loaded into the DMA circuit with PCSS- followed by PCSD- with the hex value of desired high ordered address. Bit 7 of the data bus determines the direction of the transfer, a low will cause a write operation to host and a high will cause a read from host.

Then state FD is loaded into the DMA circuit with PCSS followed by PCSD- with the value of desired address on the data bus. This sets up address lines A16-A9.

State DE is loaded with PCSS- for a word transfer. A value of 06 is loaded with PCSD- to point to the 12th and 13th bytes of the command block. On the falling edge of PCSD- the DMA word cycle will begin. Byte 12 must be FF before the command is executed.

Step 2: Reading Data

The state EF is loaded with PCSS- so that on the falling edge of PCSD- internal DMA status will be outputed. The data lines DATA7, DATA6, and DATA5 are examined until they are high indicating completion of the DMA cycle and that data has shifted through the FIFO. For a block write operation to the disk, DATA6 is examined until low. The HDC will sample the status for about 20 mS, until the data bus

contains E0 or A0, before attempting to clear the FIFO and retransmit the block of data, if necesssary. If the FIFO cannot be cleared after within 20 mS, the command will be terminated in the normal manner, if possible.

Step 3: Reading The Command Block

If byte 12 is an FF, the rest of the command block is retrieved by the CMD. This requires the execution of Step 1 (LD0 only) followed by Step 2 four times. The data value for state DE of Step 1 is incremented from 00 to 03, by the HDC for each word transfer to get all eight command bytes.

Step 4: Data Block Transfer

Block transfers are initiated as in Step 1 except that the third state loaded is FE. The state DE was a single word transfer. The direction of transfer is determined by data line DATA7 when initializing the high order address lines. Status is read by the HDC at the end of every block or word transfer, and at the start of every new command.

Step 5: Command Completion

To complete a command status must be returned to the host. The status information returned is that defined by the 'Request Sense' command. To do this, 2 status words must be transferred to the command block. The host DMA is setup for a word transfer, by setting the LD2, LD1, and the LD0 counters similar to the read of the command block byte 12 (see Step 1). The four status bytes: ERROR CODE, LUN:LADD2, LADD1, and LADD0 are loaded into the FIFO on the rising edge of PCSD-, a word at a time. As usual, the DMA status is examined, between word transfers. If the command, just executed by the HDC required a disk access, then the ADV (address valid) bit is set. Otherwise ADV = 0 to indicate that the LSA, reported in the 4 byte status block, is meaningless. This completes the instruction. The host is acknowledged by writing state BF to set the host vectored interrupt line low. Also IREQ- is deasserted by the HDC.

COMMANDS

Command Block

In the 68000 memory located at an address determined by Amiga DOS is a 16 byte command block. The first byte received through the FIFO is the MSB even byte, followed by the LSB odd byte. During the command block transfer phase, 8 bytes specifying the command are read by the HDC. The command block is organized as follows:

Table 5.5. Host Command Block

BYTE	WORD	7	6	5	4	3	2	1	0
0	0	Command Class			OP Code				
1	0	Logical Unit Number			Logical Sector Address (High)				
2	1	Logical Sector Address (Middle)							
3	1	Logical Sector Address (Low)							
4	2	Block (sector) Count							
5	2	Control Byte (reserved in DMA spec)							
6	3	High Order DMA DB Address (A23-A17)							
7	3	Mid Order DMA DB Address (A15-A9)							
8	4	Low Order DMA DB Address (A1-A8)							
9	4	Reserved							
10	5	Reserved							
11	5	Reserved							
12	6	ADV	Error Type		Error Code				
13	6	LUN			LADD 2				
14	7	LADD1							
15	7	LADD0							

Byte 0 must be specified for all commands. Depending on the value of Byte 0, each parameter in Bytes 1 through 5 may require specification. Table 6.2 specifies the supported commands and their parameters. It also includes information in data transfers required during execution. All other commands are reserved.

Command Class

There are eight command classes. Command class 0 contains the commands used in normal operation. Command class 7 contains the diagnostic commands. Command classes 1, 2, 4, 5, and 6 are reserved for future use.

Operation Code

There are 32 operation codes in each command class. For a description of all the available op codes see the Command Description Section.

Logical Unit Number

This is contained in the upper three bits of Byte 1 specifyfing one of eight logical unit numbers. Logical units 0 and 1 are hard disk drives 0 and 1 respectively. Logical units 2 and 7 are reserved for future use. The HDC reports an invalid command if the logical unit number is out of range. However, for error reporting, all even LUN's are treated as drive 0 and all odd LUN's are treated as drive 1.

Logical Sector Address

A logical sector address is a 21 bit unsigned integer that specifies a unique physical sector. The one-to-one correspondence between the set of logical sector addresses and the set of physical sectors is computed by the HDC from the Cylinder (C), Head (H), and Sector (S) address, as well as the drive parameters, heads per drive (HD) and Sectors per track (ST):

$$L = (((C * HD) * H) * ST) + S$$

C, H and S can be derived from L, HD, and ST as follows:

$$S = L \text{ Modulo } ST$$
$$H = ((L-S)/ST) \text{ Modulo } HD$$
$$C = (((L-S)/ST)-H)/HD$$

This field specifies a sector or the first sector for the Read and Write Drive commands. When only a track specification is required, the sector number implied by the Logical Sector Address is ignored. Hence each format type command begins operation at the beginning of the track containing the specified sector. The HDC will report an invalid command, if the logical address specified is out of range.

Block Count

The sector count is a parameter for each data transfer command. It specifies the number of logical sectors to be transferred during any disk READ or WRITE operations. The sector count is an unsigned, non-zero integer. All zeros in the sector count field specify a count of 256.

For a format command, the number of sectors to be formatted per track is specified by this byte. The interleave factor need not be explicitly furnished by the host, since it is implicitly contained in the interleave table furnished by the host.

Control Field

The control field is reserved for future use.

DMA Memory Address

The next three bytes, bytes 6, 7, and 8, make up the 23 bit address which points to the block of 512 byte to be transfer via DMA. This block of memory contains data bytes or specifies an address value as required by the command to be executed. Since the R/W- bit is part of the LD2 memory address counter, address bits A1-A23 are shifted right 1 bit by the HDC before being stored for command execution.

Status and Error Bytes

At the completion of each command the HDC will return status in the last four bytes (12-15) of the command block. The status format is similar to that returned by the 'Request Sense' SCSI command. This four byte block contains error and status information pertaining to the last block of data transferred or a non-disk operation executed by the HDC. The ADV bit will be set, to indicate a valid address, if the last operation required a disk access, otherwise ADV = 0.

The logical unit number returned is simply the contents of the logical unit field, where the error occurred, as defined in the drive control block. For those commands that do not take a logical unit number as an input parameter, the logical unit number returned in the command status byte is not meaningful.

A list of possible error codes, along with their descriptions, follows:

Error Bytes

The logical sector address bytes are to be in the same format as that defined in the command block. Bits 3-0 of the error byte is used for the error codes. Bits 4, 5 indicate the error type and 7 is the ADV bit. Bit 6 is not used presently.

Disk Drive Error Codes (Type 0)

0 No error
1 No Index
2 Seek not complete
3 Write fault
4 Drive not ready
6 Track 0 not found

Controller Error Codes (Type 1)

11 Uncorrectable data error
12 Address mark not found
13 Sector not Found, Read
14 Sector not Found, Write
15 IDNF error

Command Error Codes (Type 2)

20 Invalid command
21 Invalid sector address
22 Invalid LUN

Hardware Error Codes (Type 3)

30 RAM failure (HDC)
31 ROM Checksum Error
32 Host DMA status error

Error Code Description

No Error

A code of 00 or 80 is returned if no errors were detected during the execution of the last operation.

No Index (1)

The HDC does not detect index signal from drive.

Seek in Progress (2)

This error code is only returned by the test drive ready command when the target drive is a hard disk that supports buffered seeks. It indicates that drive is busy doing a buffered seek. No other command will be executed on the selected drive, until the seek is completed.

Write Fault (3)

This error code is returned by the hard disk drives. It indicates that there was write current to the head when the write gate was off. This is a very serious problem and should be fixed immediately. No command will be executed, when this condition is detected.

Drive Not Ready (4)

No disk operations are executed unless the drive is ready.

Track 0 Not Found (6)

This error code is only returned by the recalibrate command. It indicates that the track 0 status from the drive did not become active after the maximum necessary steps towards cylinder 0. Besides drive malfunction, this type of error usually occurs if more than 1 disk drive is selected at the same time, either by the HDC or by the option switches on the supported drives.

Uncorrected Data Error (11)

For a Winchester drive this error code indicates one or more error bursts in the data field were beyond the error correction code's ability to correct. It could also mean that the HDC was unable to obtain a match of two consecutive syndromes within eight read attempts. The sector data for the sector in error is sent to the host, prior to any retries and correction algorithms used.

Address Mark Not Found (12)

It indicates that the header for the target sector was found, but its address mark was not detected. This is treated like a data field error, except that no data transfer to the host takes place. If the error per-

sists after 8 attempts, an auto-restore is performed, followed by a reseek, and another 8 attempts to read the desired LSA.

Sector Not Found, Read (13)

The HDC found the correct cylinder and head but not the target sector.

Sector Not Found, Write (14)

The HDC found the correct cylinder and head but not the target sector.

I.D. Not Found (15)

If the ID field cannot be read correctly after all the retries have been exhausted, this error code is set and the operation terminated. The WDC searches for the ID field twice, followed by another 8 attempts by the HDC.

Format Error (1A)

During a check track command the HDC detects one of the following errors:
1) Track not found.
2) Bad ID

Illegal Parameters (20, 21, 22)

These error codes, invalid command (20), illegal LSA (21), and illegal LUN (22) are self explanatory.

HDC RAM Error (30)

During internal diagnostic the HDC detects a RAM error.

HDC ROM Checksum Error (31)

During internal diagnostic the HDC detects a ROM checksum error.

Host DMA Failure

This error code is set whenever invalid status is read from the DMA during any data or command access. For most operations the status checked is E0 (hex), except for a block write. In this case the valid status checked for is A0. The status is read continuously for about 20 mS.

COMMAND DESCRIPTION

All commands executed by the HDC are summarized in the table below. Fields of the command block not specified are don't cares. Following this summary is a generalized description of the commands.

Table 5-6. Command Summary

Command Description	Class Opcod	LUN Num	LADD (21)	Int/ BCNT	Control Options	Possible Error Codes
Read Drive Status	00	0-1				RDS
Restore to TKO	01	0-1				06, RDS
Request Status	03	0-1				Last Oper.
Check Trk Fmt	05	0-1	L		R	RDE, RDS, IDA
Format Track	06	0-1	L	B	S	IDA
Read Drive	08	0-1	L	B	R, S	RDE, RDS, IDA
Write Drive	0A	0-1	L	B	R, S	15, 19, RDS, IDA
Seek	0B	0-1	L			RDS, IDA
Set Drive Param.	0C	0-1				20, 32
Change Command Block Address/	0F					20, 32
Read Drive Long	E5	0-1	L	B	R, S	RDE, RDS, IDA
Write Drive Long	E6	0-1	L	B	R, S	15, 19, RDS, IDA
Init. Unit 1	CC	1				20, 32

R = 0 Retries/ECC enable
= 1 Retries/ECC disabled

S = 0 Set correction span to 5 bits
= 1 Set correction span to 11 bits

L = Logical Sector Address

B = Block or sector count required

Read Drive Status (RDS) = 02, 03, 04, 20, 32
Illegal Disk Access (IDA) = 20, 21, 22, 32
Read Sector Error (RDE) = 11, 12, 13, 14, 15

Read Drive Status (Class 0, Opcode 0)

Action

Read the drive's status and determine if drive is ready. For Hard disk drives supporting buffered seeks this command is useful for determining the first drive to reach its target track. The command will be aborted, if the drive status read is incorrect.

Possible Error Codes

No error, invalid command, seek in progress, drive not ready, write fault, DMA error.

Restore (Class 0, Opcode 1)

Action

The Restore command positions the heads to cylinder 0. It is usually issued by the host when the drive has been turned on, or before a format drive operation is initiated by the host.

Possible Error Codes

No error, invalid command, Track 0 not found, drive not ready, write fault, DMA error.

Request Status (Class 0, Opcode 3)

Action

Send the host four bytes of error information for the specified drive. The status of the last command executed may have already set the error register but the execution of this command will not set any new bits. If however, the command requesting the status is invalid, then the previous command status will be lost.

Possible Error Codes

No error, invalid command, last operation status, DMA error.

Check Track Format (Class 0, Opcode 5)

Action

Verify that the specified track is formatted with the correct number of logical sectors. A multiple read command is issued by the HDC to verify all the ID fields on that track and the data read back from the disk is discarded. Retries maybe enabled if desired.

Possible Error Code

No error, invalid command, invalid sector address, IDNF error, drive not ready, write fault, invalid LUN, seek not complete, DAM found, uncorrectable data error, DMA error.

Format Track (Class 0, Opcode 6)

Action

The format track command is used for initializing the ID and data fields on a specified track. The current contents of the specified track are overwritten. This command is useful for marking any bad sectors or tracks after the entire disk surface has been formatted. Assignment of alternate tracks or simply not specifying bad logical addresses is best handled by the host driver routines in the interest of flexibility and reducing onboard firmware requirements.

Possible Error Codes

No error, invalid command, invalid sector address, drive not ready, seek not complete, write fault, invalid LUN, DMA error.

Interleave Considerations

During this command the sector is set up by the host to contain additional parameter information instead of data. Each sector requires a two byte sequence. The first byte designates if a bad block (80) or a good block (00) is to be recorded in the ID field. The second byte indicates the logical sector number to be recorded on the disk, as shown below:

Table 5-7. Interleave Factor Table

Addr. in Hex	Data for an Interleave factor of: (Hex) 1	2	3	4
00	00	00	00	00
01	00	00	00	00
02	80	00	00	00
03	01	09	06	0D
04	00	80	00	00
05	02	01	0C	09
06	00	00	80	00
07	03	0A	01	05
08	80	00	00	80
09	04	02	07	01
0A	00	00	00	00
0B	05	0B	0D	0E
0C	00	00	00	00
0D	06	03	02	0A
0E	00	00	00	00
0F	07	0C	08	06
10	00	80	00	00
11	08	04	0E	02
12	00	00	00	00
13	09	0D	03	0F
14	00	00	00	00
15	0A	05	09	0B
16	00	00	00	00
17	0B	0E	0F	07
18	00	00	80	00
19	0C	06	04	03
1A	00	00	00	00
1B	0D	0F	0A	10
1C	00	00	00	00
1D	0E	07	20	0C
1E	00	00	00	00
1F	0F	10	05	08
20	00	00	00	80
21	10	08	0B	04
All Rest	XX	XX	XX	XX

These numbers can be from 00 to 10 (hex), or 17 sectors per track or any number that the host wishes to specify that meets the drive track capacity. Bad block marks are shown for sector numbers 1 and 4 in all four interleave factors illustrated. The other requirement of the host is to provide the logical sector number. Using this scheme, sectors can be recorded in any interleave factor desired. Byte four of the command block then specifies the number of sectors to be formatted per track. Also the host is free to choose marking individual sectors or entire tracks bad. At the end of a track format, the host can re-issue the command, for formatting the track across head boundaries as shown below:

Table 5-8: Interleaving Across Head Boundaries

00	01	02	03	04	. . . . .	0E	0F	10
10	00	01	02	03	. . . . .	0D	0E	0F
0F	10	00	01	02	. . . . .	0C	0D	0E
0E	0F	10	00	01	. . . . .	0B	0C	0D

Using the above spiral format approach, the HDC has approximately 1 mS for any processing overhead required. This 1 mS loss in the 1:1 performance across head boundaries, assuming a disk rotational speed of 3600 r.p.m. is reasonable. Across cylinder boundaries, the 1:1 interleave factor cannot be maintained because of the step rates involved. To format the entire disk using the Format Track command the host must update the buffer, if desired, and re-issue the command every track formatted. This is not really a major advantage since the host driver routines can easily re-issue the command in a loop until the entire disk is formatted. This gives the host total flexibility to format the drive using any clever algorithms for formats across head and cylinder boundaries instead of a canned approach.

Physical Track Format

The data fields are filled with FF hex, and the ECC is generated as specified by the related coding options. The Gap 3 value is determined by the drive motor speed variation, data sector length, and the interleave factor. The interleave factor is only important when 1:1 interleave is used. The formula for determining the minimum Gap 3 is:

$$\text{Gap 3} = 2 \times M \times S + K + E + V$$

M = motor speed variation (e.g. .01 for +/− 1%)
S = sector length in bytes
K = 18 for an interleave factor of 1
E = 2 if ECC is enabled
V = number of overhead bytes required for the HDC between sectors
= 9 (for an interleave factor of 1)

To maximize data read back efficiency and maintain the interleave factor of one, as closely as possible, it is required that the physical sector numbers be offset by a sector from track to track, (see table) so that the HDC has a sector length available for overhead to switch heads while on the same cylinder.

Read Drive (Class 0, Opcode 8)

Action

Read the specified number of consecutive sectors beginning with the specified sector in the command block to the host computer. If ECC is enabled, ECC bytes are recomputed by the HDC. After the data is transferred to the host, the recorded ECC bytes are compared to the generated bytes to generate the syndrome bytes. If the syndrome is non-zero, errors have occurred. Error correction is invoked by the HDC if two consecutive syndromes match, otherwise a maximum of 8 retries are attempted by the HDC.

Possible Error Codes

No error, invalid command, invalid sector address, invalid LUN, IDNF error, bad block mark, address mark not found, uncorrectable data error, write fault, drive not ready, seek in progress, DMA error.

Write Drive (Class 0, Opcode A)

Action

The Write Sector command is used to write the specified number of sectors of data from the host computer to the disk, beginning with the specified logical address in the command block. The write operation is identical to the read, except for error handling and reading the host status.

Possible Error Codes

No error, invalid command, invalid sector address, invalid LUN, drive not ready, IDNF error, bad block mark, write fault, seek in progress, DMA error.

Seek (Class 0, Opcode B)

Action

The Seek command positions the R/W head to the cylinder contained in the logical address. No ID field is read to verify start or end position. Seek It is primarily used to move the R/W head to the Shipping zone for transportation of the hard disk.

Possible Error Codes

No error, invalid command, invalid sector address, invalid LUN, drive not ready, write fault, DMA error.

Set Drive Parameters (Class 0, Opcode C)

Action

This command points to a 6 byte block of memory, specified by bytes 6 and 7 of the command block, that sets the following parameters for both of the hard disk drives (logical units 0 and 1):

Table 5-9. Set Drive Parameters

D7	D6	D5	D4	D3	D2	D1	D0
User Options				Step Rate			
0	Num. of Heads			CYL. Nums. MSN			
Number of Cylinders LSB							
Precompensation Cylinder / 16							
Reduce Write Current Cylinder / 16							
Number of Sector per Track							

If the above command is not executed after power up or every reset, the HDC will assume the following default parameters:

306. = Number of cylinders (131 hex)
4 = Number of heads
128. = Starting write precompensation cylinder
128. = Reduce write current cylinder
3 mS = Step rate
5 = Maximum length of an error bust to be corrected
17. = Number of sectors per track
8. = Retries & ECC enable

The acceptable range of values for these parameters are as follows:

0 – 2047. Number of cylinders
0 – 7 Number of heads
0 – 255 Sector Numbers
0 – 1023. Starting write precompensation cylinder
5 / 11. Maximum length of error burst to be corrected
0 / 8 Retries

If one of the parameters is out of range, then an "invalid command" error code is generated by the HDC. Bytes 2 and 5 of table are self explanatory and will not be discussed any further.

User Options

This four bit field can be used to specify options as indicated below:

Bit 7 = 0 5 bit correction span (default value)
= 1 11 bit correction span

Bit 6 = 0 Retries & ECC enabled (default value)
= 1 Retries & ECC disabled

Bit 5 = 0 Not Used

Bit 4 = 0 Not Used

Step Rate

Step Rate 14 = 11.1 usec
Step Rate 15 = 30 usec
All Others = 3 msec

Possible Error Codes

No error, invalid command, DMA error.

Initialize Unit 1 (Opcode CC)

Action

This command with initialize or set drive parameters of unit 1 only. This allows for the HDC to support two different drive types at the same time. The action of this command is identical to the action of the 'Set Drive Parameter' command noted above except that it will effect only unit 1. For command details see section 6.2.9.

Change Command Block (Class 0, Opcode F)

Action

The Change Command Block is used to move the location of the command block from the default on power up to a new location. Bytes 6 and 7 of the command block are used as indirect address pointers for the beginning of a 7 byte block of memory organized as follows:

Table 5-10. Change Command Block Address

D7	D6	D5	D4	D3	D2	D1	D0
0							0
A23			High Order DMA Byte				A16
A15			Mid Order DMA Byte				A08
A07			Low Order DMA Byte				0

Since the host R/W bit, and address bits A23-A17, form the data byte for the host LD2- counter, the DMA high and middle order address bytes are shifted right 1 bit position before being used. Since a copy of the previous address is not maintained, the command status is returned to the new address location specified and not the old one.

Possible Error Codes

No error, invalid command, DMA error.

Read Long (Class 7, Opcode 5)

Action

Similar to Read Sector except the ECC operation producing the syndrome is inhibited in the HDC. Instead the HDC copies the recorded CHECK bytes from the disk and passes them unaltered to the host. This command is useful in debugging and verifying the ECC hardware and software. To do this first write normally, and then READLONG. The data or the check bits may now be altered by the host and written to the disk using the WRITELONG command. If a READ command were issued, then the HDC should invoke error correction on the data field and correct it as long as the error induced is within the correction capability of the ECC polynomial.

Because there is no storage register on board, this command is implemented only for diagnostic purposes. Also note that the 4 extra checkbytes are to be accessed directly to the host. Hence the diagnostic tester used is required to support a 516 byte block transfer instead of the standard 512 byte block transfer supported by the Amiga system.

Possible Error Codes

No error, invalid command, invalid sector address, invalid LUN, IDNF error, bad block mark, address mark not found, write fault, drive not ready, seek not complete, DMA error.

Write Long (Class 7, Opcode 6)

Action

The Write Long command functions similarly to the Write Sector command except the ECC operation of computing the ECC word is inhibited in the HDC. Instead, the HDC accepts a 32 bit appendage from the host and passes it unaltered to the DJC to be written on the disk after the data. This command is useful for diagnostic purposes only. It allows the generation of a sector containing a correctable ECC error. See the Read Long command description for operation details and system requirements.

Possible Error Codes

No error, invalid commands, invalid sector address, invalid LUN, IDNF error, bad block mark, write fault, seek not complete, drive not ready, DMA error.

Fat Agnus Chip

DESCRIPTION

This specification describes the Fat Agnus chip, an N-channel HMOS DMA Controller. This IC device is able to produce, in a 68000 microprocessor environment, DMA addresses by using a RAM Address Generator and a Register Address Encoder. This device contains 25 DMA channel controllers, including the Blitter, Bitplanes, Copper, Audio, Sprites, Disk and Memory refresh.

The IC accepts a 28.63636 MHz crystal clock for the purpose of generating 7.16 MHz and 3.58 MHz system clocks, dynamic RAM interface for addressing up to 1 megabyte of memory and NTSC video synchronization pulses.

Refer to Figure 6.1 for pin configuration, Figure 6.2 for IC block diagram and Table 6-1 for pin description.

This IC device is equivalent to an 8370.

Warning

Improved versions of the Amiga custom chips are under development. These chips are intended to be software compatible with the existing chips. Writing incorrect values to reserved bits, accessing undefined register addresses, reading write-only registers or excessive cleverness may lead to compatibility problems.

CONFIGURATION

This IC device is configured in a standard 84 pin plastic chip carrier package.

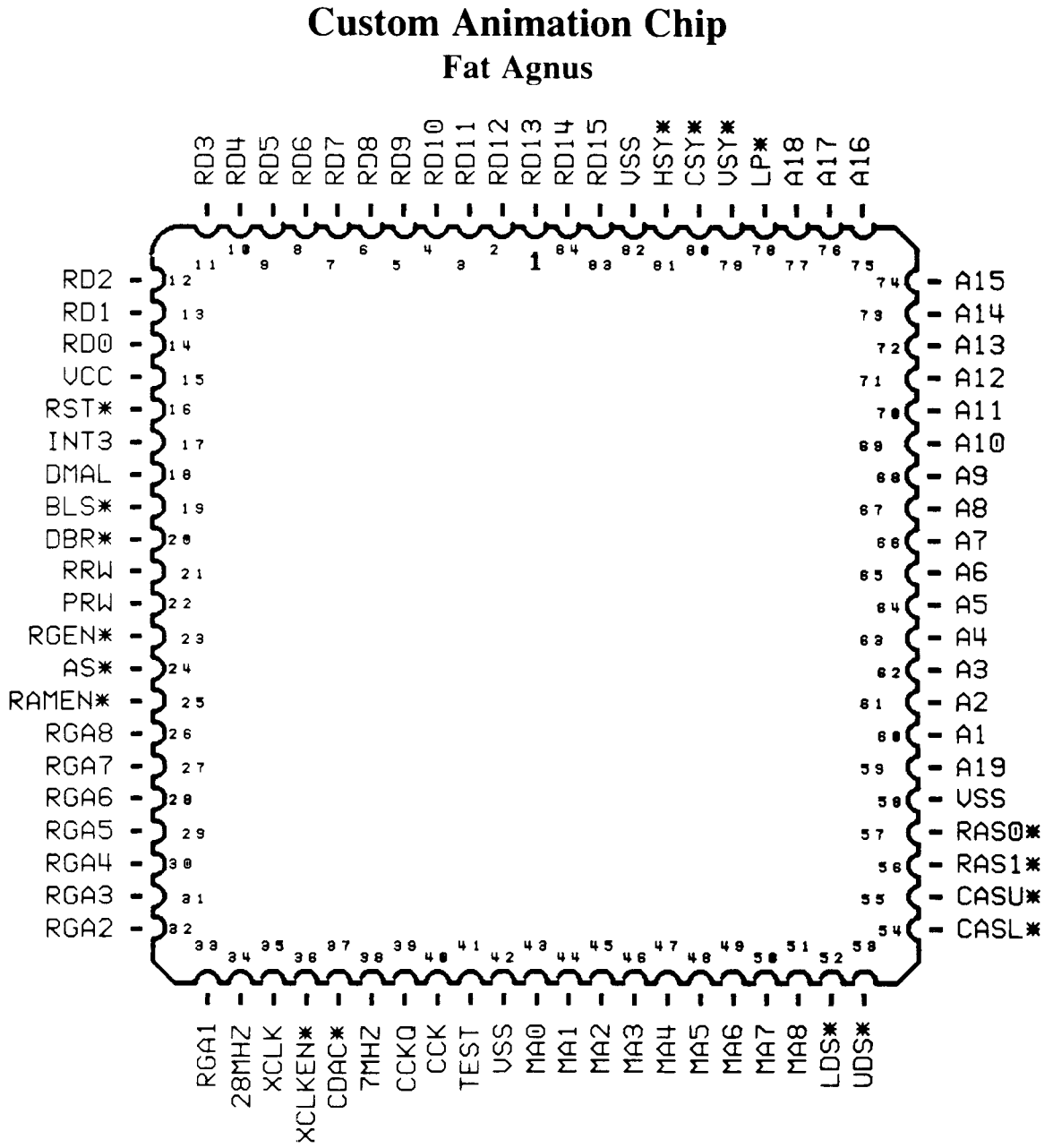


Figure 6.1 Configuration

Fat Agnus Block Diagram

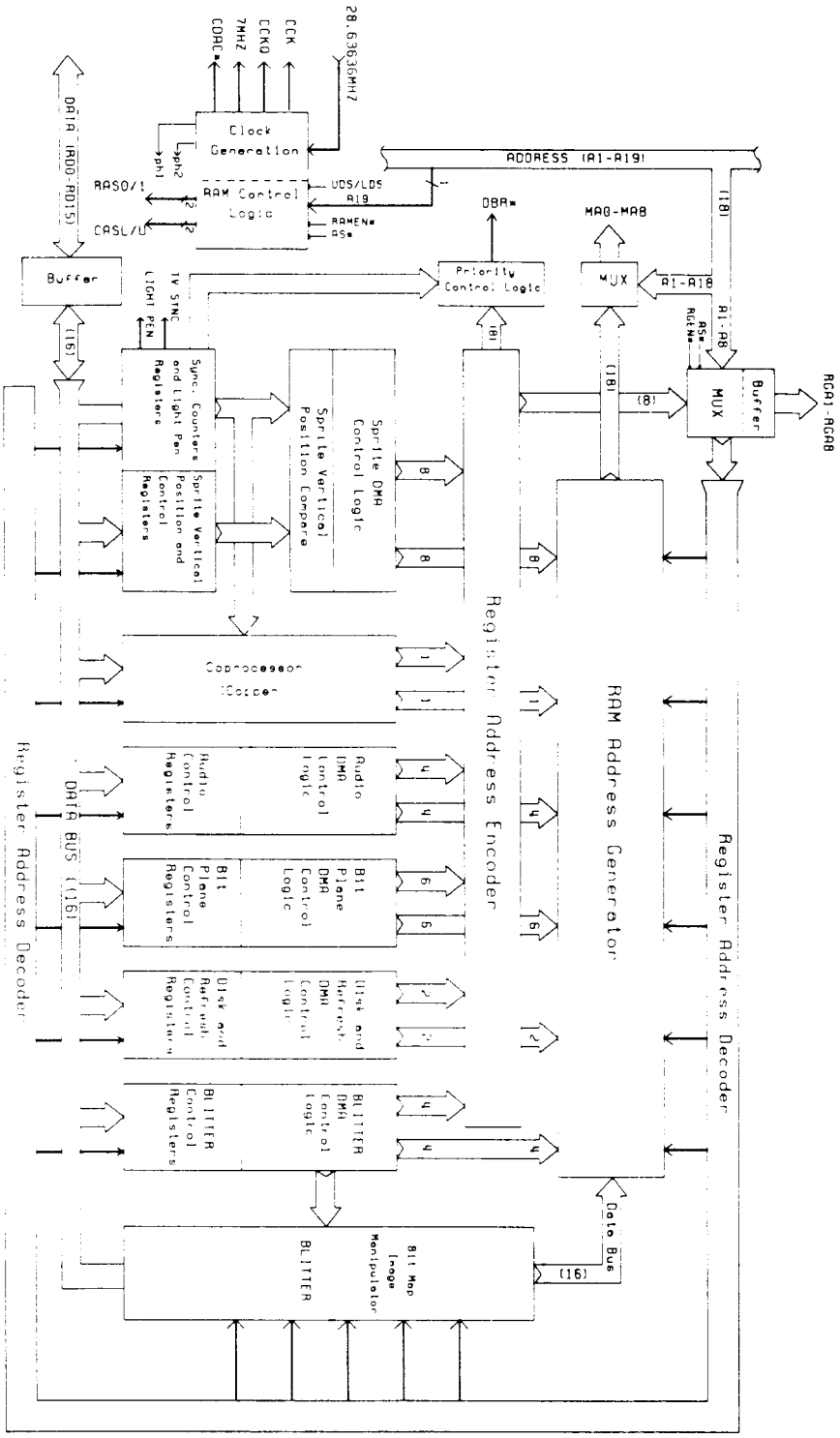


Figure 6.2 Block Diagram

Table 6-1 Pin Description

PIN NAME	PIN NUMBER	SIGNAL DIRECTION	DESCRIPTION
A19-A1	59 thru 77	IN	Address bus—A1 to A8 are used by the processor to select the internal registers and put an address code on the RGA lines to select registers outside the device. The processor uses A1 to A18 to generate multiplexed DRAM addresses on the MA outputs. The A19 line is used to indicate which RAS line is activated. If A19 is high, RAS1* is asserted; if low, RAS0* is asserted.
RD15-RD0	1 thru 14 and 83 & 84	I/O	This data bus is buffered and is used by the processor to access the device registers. The data bus is also accessed during DMA operations.
AS*	24	IN	Active low. This input is the processor address strobe signal. When asserted, it indicates that the address lines (A1 to A19) are valid.
RGEN*	23	IN	Active low. When this signal is asserted along with AS*, the processor uses A1 to A8 to access one of the device registers or put a value on the RGA outputs to select registers outside the device.
RAMEN*	25	IN	Active low. When this signal is asserted together with AS*, the processor is doing a DRAM access. The processor supplies an address on the A1 to A18 inputs and the device multiplexes this address onto the MA outputs; during the same cycle, the processor controls the A19 line to select one of the RAS lines.
PRW	22	IN	This signal defines the data bus transfer as a read or write cycle to memory. The signal is only enabled when the processor is undergoing a DRAM access. A low on this signal signifies a processor write cycle to memory; a high indicates a processor read cycle from memory.
RRW	21	OUT	The device controls this signal to indicate either a DMA or processor DRAM read/write access. In both cases, a low on this line indicates a write operation and a high indicates a read operation.

PIN NAME	PIN NUMBER	SIGNAL DIRECTION	DESCRIPTION
MA0-MA8	43 thru 51	OUT	Output bus. This 9 bit output bus provides multiplexed addresses to DRAMs. This bus operates in two cycles. The first cycle provides the DRAMs with the row address, the second cycle with the column address. It includes full 512K addressing for use with 256KX1 DRAMS. The IC only activates this bus when the processor is doing a DRAM access (RAMEN* is low) or when the device itself is performing a DMA data transfer (DBR* is low).
LDS*	52	IN	Active low. This input is the processor lower data strobe. It is enabled only during a processor DRAM access and forces the IC to assert CASL*.
UDS*	53	IN	Active low. This input is the processor upper data strobe. It is enabled only during a processor DRAM access and forces the IC to assert CASU*.
CASL*	54	OUT	Active low. This output strobes the column address into the DRAMS and corresponds to the low byte of the data word.
CASU*	55	OUT	Active low. This output strobes the column address into the DRAMS and corresponds to the high byte of the data word.
RAS0*	57	OUT	Active low. This output is used to strobe the row address into the DRAMs. This signal is asserted only if the processor is doing a DRAM access and A19 is low or if the IC is performing a DMA cycle (DBR* is low). RAS0* corresponds to the lower 512K bytes of memory.
RAS1*	56	OUT	Active low. This output is used to strobe the row address into the DRAMs. This signal is asserted only if the processor is doing a DRAM access and A19 is high. The signal is not asserted when the device is doing a DMA cycle. RAS1* corresponds to the upper 512K bytes of memory.
DBR*	20	OUT	Active low. The device asserts this signal to indicate that a DMA cycle is underway. The device performs only DMAs on the lower 512K bytes of memory when DBR* is low and RAS0* is asserted. The only exception is when the device is performing a DRAM refresh, in which case RAS0*, RAS1* and DBR* are all asserted. The device also asserts both CASL* and CASU* during DMAS except on a DRAM refresh cycle.

PIN NAME	PIN NUMBER	SIGNAL DIRECTION	DESCRIPTION
RGA8-RGA1	26 thru 33	OUT	Output bus. The 8 bit output bus allows the device and the processor to access registers located outside the device.
HSY*	81		This line is bidirectional and buffered. This signal is the horizontal synchronization pulse and is NTSC compatible. When set as an input, an external video source drives this signal to synchronize the horizontal beam counter.
VSY*	79		This line is bidirectional and buffered. This signal is the vertical synchronization pulse and is NTSC compatible. When set as an input, an external video source drives this signal to synchronize the vertical beam counter.
CSY*	80	OUT	This signal is the composite video synchronization pulse and is NTSC compatible.
LP*	78	OUT	Active low. This input is used to indicate when the light pen is coincident with the monitor beam.
RST*	18	IN	Active low. This input initializes the device to a known state.
INT3*	17	OUT	Active low. The device asserts this line to indicate that the blitter has completed the requested data transfer and that the blitter is then ready to accept another task.
DMAL	18	IN	Active high. When this signal is enabled, it indicates that an external device is requesting audio and/or disk DMA cycles to be executed by the device.
BLS*	19	IN	Active low. When this line is asserted, the device suspends its blitter operation and allows the processer to have control of the cycle.
28MHZ	34	IN	This is a 28.63636MHz input clock that provides the master time base for the device. This clock is enabled only when XCLKEN* is high.
XCLK	35	IN	This input is an alternate master clock to the device. It is enabled when XCLKEN* is low. This input is used to synchronize the device with an external video source.

PIN NAME	PIN NUMBER	SIGNAL DIRECTION	DESCRIPTION
XCLKEN*	36	IN	This input is used to select the master clock to the device. If it is high, the 28MHz input is enabled; if low, the XCLK is enabled.
CCK	40	OUT	This signal is a clock, which is obtained after dividing the 28.63 MHZ clock by eight. It is also known as the color clock frequency for NTSC applications.
CCKQ	39	OUT	This clock is the CCK clock shifted by 90 degrees.
7MHZ	38	OUT	This clock is obtained after dividing the 28MHZ clock by four.
CDAC*	37	OUT	This clock is obtained after inverting the 7MHZ clock and shifting it by 90 degrees.
TEST	41	IN	Active high. When this signal is asserted, it disables the processor cycle and the 8370 internal registers can be accessed on every CCK clock cycle.

MODES OF OPERATION

General Information

This device is an address generator type IC. Its main function is as a RAM address generator and register address encoder that produces all DMA addresses from 25 channels.

The block diagram (Figure 6.3) for this device shows the DMA control and address bus logic. The output of each controller indicates the number of DMA channels driving the Register Address Encoder and RAM Address Generator.

The Register Address Encoder is a simple PLA type of structure that produces a predetermined address on the RGA bus whenever one of the DMA channels is active.

The RAM Address Generator contains an 18-bit pointer register for each of the 25 DMA channels. It also contains pointer restart (backup) registers and jump registers for six (6) of the channels. A full 18-bit adder carries out the pointer increments and adds for jumps.

The priority control logic looks at the pipe-lined DMA requests from each controller and stages the DMA cycles based upon their programmed priority and sync counter time slot. Then it signals the processor to get off the bus by asserting the DBR line.

The following is a brief description of the device's major operational modes.

Blitter

The procedure for moving and combining bit-mapped images in memory received the name Bit Blit from a computer instruction that did block transfers of data on bit boundaries. These routines became known as Bit Blitters or Blitters. The Blitter DMA Controller is preloaded with the address and size of three source images (A, B, and C) and one destination (D) in the dynamic RAM (refer to Figure 6.3). These images can be as small as a single character or as large as twice the screen size. They can be full images or smaller windows of a larger image. After one work of each source image is sequentially loaded into the source buffer (A, B, C) they are shifted and then combined together in the logical unit to perform image movement overlay, masking, and replacements. The result is captured in the destination buffer (D) and sent back to the RAM memory destination address.

This operation is repeated until the complete image has been processed. The unit has extensive pipelining to allow for shifter and logic unit propagation time, while the next set of source words is being fetched.

A control register determines which of 256 possible logic operations is to be performed as the source images are combined and how far they are to be moved (barrel shifted). In addition to the image combining and movement powers, the Blitter can be programmed to do line drawing or area fill between lines.

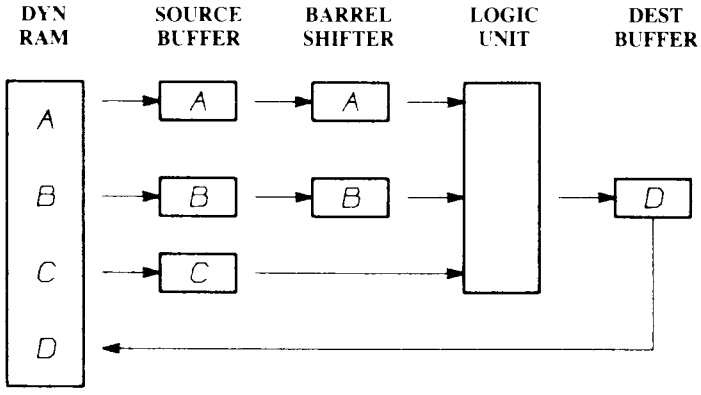


FIGURE 6.3. Blitter Block Diagram

Bitplane Addressing

Some computer bitmap displays are organized so that the bitplanes for each pixel are all located within the same address. This is called *pixel addressing.* If the entire data word of one address is used for a single pixel with 8 bit planes, the data word will look like this. (numbers are bitplanes):

12345678--------

The data compression can be improved by packing more than one pixel into a single address like this:

1234567812345678

or like this, if there are only 4 bitplanes:

1234123412341234

The IC device uses a bitmap technique called *Bitplane Addressing.* This separates the bitplanes in memory. To create a 4 plan (16 color) image, the bitplane display DMA channels fetch from 4 separate areas of memory like this:

1111111111111111
2222222222222222
3333333333333333
4444444444444444

These are held in buffer registers and are used together as pixels, one bit at a time, by the display (left to right).

This technique allows reduced odd numbers of bitplanes (such as 3 or 5) while maintaining packing efficiency and speed. It also allows grouping bitplanes into two separate images, each with independent hardware high speed image manipulation, line draw, and area fill.

DMA Channel Functions

Each channel has an 18 bit RAM address pointer that is placed on the MA memory address bus and is used to select the location of the DMA data transfer from anywhere in 256K words (512K bytes) of RAM.

An eight bit destination address is simultaneously placed on the register address bus (RGA), sending the data to the corresponding register.

Figure 6.4 shows a typical DMA channel; almost all channels have RAM as source and chip registers as destination.

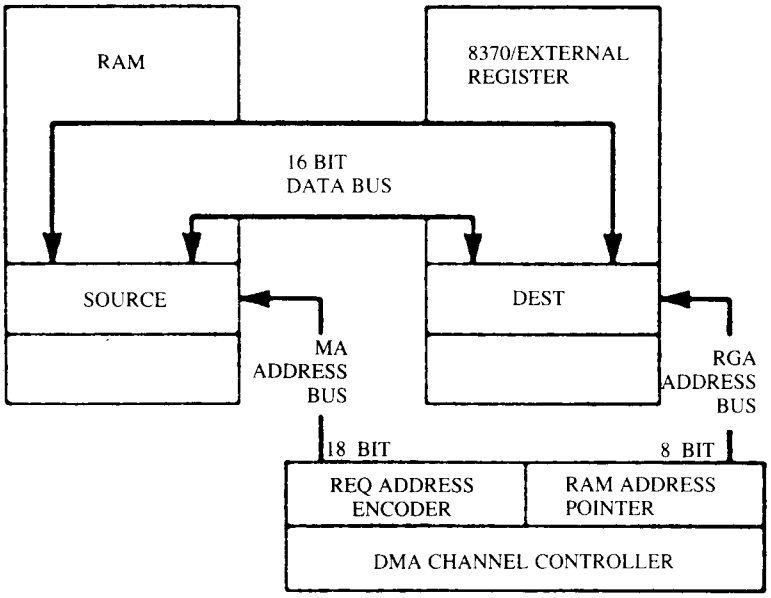


FIGURE 6.4. DMA CHANNEL (TYPICAL)

The pointer must be preloaded and is automatically incremented each time a data transfer occurs.

Each controller utilizes one or more of these DMA channels for its own purposes. The following is a brief summary of these controllers and the DMA channels they use.

A-BLITTER (4 CHANNELS)

The Blitter uses four DMA channels, three sources and one destination as previously described.

Once the Blitter has been started, the four DMA channels are synchronized and pipelined to automatically handle the data transfers without further processor intervention. The images are manipulated in memory, independent of the display (bitplane DMA).

B-BITPLANE (SIX (6) CHANNELS)

The bitplane controller continuously (during display) transfers display data from memory to display buffer registers. There are six DMA channels to handle the data from six independent bit planes. The buffers convert this bitplane data into pixel data for the display.

Each bitplane can be a full image or a window into an image that is up to four times the screen size. They can be grouped into two separate images, each with its own color registers.

C-COPPER (ONE (1) CHANNEL)

The Copper is a coprocessor that uses one of the DMA channels to fetch its instructions. The DMA pointer is the instruction counter and must be preloaded with the starting address of Copper's instructions.

The Copper can move (write) data into chip registers. It can skip, jump, and wait (halt). These simple instructions give great power and flexibility because of the following features.

When the Copper is halted, it is off the data bus, using no bus cycles until the wait is over. The programmed wait value is compared to a counter that keeps track of the TV beam position (Beam Counter) and when they are equal, the Copper will resume fetching instructions.

It can cause interrupts, reload the color registers, start the Blitter or service the audio. It can modify almost any register inside or outside the IC device, based on the TV screen coordinates given by the Beam Counter and the actual address encoded on the RGA bus.

D-AUDIO (FOUR (4) CHANNELS)

There are four audio channels, all of which are located outside of the DMA Controller IC. Each controller is independent and uses one DMA channel from the DMA Controller IC and fetches its data during a dedicated timing slot within horizontal blanking. This is accomplished by a controller asserting the DMAL input on the DMA Controller.

E-SPRITES (EIGHT (8) CHANNELS)

There are eight independent Sprite controllers, each with its own DMA channel and its own dedicated time slot for DMA data transfer. Sprites are line buffered objects that can move very fast because of their position are controlled by hardware registers and compacitors.

Each Sprite has two 16 bit data registers that define a 16 pixel wide Sprite with four colors. Each has a horizontal position register, a vertical start position register and a vertical stop position register. This allows variable vertical size sprites.

The Sprite DMA controller fetches image and position data automatically from anywhere in 512K of memory.

Sprites can be run automatically in DMA mode or they can be loaded and controlled by the microprocessor.

Each Sprite can be reused vertically as often as desired. Horizontal reusing is also possible with microprocessor control.

F-DISK (ONE (1) CHANNEL)

The disk controller, which is located outside of the DMA controller, uses a single DMA channel from the device. The controller uses this DMA time slot for data transfer and can read or write a block of data up to 128K anywhere in 512K of memory.

G-MEMORY REFRESH (ONE (1) CHANNEL)

The refresh controller uses a single DMA channel with its own time slots. It places RAS addresses on the memory address bus (MAS) during these slots, in order to refresh the dynamic RAM. Memory is refreshed on every roster line.

During the DMA no data transfer actually takes place. The register address bus (RGA) is used to supply video synchronizing codes. At this time, RAS0* and RAS1* are low and CASU* and CASL* are inactive.

RAM and Register Addressing

The device generates RAM addresses from two sources, the processor or from the device performing DMA cycles selected by a multiplexer. This multiplexer allows the processor to access RAM when AS* and RAMEN* are both low. At this time, the device also multiplexes the processor address (A1-A18) onto the MA bus. The device places A1 to A8 & A17 on the MA0 to MA9 outputs, respectively, during the row address time and places A9 to A16 & A18 on the MA0 to MA9, respectively, during the column address time. The A19 line is used by the IC to determine which RAS line is to be asserted. If A19 is low, RAS0* is enabled, and if high, RAS1* is enabled. The device also senses the LDS* and UDS* inputs to determine which CAS to drop. If LDS* is low, the IC will drop CASL*; if UDS* is low, CASU* is dropped.

When the device needs to do a DMA cycle, the multiplexer disables the processor from accessing RAM by asserting the Data Bus Request line (DBR*). At this time, the device multiplexes its generated RAM address onto the MA lines and will only make RAS0* 90 low, unless it is a refresh cycle where RAS1* will also go low. During a DMA cycle, the IC device also asserts both CASU* and CASL*, unless it is a refresh cycle where they both remain inactive.

The device also generates RGA addresses from either the processor or device DMAs, each of which is selected by another internal multiplexer. This multiplexer allows the processor to perform a register read/write access when AS* and RGEN* are both low. The device then takes the low order byte of the processor address A1 to A8 and reflects its value on the RGA output bus RGA1 to RGA8. The device will reflect the status of PRW input on the RRW output line, to indicate a memory read or write operation.

During a device DMA cycle, the multiplexer prevents the processor from doing a register access by asserting the DBR* line. The device then places the contents of its register address encoder onto the RGA bus.

REGISTER DESCRIPTION

This DMA controller device contains 97 registers that can be accessed after the following conditions have been met: the state of AS* and RGEN* must be an active low level and the least 8 significant address bits (A1 thru A8) must contain the valid address of the register to be accessed.

The following is a detailed description of the register set.

REGISTER	FUNCTION
AUD x LCH	*Audio channel x location (high 3 bits)*
AUD x LCL	*Audio channel x location (low 15 bits)*

This pair of registers contains the 18 bit starting address (location) of Audio channel x (x = 0,1,2,3) DMA data. This is not a pointer register and therefore only needs to be reloaded if a different memory location is to be outputted.

BLT x PTH *Blitter pointer to x (high 3 bits)*
BLT x PTL *Blitter pointer to x (low 15 bits)*

This pair of registers contains the 18 bit address of Blitter source (x = A,B,C) or dest. (x = D) DMA data. This pointer must be preloaded with the starting address of the data to be processed by the blitter. After the Blitter is finished it will contain the last data address (plus increment and modulo).

LINE DRAW: BLTAPTL is used as an accumulator register and must be preloaded with the starting value of (2Y-X) where Y/X is the line slope. BLTCPT and BLTDPT (both H and L) must be preloaded with the starting address of the line.

BLT x MOD *Blitter Modulox*

This register contains the Modulo for Blitter source (x = A,B,C) or Dest (x = D). A Modulo is a number that is automatically added to the address then points to the start of the next line. Each source or destination has its own Modulo, allowing each to be a different size, while an identical area of each is used in the Blitter operation.

LINE DRAW: BLTAMOD and BLTBMOD are used as slope storage registers and must be preloaded with the values (4Y-4X) and (4Y) respectively. Y/X = line slope BLTCMOD and BLTDMOD must both be preloaded with the width (in bytes) of the image into which the line is being drawn (normally 2 times the screen width in words).

BLTAFWM *Blitter first word mask for Source A*
BLTALWM *Blitter last word mask for Source A*

The patterns in these two registers are "anded" with the first and last words of each line of data from Source A into the Blitter. A zero in any bit overrides data from Source A. These registers should be set to all "ones" for fill mode or for line drawing mode.

BLT x DAT *Blitter source x data register*

This register holds Source x (x = A,B,C) data for use by the Blitter. It is normally loaded by the Blitter DMA channel, however, it may also be preloaded by the microprocessor.

LINE DRAW: BLTADAT is used as an index register and must be preloaded with 8000. BLTBDAT is used for texture. It must be preloaded with FF if no texture (solid line) is desired.

BLTDDAT *Blitter destination data register*

This register holds the data resulting from each word of Blitter operation until it is sent to a RAM destination. This is a dummy address and cannot be read by the micro. The transfer is automatic during Blitter operation.

BLTCON0 *Blitter control register 0*
BLTCON1 *Blitter control register 1*

These two control registers are used together to control Blitter operations. There are 2 basic modes, area and line, which are selected by bit 0 of BLTCON1, as shown below.

AREA MODE ("normal")

BIT#	BLTCON0	BLTCON1
15	ASH3	BSH3
14	ASH2	BSH2
13	ASH1	BSH1
12	ASA0	BSH0
11	USEA	X
10	USEB	X
09	USEC	X
08	USED	X
07	LF7	X
06	LF6	X
05	LF5	X
04	LF4	EFE
03	LF3	IFE
02	LF2	FCI
01	LF1	DESC
00	LF0	LINE(=0)

ASH3-0	Shift value of A source
BSH3-0	Shift value of B source
USEA	Mode control bit to use Surce A
USEB	Mode control bit to use Source B
USEC	Mode control bit to use Source C
USED	Mode control bit to use Destination D
LF7-0	Logic function minterm select lines
EFE	Exclusive fill enable
IFE	Inclusive fill enable
FCI	Fill carry input
DESC	Descending (decreasing address) control bit
LINE	Line mode control bit (set to 0)

LINE DRAW:

LINE MODE (line draw)

BIT#	BLTCON0	BLTCON1
15	START3	0
14	START2	0
13	START1	0
12	START0	0
11	1	0
10	0	0
09	1	0
08	1	0
07	LF7	0
06	LF6	SIGN
05	LF5	OVF
04	LF4	SUD
03	LF3	SUL
02	LF2	AUL
01	LF1	SING
00	LF0	LINE(=1)

START3-0 Starting point of line (0 thru 15 hex)

LF7-0 Logic function minterm select lines should be preloaded with 4A in order to select the equation $D = (\bar{A}C + AB\bar{C})$. Since A contains a single bit true (8000), most bits will pass the C field unchanged (not A and C), but one bit will invert the C Field and combine it with texture (A and B and not C). The A bit is automatically moved across the word by the hardware.

LINE Line mode control bit (set to 1)

SIGN Sign flag

OVF Word overflow flag

SING Single bit per horiz. line for use with subsequent Area Fill

SUD Sometimes Up or Down (=AUD*)

SUL Sometimes Up or Left

AUL Always Up or Left

The 3 bits above select the Octant for line draw:

OCT	SUD	SUL	AUL
0	1	1	0
1	0	0	1
2	0	1	1
3	1	1	1
4	1	0	1
5	0	1	0
6	0	0	0
7	0	0	0

Blitter start and size (Window, width height)

This register contains the width and height of the blitter operation (in line mode width must = 2, height = line length). Writing to this register starts the Blitter, and should be done last, after all pointers and control registers have been initialized.

BIT# 15, 14, 13, 12, 11, 10, 09, 08, 07, 06, 05, 04, 03, 02, 01, 00
h9 h8 h7 h6 h5 h4 h3 h2 h1 h0, w5 w4 w3 w2 w1 w0

h = Height = Vertical lines (10 bits = 1024 lines max)
w = Width = Horiz. pixels (6 bits = 64 words = 1024 pixels max)

LINE DRAW: BLTSIZE controls the line length and starts the line draw when written to. The h field controls the line length (10 bits gives lines up to 1024 dots long). The w field must be set to 02 for all line drawing.

BPL x PTH	*Bit plane x pointer (high 3 bits)*
BPL x PTL	*Bit plane x pointer (low 15 bits)*

This pair of registers contains the 18 bit pointer to the address of Bit plane x (x = 1,2,3,4,5,6) DMA data. This pointer must be reinitialized by the processor or Copper to point to the beginning of Bit Plane data every vertical blank time.

BPL1MOD	*Bit plane modulo (odd planes)*
BPL2MOD	*Bit plane modulo (even planes)*

These registers contain the Modulos for the odd and even bit planes. A Modulo is a number that is automatically added to the address at the end of each line, in order that the address then points to the start of the next line. Since they have separate modulos, the odd and even bit planes may have sizes that are different from each other, as well as different from the Display Window size.

BPLCONO	*Bit plan control register (miscellaneous control bits)*

This register controls the operation of the Bit Planes and various aspects of the display.

BIT#	BPLCON0
15	HIRES
14	BPU2
13	BPU1
12	BPU0
11	HOMOD
10	DBLPF
09	COLOR
08	GAUD
07	X
06	X
05	X
04	X
03	LPEN
02	LACE
01	ERSY
00	X

HIRES = High resolution (640) mode
BPU = Bit plane use code 000-110 (NONE through 6 inclusive)
HOMOD = Hold and Modify mode
DBLPF = Double playfield (PF1 = odd, PF2 = even bit planes)
COLOR = Composite video COLOR enable
GAUD = Genlock audio enable (mixed on BKGND pin during vertical blanking)
LPEN = Light pen enable (reset on power up)
LACE = Interlace enable (reset on power up)
ERSY = External Resync (HSYNC, VSYNC pads become inputs; reset on power up)

COPCON *Copper control register*

This is a 1-bit register that when set true, allows the Copper to access the Blitter hardware. This bit is cleared by power on reset, so that the Copper cannot access the Blitter hardware.

BIT#	NAME	FUNCTION
01	CDANG	Copper danger mode. Allows Copper access to Blitter if true.

COPJMP1 Copper restart at first location
COPJMP2 Copper restart at second location

These addresses are strobe addresses; when written to, they cause the Copper to jump indirect using the address contained in the First or Second Location registers described below. The Copper itself can write to these addresses, causing its own jump indirect.

COP1LCH	Copper first location register (high 3 bits)
COP1LCL	Copper first location register (low 15 bits)
COP2LCH	Copper second location register (high 3 bits)
COP2LCL	Copper second location register (low 15 bits)
COP1NS	*Copper instruction fetch identify*

This is a dummy address that is generated by the Copper whenever it is loading instructions into its own instruction register. This actually occurs every Copper cycle except for the second (IR2) cycle of the MOVE instruction. The three types of instructions are shown below:

MOVE	*Move immediate* to dest.
WAIT	*Wait until* beam counter is equal to, or greater than (keeps Copper off of bus until beam position has been reached).
SKIP	*Skip if* beam counter is equal to, or greater than (skips following MOVE inst. unless beam position has been reached).

	MOVE		WAIT UNTIL			SKIP IF		
BIT#	IR1	IR2	IR1	IR2		IR1	IR2	
15	X	RD15	VP7	BFD	*	VP7	BFD	*
14	X	RD14	VP6	VE6		VP6	VE6	
13	X	RD13	VP5	VE5		VP5	VE5	
12	X	RD12	VP4	VE4		VP4	VE4	
11	X	RD11	VP3	VE3		VP3	VE3	
10	X	RD10	VP2	VE2		VP2	VE2	
09	X	RD09	VP1	VE1		VP1	VE1	
08	DA8	RD08	VP0	VE0		VP0	VE0	
07	DA7	RD07	HP8	HE6		HP8	HE6	
06	DA6	RD06	HP7	HE7		HP7	HE7	
05	DA5	RD05	HP6	HE6		HP6	HE6	
04	DA4	RD04	HP5	HE5		HP5	HE5	
03	DA3	RD03	HP4	HE4		HP4	HE4	
02	DA2	RD02	HP3	HE3		HP3	HE3	
01	DA1	RD01	HP2	HE2		HP2	HE2	
00	0	RD00	1	1		1	1	

IR1 = First instruction register
IR2 = Second instruction register
DA = Destination Address for MOVE instruction. Fetched during IR1 time, used during IR2 time on RGA bus.
RD = RAM data moved by MOVE instruction at IR2 time directly from RAM to the address given by the DA field.
VP = Vertical Beam Position comparison bit
HP = Horizontal Beam Position comparison bit
VE = Enable comparison (mask bit)
HE = Enable comparison (mask bit)

*NOTE BFD = Blitter finished disable. When this bit is true, the Blitter Finished flag will have no effect on the Copper. When this bit is zero, the Blitter Finished flag must be true (in addition to the rest of the bit comparisons) before the Copper can exit from its wait state, or skip over an instruction. Note that the V7 comparison cannot be masked.

The Copper is basically a 2-cycle machine that requests the bus only during odd memory cycles (4 memory cycles per in). This prevents collisions with Display, Audio, Disk, Refresh, and Sprites, all of which use only even cycles. It therefore needs (and has) priority over only the Blitter and Micro.

There are only three types of instructions: MOVE immediate, WAIT until, and SKIP if. All instructions (except for WAIT) require 2 bus cycles (and two instruction words). Since only the odd bus cycles are requested, 4 memory cycle times are required per instruction (memory cycles are 280 ns).

There are two indirect jump registers, COP1LC and COP2LC. These are 18-bit pointer registers whose contents are used to modify the program counter for initialization or jumps. They are transferred to the program counter whenever strobe addresses COPJMP1 or COPJMP2 are written. In addition, COP1LC is automatically used at the beginning of each vertical blank time.

It is important that one of the jump registers be initialized and its jump strobe address hit, after power up but before Copper DMA is initialized. This insures a determined startup address and state.

DIWSTRT	*Display window start (upper left vertical-horizontal position)*
DIWSTOP	*Display window stop (lower right vertical-horizontal position)*

These registers control the Display Window size and position, by locating the upper left and lower right corners.

BIT# 15,14,13,12,11,10,09,08,07,06,05,04,03,02,01,00
USE v7 v6 v5 v4 v3 v2 v1 v0 h7 h6 h5 h4 h3 h2 h1 h0

DIWSTRT is vertically restricted to the upper ⅔ of the display (v8 = 0), and horizontally restricted to the left ¾ of the display (h8 = 0).
DIWSTOP is vertically restricted to the lower ½ of the display (v8 = / = v7), and horizontally restricted to the right ¼ of the display (h8 = 1).

DDFSTRT	*Display data fetch start (horiz.position)*
DDFSTOP	*Display data fetch stop (horiz.position)*

These registers control the horizontal timing of the beginning and end of the Bit Plane DMA display data fetch. The vertical Bit Plane DMA timing is identical to the Display windows described above. The Bit Plane Modulos are dependent on the Bit Plane horizontal size, and on this data fetch window size.

Register bit assignment

BIT# 15,14,13,12,11,10,09,08,07,06,05,04,03,02,01,00
USE X X X X X X X X HB H7 H6 H5 H4 H3 X X

(X bits should always be driven with 0 to maintain upward compatibility)

The tables below show the start and stop timing for different register contents.

DDFSTRT (Left edge of display data fetch)

PURPOSE	H8,	H7,	H6,	H5,	H4
Extra wide (max) *	0	0	1	0	1
wide	0	0	1	1	0
normal	0	0	1	1	1
narrow	0	1	0	0	0

DDFSTOP (Right edge of display data fetch)

PURPOSE	H8,	H7,	H6,	H5,	H4,
narrow	1	1	0	0	1
normal	1	1	0	1	0
wide (max)	1	1	0	1	1

DMACON *DMA control write (clear or set)*
DMACONR *DMA control (and Blitter status) read*

This register controls all of the DMA channels, and contains Blitter DMA status bits.

BIT#	FUNCTION	DESCRIPTION
15	SET/CLR	Set/Clear control bit. Determines if bits written with a 1 get set or cleared.
14	BBUSY	Blitter busy status bit (read only)
13	BZERO	Blitter logic zero status bit (read only)
12	X	
11	X	
10	BLTPRI	Blitter DMA priority (over CPU micro) —also called "Blitter Nasty" —disables /BLS pin, preventing micro from stealing any bus cycles while blitter DMA is running.

BIT#	FUNCTION	DESCRIPTION
09	DMAEN	Enable all DMA below.
08	DPLEN	Bit Plane DMA enable.
07	COPEN	Copper DMA enable.
06	BLTEN	Blitter DMA enable.
05	SPREN	Sprite DMA enable.
04	DSKEN	Disk DMA enable.
03	AUD3EN	Audio channel 3 DMA enable.
02	AUD2EN	Audio channel 2 DMA enable.
01	AUD1EN	Audio channel 1 DMA enable.
00	AUD0EN	Audio channel 0 DMA enable.

DSKPTH *Disk pointer (high 3 bits)*
DSKPTL *Disk pointer (low 15 bits)*

This pair of registers contains the 18-bit address of Disk DMA data. These address registers must be initialized by the processor or Copper before disk DMA is enabled.

REFPTR *Refresh pointer*

This register is used as a Dynamic RAM refresh address generator. It is writeable for test purposes only, and should never be written by the microprocessor.

SPRxPTH *Sprite x pointer (high 3 bits)*
SPRxPTL *Sprite x pointer (low 15 bits)*

This pair of registers contains the 18-bit address of Sprite x (x = 0,1,2,3,4,5,6,7) DMA data. These address registers must be initialized by the processor or Copper every vertical blank time.

SPRxPOS *Sprite x vertical-horizontal position data*
SPRxCTL *Sprite x vertical-horizontal*

These 2 registers work together as position, size and feature Sprite control registers. They are usually loaded by the Sprite DMA channel, during horizontal blank; however, they may be loaded by either processor any time.

SPRxPOS register:

BIT#	SYM	FUNCTION
15-08	SV7-SV0	Start vertical value. High bit (SV8) is in SPRxCTL reg. below.
07-00	SH8-SH1	Start horizontal value. Low bit (SH0) is in SPRxCTL reg. below.

SPRxCTL register (writing this address disables sprite horizontal comparator circuit):

BIT#	SYM	FUNCTION
15-08	EV7-EV0	End (stop) vert.value.low 8 bits
07	ATT	Sprite attach control bit (odd sprites)
06-04	X	Not used
02	SV8	Start vert. value high bit
01	EV8	End (stop) vert. value high bit
00	SH0	Start horiz. value low bit

VPOSR *Read vertical most significant bit (and frame flop)*

VPOSW *Write vertical most significant bit (and frame flop)*

BIT# 15,14,13,12,11,10,09,08,07,06,05,04,03,02,01,00
USE LOF_____V8
LOF = Long frame (auto toggle control bit in BPLCONO)

VHPOSR *Read vertical and horizontal position of beam or lightpen*

VHPOSW *Write vertical and horizontal position of beam or lightpen*

BIT# 15,14,13,12,11,10,09,08,07,06,05,04,03,02,01,00
USE V7 V6 V5 V4 V3 V1 V0,H8 H7 H6 H5 H4 H3 H1
RESOLUTION = 1/160 OF SCREEN WIDTH (280 NS)

AGNUS NOTES

1) The Agnus pointer registers are updated via a pipelining scheme that requires that a register not be accessed on two contiguous cycles.

   This precludes the use of "single operand" blitter functions that might seem possible based on the register descriptions.

   Caution is also required to prevent processor access to registers that may be subject to concurrent DMA access.

DMA Time Slot Allocation/Horizontal Line

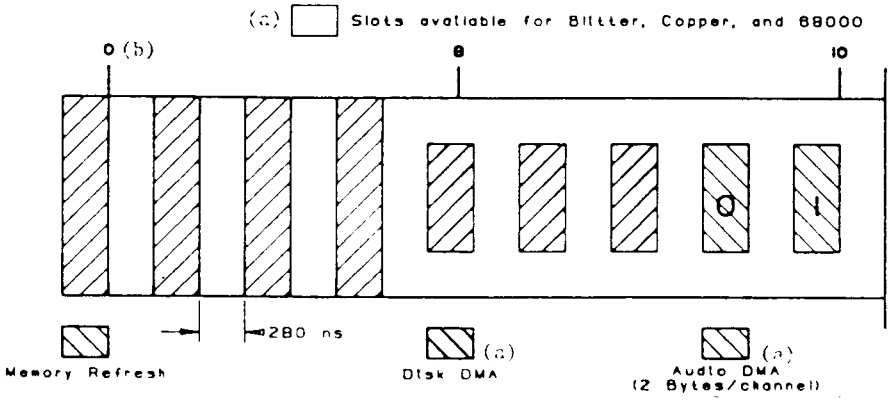
NOTES

1) These operations only take slots if the associated operation is being performed

   Note: Copper Data Move instructions require 4 slots.
   Copper Wait instructions require 6 slots.

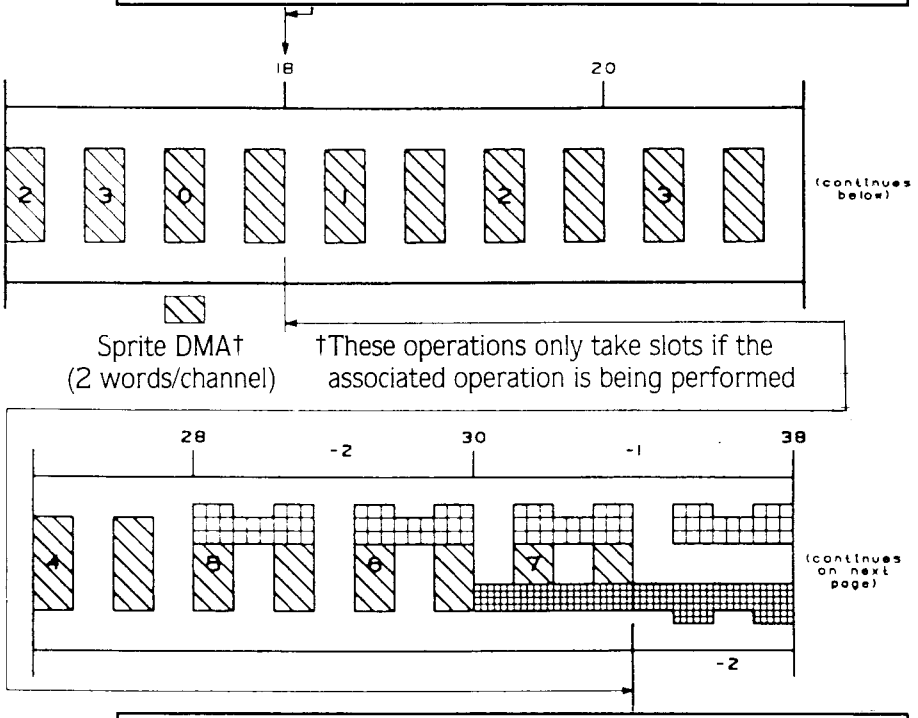
2) This cycle 0 appears to exclude one of the memory refresh cycles. This is not the case.

   Actual system hardware demands certain specific values for data fetch start and display start. Therefore this timing chart has been "adjusted" to match those requirements.



DMA Time Slot Allocation / Horizontal Line (Cont'd)

Hardware stop installed here. Data fetch cannot begin any sooner than cycle 18. This allows the user to wipe out most of the sprites if desired (by defining an extra-wide display) but leaves the audio and disk DMA untouched.

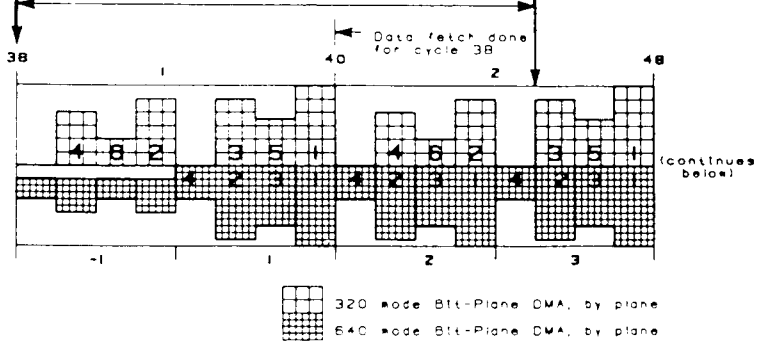


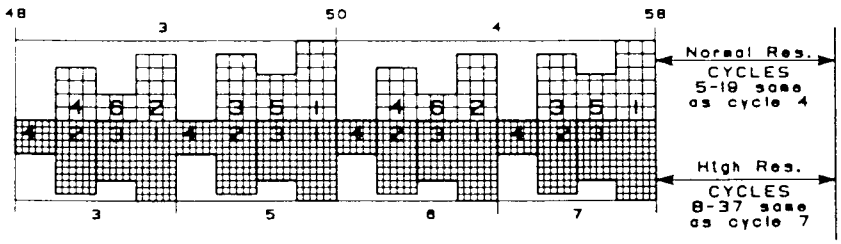
Some sprites are unusable if the display starts early due to an extra word(s) associated with a wide display and/or horizontal scrolling. In this case, the bit-plane DMA steals the cycles normally allocated to the sprites, as illustrated above.

DMA Time Slot Allocation / Horizontal Line (Cont'd)

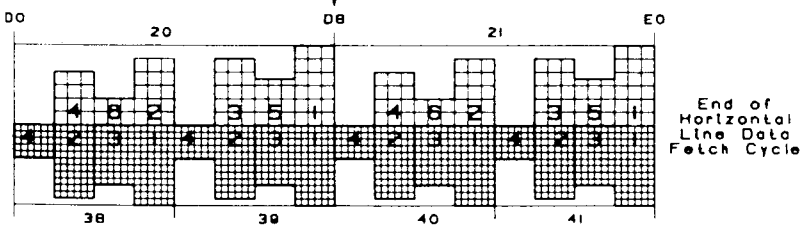
Data fetch start can only be specified at even multiples of 8 clocks. This is the clock position which should be specified for the normal width display. (20 word fetch for 320 pixel, 40 word fetch for 640 pixel width).

Five clocks must occur before the data which was fetched for a particular position can appear onscreen. For example, if data fetch start is specified as 38, it will not be available for display until clock number 45.





A hardware data-fetch stop has been installed at count D8 so as to prevent the bit-plane data-fetch from overrunning the time allotted for the memory refresh or disk DMA.



The 8520 Chip

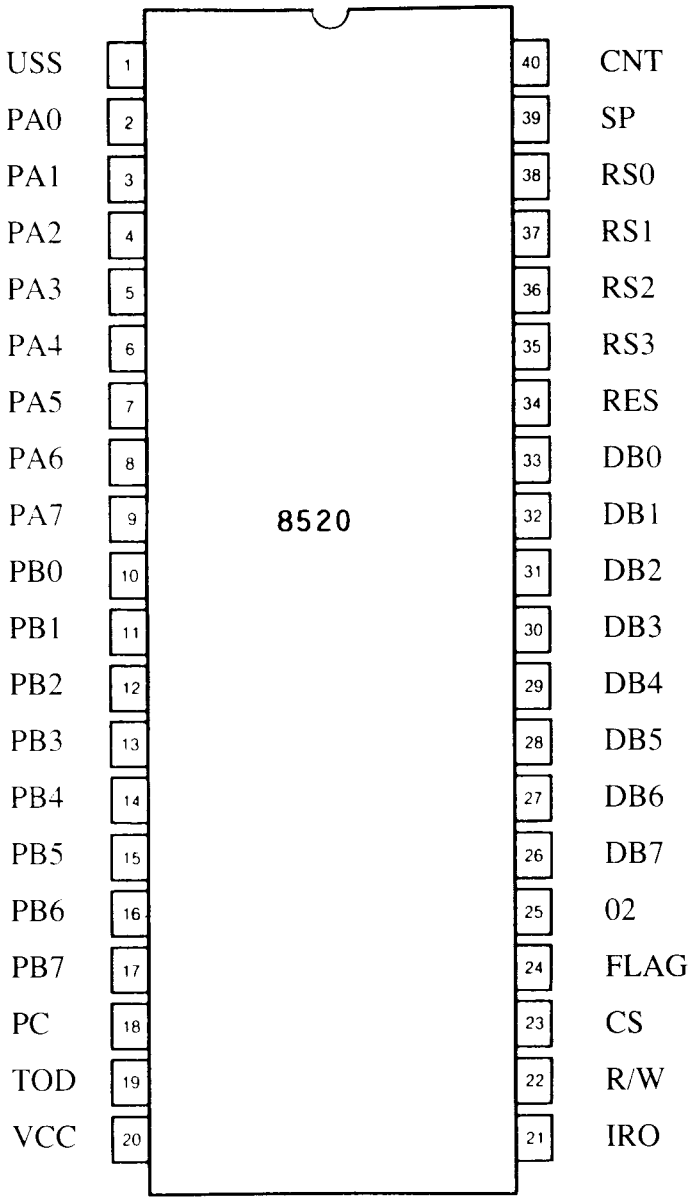


Figure 6.5. 8520 Pin Configuration

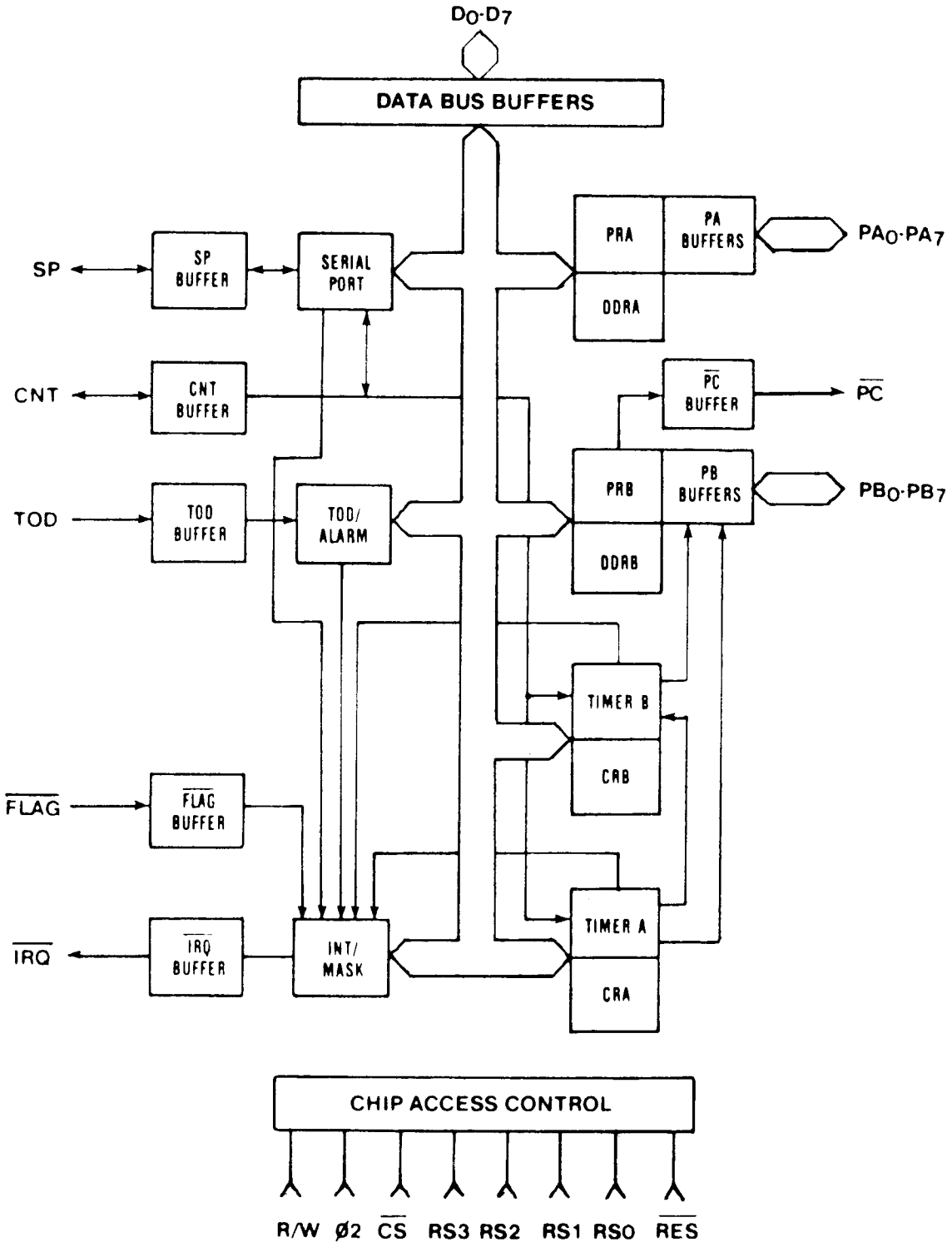
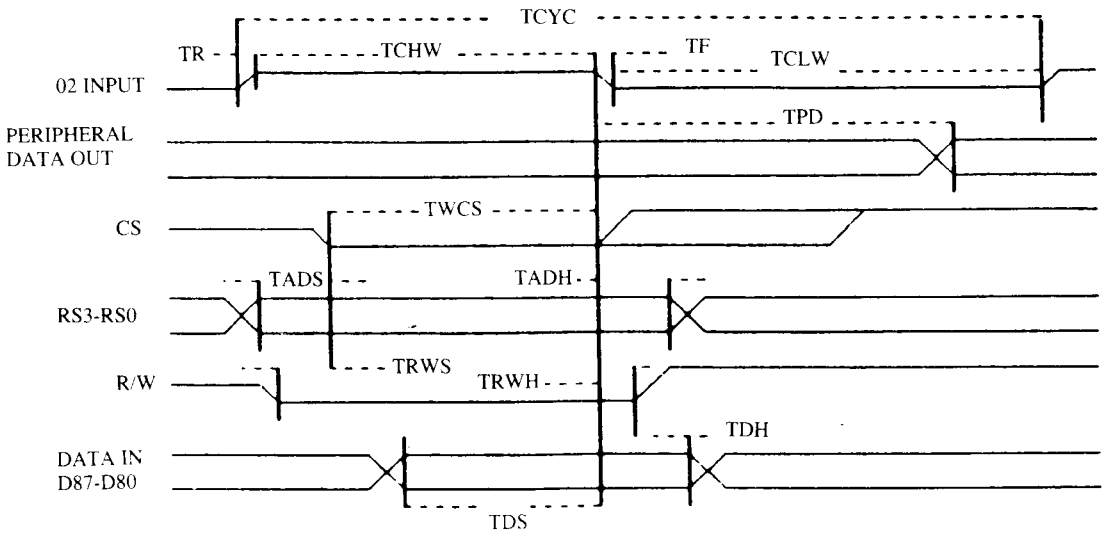


Figure 6.6. 8520 Block Diagram

WRITE TIMING DIAGRAM



READ TIMING DIAGRAM

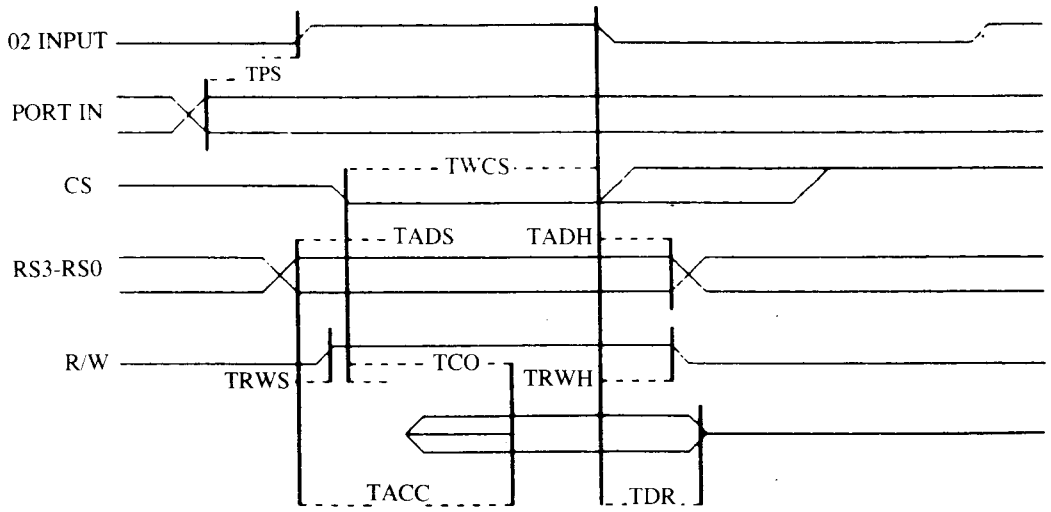


Figure 6.7. 8520 Timing Diagrams

INTERFACE SIGNALS

02-Clock Input

The 02 clock is a TTL compatible input used for internal device operation and as a timing reference for communicating with the system data bus.

CS-Chip Select Input

The CS input controls the activity of the 8520. A low level on CS while 02 is high causes the device to respond to signals on the R/W and address (RS) lines. A high on CS prevents these lines from controlling the 8520. The CS line is normally activated (low) at 02 by the appropriate address combination.

R/W-Read/Write Input

The R/W signal is normally supplied by the microprocessor and controls the direction of data transfers of the 8520. A high on R/W indicates a read (data transfer out of the 8520), while a low indicates a write (data transfer into the 8520).

RS3-RS0 — Address Inputs

The address inputs select the internal registers as described by the Register Map.

DB7-DB0 — Data Bus Inputs/Outputs

The eight bit data bus transfers information between the 8520 and the system data bus. These pins are high impedance inputs unless CS is low and R/W and 02 are high, to read the device. During this read, the data bus output buffers are enabled, driving the data from the selected register onto the system data bus.

IRQ-Interrupt Request Output

IRQ is an open drain output normally connected to the processor interrupt input. An external pullup resistor holds the signal high, allowing multiple IRQ-outputs to be connected together. The IRQ output is normally off (high impedance) and is activated low as indicated in the functional description.

RES-Reset Input

A low on the RES pin resets all internal registers. The port pins are set as inputs and port registers to zero (although a read of the ports will return all highs because of passive pullups). The timer control registers are set to zero and the timer latches to all ones. All other registers are reset to zero.

REGISTER MAP

RS3	RS2	RS1	RS0	REG		
0	0	0	0	0	PRA	Peripheral Data Reg. A
0	0	0	1	1	PRB	Peripheral Data Reg. B
0	0	1	0	2	DDRA	Data Direction Reg. A
0	0	1	1	3	DDRB	Data Direction Reg. B
0	1	0	0	4	TA LO	Timer A Low Register
0	1	0	1	5	TA HI	Timer A High Register
0	1	1	0	6	TB LO	Timer B Low Register
0	1	1	1	7	TB HI	Timer B High Register
1	0	0	0	8		Event LSB
1	0	0	1	9		Event 8-15
1	0	1	0	A		Event MSB
1	0	1	1	B		No Connect
1	1	0	0	C	SDR	Serial Data Register
1	1	0	1	D	ICR	Interrupt Control Register
1	1	1	0	E	CRA	Control Register A
1	1	1	1	F	CRB	Control Register B

FUNCTIONAL DESCRIPTION

I/O Ports (PRA, PRB, DDRA, DDRB)

Ports A and B each consist of an 8-bit Peripheral Data Register (PR) and an 8-bit Data Direction Register (DDR). If a bit in the DDR is set to the corresponding bit in the PR is an output if a DDR bit is set to zero, the corresponding PR bit is defined as an input. On a READ, the PR reflects the information present on the actual port pins (PA0-PA7, PB0-PB7) for both input and output bits. Port A has both passive and active pullup devices, providing both CMOS and TTL compatibility. It can drive 2 TTL loads. Port B has only passive pullup devices and has a much higher current-sinking capability.

Handshaking

Handshaking on data transfers can be accomplished using the PC output pin and the FLAG input pin. PC will go low on the 3rd cycle after a PORT B access. This signal can be used to indicate "data ready" at PORT B or "data accepted" from PORT B. Handshaking on a 16-bit data transfers (using both PORT A and PORT B) is possible by always reading or writing PORT A first. FLAG is a negative edge sensitive input which can be used for receiving the PC output from another 8520 or as a general purpose interrupt input. Any negative transition on FLAG will set the FLAG interrupt bit.

Reg	Name	D7	D6	D5	D4	D3	D2	D1	D0
0	PRA	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
1	PPB	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
2	DDRA	DPA7	DPA6	DPA5	DPA4	DPA3	DPA2	DPA1	DPA0
3	DDRB	DPB7	DPB6	DPB5	DPB4	DPB3	DPB2	DPB1	DPB0

Interval Timers (Timer A, Timer B)

Each interval timer consists of a 16-bit read-only Timer Counter and a 16-bit write-only Timer Latch. Data written to the timer are latched in the Timer Latch, while data read from the timer are the present contents of the Timer Counter. The timers can be used independently or linked for extended operations. The various timer modes allow generation of long time delays, variable width pulses, pulse trains and variable frequency waveforms. Utilizing the CNT input, the timers can count external pulses or measure frequency, pulse width and delay times of external signals. Each timer has an associated control register, providing independent control of the following functions:

Start/Stop

A control bit allows the timer to be started or stopped by the microprocessor at any time.

PB On/Off

A control bit allows the timer output to appear on a PORT B output line (PB6 for TIMER A and PB7 for TIMER B). This function overrides the DDRB control bit and forces the appropriate PB line to an output.

Toggle/Pulse

A control bit selects the output applied to PORT B. On every timer underflow the output can either toggle or generate a single positive pulse of one cycle duration. The toggle output is set high whenever the timer is started and is set low by RES.

One-Shot/Continuous

A control bit selects either timer mode. In one-shot mode, the timer will count down from the latched value to zero, generate an interrupt, reload the latched value, then stop. In continuous mode, the timer will count from the latched value to zero, generate an interrupt, reload the latched value and repeat the procedure continuously. In one-shot mode; a write to Timer High (registers 5 for TIMER A, 7 for TIMER B) will transfer the timer latch to the counter and initiate counting regardless of the start bit.

Force Load

A strobe bit allows the timer latch to be loaded into the timer counter at any time, whether the timer is running or not.

Input Mode

Control bits allow selection of the clock used to decrement the timer. TIMER A can count 02 pulses or external pulses applied to the CNT pin. TIMER B can count 02 pulses, external CNT pulses, TIMER A underflow pulses or TIMER A underflow pulses while the CNT pin is held high.

The timer latch is loaded into the timer on any timer underflow, on a force load or following a write to the high byte of the prescaler while the timer is stopped. If the timer is running, a write to the high byte will load the timer latch, but not reload the counter.

READ (TIMER)

REG	Name								
4	TA LO	TAL7	TAL6	TAL5	TAL4	TAL3	TAL2	TAL1	TAL0
5	TA HI	TAH7	TAH6	TAH5	TAH4	TAH3	TAH2	TAH1	TAH0
6	TB LO	TBL7	TBL6	TBL5	TBL4	TBL3	TBL2	TBL1	TBL0
7	TB HI	TBH7	TBH6	TBH5	TBH4	TBH3	TBH2	TBH1	TBH0

WRITE (PRESCALER)

REG	Name								
4	TA LO	PAL7	PAL6	PAL5	PAL4	PAL3	PAL2	PAL1	PAL0
5	TA HI	PAH7	PAH6	PAH5	PAH4	PAH3	PAH2	PAH1	PAH0
6	TB LO	PBL7	PBL6	PBL5	PBL4	PBL3	PBL2	PBL1	PBL0
7	TB HI	PBH7	PBH6	PBH5	PBH4	PBH3	PBH2	PBH1	PBH0

TOD

TOD consists of a 24 bit binary counter. Positive edge transitions on this pin cause the binary to increment. The TOD pin has a passive pull-up on it. A programmable ALARM is provided for generating an interrupt at a desired time. The ALARM registers are located at the same addresses as the corresponding TOD register. Access to the ALARM is governed by a Control Register bit. The ALARM is write-only; any read of a TOD address will read time regardless of the state of the ALARM access bit.

A specific sequence of events must be followed for proper setting and reading of TOD. TOD is automatically stopped whenever a write to the register occurs. The clock will not start again until after a write to the LSB Event Register. This assures TOD will always start at the desired time. Since a carry from one stage to the next can occur at any time with respect to a read operation, a latching function is included to keep all Time of Day information constant during a

read sequence. All TOD registers latch on a read of MSB event and remain latched until after a read of LSB Event. The TOD clock continues to count when the output registers are latched. If only one register is to be read, there is no carry problem and the register can be read "on the fly", provided that any read of MSB Event is followed by a read of LSB Event to disable the latching.

READ

REG	NAME								
8	LSB EVENT	E7	E6	E5	E4	E3	E2	E1	E0
9	EVENT 8-15	E15	E14	E13	E12	E11	E10	E9	E8
A	MSB EVENT	E23	E22	E21	E20	E19	E18	E17	E16

WRITE

CRB7 = 0
CRB7 = 1 ALARM
(SAME FORMAT AS READ)

Serial Port (SDR)

The serial port is a buffered, 8-bit synchronous shift register system. A control bit selects input or output mode. In input mode, data on the SP pin is shifted into the shift register on the rising edge of the signal applied to the CNT pin. After 8 CNT pulses, the data in the shift register is dumped into the Serial Data Register and an interrupt is generated. In the output mode, TIMER A is used for the baud rate generator. Data is shifted out on the SP pin at ½ the underflow rate of TIMER A. The maximum baud rate possible is 02 divided by 6, but the maximum useable baud rate will be determined by line loading and the speed at which the receiver responds to input data. Transmission will start following a write to the Serial Data Register (provided TIMER A is running and in continuous mode). The clock signal derived from TIMER A appears as an output on the CNT pin. The data in the Serial Data Register will be loaded into the shift register then shift out to the SP pin when a CNT pulse occurs. Data shifted out becomes valid on the falling edge of CNT and remains valid until the next falling edge. After 8 CNT pulses, an interrupt is generated to indicate more data can be sent. If the Serial Data Register was loaded with new information prior to this interrupt, the new data will automatically be loaded into the shift register and transmission will be continuous. If no further data is to be transmitted, after the 8th CNT pulse, CNT will return high and SP will remain at the level of the last data bit transmitted. SDR data is shifted out MSB first and serial input data should also appear in this format.

The bidirectional capability of the Serial Port and CNT clock allows several devices to be connected to a common serial communication bus on which one acts as a master, sourcing data and shift clock, while all other chips act as slaves. Both CNT and SP outputs are open

drain, with passive pullups, to allow such a common bus. Protocol for slave/master selection can be transmitted over the serial bus, or via dedicated handshaking lines.

REG	NAME								
C	SDR	S7	S6	S5	S4	S3	S2	S1	S0

Interrupt Control (ICR)

There are five sources of interrupts on the 8520: underflow from TIMER A, underflow from TIMER B, TOD ALARM, Serial Port full/empty and FLAG. A single register provides masking and interrupt information. The Interrupt Control Register consists of a write-only MASK register and a read-only DATA register. Any interrupt which is enabled by the MASK register will set the IR bit (MSB) of the DATA register and bring the IRQ pin low. In a multi-chip system, the IR bit can be polled to detect which chip has generated an interrupt request.

The interrupt DATA register is cleared and the IRQ line returns high following a read of the DATA register. Since each interrupt sets an interrupt bit regardless of the MASK, and each interrupt bit can be selectively masked to prevent the generation of a processor interrupt, it is possible to intermix polled interrupts with true interrupts. However, polling the IR bit will cause the DATA register to clear, therefore, it is up to the user to preserve the information contained in the DATA register if any polled interrupts were present.

The MASK register provides convenient control of individual mask bits. When writing to the MASK register, if bit 7 (SET/CLEAR) of the data written is a ZERO, any mask bit written with a one will be cleared, while those mask bits written with a zero will be unaffected. If bit 7 of the data written is a ONE, any mask bit written with a one will be set, while those mask bits written with a zero will be unaffected. In order for an interrupt flag to set IR and generate an Interrupt Request, corresponding MASK bit must be set.

READ (INT DATA)

REG	NAME								
D	IRA	IR	0	0	FLG	SP	ALRM	TB	TA

WRITE (INT MASK)

REG	NAME								
D	IRC	S/C	X	X	FLG	SP	ALRM	TB	TA

Control Registers

There are two control registers in the 8520: CRA and CRB. CRA is associated with TIMER A and CRB is associated with TIMER B.

The register format is as follows:

CRA:

BIT	NAME	FUNCTION
0	START	1 = START TIMER A. 0 = STOP TIMER A. This bit is automatically reset when underflow occurs during one-shot mode.
1	PBON	1 = TIMER A output appears on PB6, 0 = PB6 normal operation
2	OUTMODE	1 = TOGGLE, 0 = PULSE
3	RUNMODE	1 = ONE-SHOT, 0 = CONTINUOUS
4	LOAD	1 = FORCE LOAD (this is a STROBE input, there is no data storage, bit 4 will always read back a zero and writing a zero has no effect.
5	INMODE	1 = TIMER A counts positive CNT transitions, 0 = TIMER A counts 02 pulses.
6	SPMODE	1 = SERIAL PORT output (CNT sources shift clock). 0 = SERIAL PORT input (external shift clock required).
7	TODIN	1 = 50 Hz clock required on TOD pin for accurate time. 0 = 60 Hz clock required on TOD pin for accurate time.

CRB:

BIT	NAME	FUNCTION
		(Bits CRB0-CRB4 are identical to CRA0-CRA4 for TIMER B with the exception that bit 1 controls the output of TIMER B on PB7).
5,6	INMODE	Bits CRB5 and CRB6 select one of four input modes for TIMER B as:

CRB6	CRB5	
0	0	TIMER B counts 02 pulses
0	1	TIMER B counts positive CNT transitions
1	0	TIMER B counts TIMER A underflow pulses
1	1	TIMER B counts TIMER A underflow pulses while CNT is high

7	ALARM	1 = writing to TOD registers set ALARM. 0 = writing to TOD registers sets TOD clock.

Clock/Calendar Information

The clock/calendar is based on the OKI MSM6242RS Direct Bus Connected-Type Real Time Clock Chip.

The A2000 features a real time clock with a perpetual calendar which is capable of reading and writing "YEAR", "MONTH", "DAY", "WEEK", "HOUR", "MINUTE" and "SECOND". This time clock is a peripheral IC, connected directly by means of a bus. It is standard on the A2000, and can be added as an option to the A500 (included in the A501 Memory Expander).

An interface between the time clock and a microcomputer uses 4 of data bus lines, 4 address bus lines, 3 control bus lines and 2 chip select pins, and performs time setting, reading and other operations.

The clock function covers second, minute, hour, day, month, year and day of week. In addition, other functions such as selection of a 24-hour time and a 12-hour time system, automatic adjustment of leap year in the Christian Era and 30-second correction by means of soft, periodical interruption (or periodical wave-form output) and stop/start of time counting.

The clock-calendar is a CMOS device, so there is low power consumption.

A crystal used is capable of 32.768 KHz for a consideration over time counting during battery backup.

When installed, the clock is located in memory at $DC0000.

Clock Warning
The addresses used by the real time clock chip access the custom chip registers without the memory expansion/real time clock module. When probing to test for the existence of the clock, care must be taken to avoid unintentional changes to the custom chip register. The test used by the setclock utility references an address that maps to either the seconds register or a static read-only chip register, then checks to see if the clock "ticks."

Note: C912 can be used as a slow/fast control to tune the clock to best effect.

REGISTER TABLE

Address	A_3	A_2	A_1	A_0	Number of Register	Data				Count value	Description
						D_3	D_2	D_1	D_0		
0	0	0	0	0	S_1	S_8	S_4	S_2	S_1	0 ~ 9	1-second digit register
1	0	0	0	1	S_{10}	*	S_{40}	S_{20}	S_{10}	0 ~ 5	10-second digit register
2	0	0	1	0	MI_1	mi_8	mi_4	mi_2	mi_1	0 ~ 9	1-minute digit register
3	0	0	1	1	MI_{10}	*	mi_{40}	mi_{20}	mi_{10}	0 ~ 5	10-minute digit register
4	0	1	0	0	H_1	h_8	h_4	h_2	h_1	0 ~ 9	1-hour digit register
5	0	1	0	1	H_{10}	*	PM/AM	h_{20}	h_{10}	0 ~ 2 or 0 to 1	PM/AM, 10-hour digit register
6	0	1	1	0	D_1	d_8	d_4	d_2	d_1	0 ~ 9	1-day digit register
7	0	1	1	1	D_{10}	*	*	d_{20}	d_{10}	0 ~ 3	10-day digit register
8	1	0	0	0	MO_1	mo_8	mo_4	mo_2	mo_1	0 ~ 9	1-month digit register
9	1	0	0	1	MO_{10}	*	*	*	mo_{10}	0 ~ 1	10-month digit register
A	1	0	1	0	Y_1	y_8	y_4	y_2	y_1	0 ~ 9	1-year digit register
B	1	0	1	1	Y_{10}	y_{80}	y_{40}	y_{20}	y_{10}	0 ~ 9	10-year digit register
C	1	1	0	0	W	*	w	w	w	0 ~ 6	Week register
D	1	1	0	1	C_D	30 sec. ADJ	IRQ FLAG	BUSY	HOLD	—	Control Register D
E	1	1	1	0	C_E	t_1	t_0	ITRPT /STND	HASK	—	Control Register E
F	1	1	1	1	C_F	TEST	24/ 12	STOP	REST	—	Control Register F

- 0–low level 1–high level
- REST–RESET
- ITRPT/STND–INTERRUPT/STANDARD

Note 1: You have the option to write data into the bit*. However, this data is treated as 0 internally. In addition, the bit* is always read as 0.

Note 2: You can write 1 into the IRQ FLAG bit and 0 or 1 into the BUSY bit. They are not executed, but can be read.

Note 3: It is possible to read and write all bits other than bit* and BUSY bit. However, only 0 can be written into IRQ FLAG.

More information on the clock/calendar may be found in the OKI MSM6242RS Direct Bus Connected-Type Real Time Clock app. notes.

Power Budgets

B2000 POWER BUDGET

A2000/B2000 POWER SUPPLY:

All of the specifications herein are suggested. When it comes right down to it, the machine is being powered by a well-defined supply, the specifications of which will follow. If you're careful not to exceed the suggested load for any port, you'll be able to fully load every port. However, some of the internal ports can draw more than the suggested amount; for example, an 8 megabyte expansion memory card for the 100 pin bus may draw more than the suggested 2.5 Amps at +5VDC. The connector is capable of supplying more without damage, but the extra current must be carefully worked into the system power budget. Any hardware add-on device that draws more than the suggested amount must state this clearly. External ports typically have a true maximum available, not a suggested; budgeting should apply to internal items only.

VOLTAGE	SYSTEMWIDE LIMIT	DESCRIPTION
+ 5 VDC	20.0 Amps	Main +5 Voltage supply
+ 5 USER	0.5 Amps	Protected +5 for externals
− 5 VDC	0.3 Amps	Negative 5 Volt supply
+12 VDC	8.0 Amps	Main +12 Voltage supply
+12 USER	---------	Protected +12 for externals, derived from main +12
−12 VDC	0.3 Amps	Main −12 Voltage supply
−12 USER	---------	Protected −12 for externals, derived from main −12

CONSUMPTION:

Everybody wants power. Here's what can be taken, based on your particular setup; you get what's left over:

MAIN SYSTEM:	+5VDC	−5VDC	+5USER	+12VDC	−12VDC
Motherboard	2.5A	---	---	50mA	50mA
Internal 3½" Floppy [1]	250mA	---	---	350mA	---
Internal 5¼" Floppy [2]	500mA	---	---	500mA	---
Internal 3½" Hard Disk [2]	750mA	---	---	1.0A	---
Internal 5¼" Hard Disk [2]	1.0A	---	---	1.5A	---

EXTERNAL PORTS:

Video Port	---	10mA	100mA	100mA	---
Floppy Port [1]	250mA	---	---	350mA	---
Parallel Port [3]	---	---	10mA	---	---
Serial Port	---	---	---	25mA	25mA
Keyboard Port [4]	250mA	---	---	---	---
Mouse Port	---	---	50mA	---	---

INTERNAL SLOTS:

CoProcessor Slot [6]	2.0A	40mA	---	40mA	35mA
Expansion Slot [6]	2.0A	40mA	---	40mA	35mA
Extra PC Bus Slots [7]	0.5A	10mA	---	40mA	15mA
Video Slot [8]	1.0A	40mA	---	40mA	---

NOTES:

[1] Expected typical consumption. This is very device dependent; consult the manufacturer's specification for particular floppy disks. The starting current is expected to be around 400mA for +12V.

[2] Expected typical consumption. This is very device dependent; consult the manufacturer's specification for particular disks. Starting current on the +12V supply for most disk drives can be as much as twice the operating current.

[3] 47 Ohm Series Resistor limits current.

[4] Expected typical consumption. Current from this port is limited.

[5] Each port.

[6] Each slot. The physical connection can handle 4 Amps; if a 4 Amp device is used in one slot, other slots cannot supply 2.5 Amps each, of course; this requires a total system power budget to be constructed.

[7] Shared PC expansion slots should be considered part of the 100 pin connector that they share. If the 100 pin connector is unused, the power suggested for that connector can be used instead of the PC bus. The connectors, like all expansion connectors, are capable of delivering 4 Amps if proper whole-system budgeting is done. The specification here is for both of the non-overlapping PC slots taken together.

[8] Like expansion slots the video slot is capable of supplying 4. If a 4 Amp device is used, it must be worked into the total system power budget.

A500 Power Budget

PARALLEL PORT:

10mA from pin 14 (+5V)
(47Ω series resistor to prevent damage if printer grounds this line)

SERIAL PORT:

20mA from pin 9 (+12V)
20mA from pin 10 (−12V)
(47Ω series resistor to limit current)

VIDEO PORT:

100mA from pin 23 (+5V)	No
100mA from pin 22 (+12V)	current
10mA from pin 21 (−12V)	limiting

JOYSTICK PORTS (TOTAL)

50mA from pins 7 (+5V)
(4.7Ω current limit)

EXPANSION PORT:

300mA from pins 5 and 6 (+5V)	No
50mA from pin 10 (+12V)	current
10mA from pin 8 (−12V)	limiting

A2000 PAL Equations

```
PAL20L8                                                PAL DESIGN SPECIFICATION
PART NO.: 380 XXX-01  DESCRPT.:PALEN     REV.2    FRANK ULLMANN   03-09-86
MEM- AND DTACK-DECODER FOR A2500 MAINBOARD (U26) ASSY 380...
COMMODORE BSW          !! PRELIMENARY !!

A23 A22 A21 A20 A19 A18 PRW AS DBR OVL OVR GND
C1 C3 VPA MYRAME CLKE RGAE RE DTACK BLS ROME XRDY VCC


IF (OVR) /VPA = /AS*A23*/A22*A21                       ; PERIPHERAL ACCESS
                                                       ; $A00000-BFFFFF

/MYRAME  = /AS*DTACK*A23*A22*A21*OVR*/C1*C3            ; $E00000-FFFFFF
         + /AS*DTACK*/A23*/A22*/A21*OVR*OVL*/C1*C3     ; $000000-1FFFFF IF
                                                       ; OVL=H, OVR=H !
         + /AS*DTACK*A23*A22*/A21*A20*A19*/A18*OVR*/C1*C3   ; $D80000-DBFFFF
         + /MYRAME*/C1
         + /MYRAME*/C3

/RE      = DBR*/AS*DTACK*/A23*/A22*/A21*OVR*/OVL*      ; $000000-1FFFFF IF
           /C1*C3                                      ; OVL=L, OVR=H !
         + /RE*/C1
         + /RE*/C3

IF (OVR) /DTACK =
           /AS*/A23*/A22*A21*XRDY                      ; $200000-3FFFFF EXP RAM
         + /AS*/A23*A22*XRDY                           ; $400000-7FFFFF  "   "
         + /AS*A23*/A22*/A21*XRDY                      ; $800000-9FFFFF  "   "
         + /MYRAME*XRDY*/C3                            ; $000000-1FFFFF OVL=H
                                                       ; AND $E00000-FFFFFF
         + /RE*/C3                                     ; $000000-1FFFFF OVL=L
         + /RGAE*/C3                                   ; $C00000-D7FFFF
                                                       ; AND $DC0000-DF0000
         + /DTACK*/AS*XRDY

/RGAE    = DBR*/AS*DTACK*A23*A22*/A21*A20*/A19*   OVR*/C1*C3;$D00000-$D7FFFF
         + DBR*/AS*DTACK*A23*A22*/A21*A20*A19*A18*OVR*/C1*C3;$DC0000-$DFFFFF
         + DBR*/AS*DTACK*A23*A22*/A21*/A20*       OVR*/C1*C3;$C00000-$CFFFFF
         + /RGAE*/C1
         + /RGAE*/C3

/BLS     = /AS*DTACK*/A23*/A22*/A21*OVR*/OVL*/C1*C3 ; $000000-1FFFFF OVL=L
         + /AS*DTACK*A23*A22*/A21*OVR*/C1*C3        ; $C00000-DFFFFF
         + /BLS*/C1
         + /BLS*/C3
```

```
/ROME    = /AS*A23*A22*A21*A20*A19*OVR*PRW          ; $F80000-FFFFFF
         + /AS*A23*A22*A21*/A20*/A19*OVR*PRW        ; $E00000-E7FFFF
         + /AS*/A23*/A22*/A21*/A20*/A19*OVR*OVL*PRW ; $000000-07FFFF
         + /AS*/A23*/A22*/A21*A20*A19*OVL*OVR*PRW   ; $180000-1FFFFF

/CLKE    = /AS*A23*A22*/A21*A20*A19*/A18*OVR        ; $D80000-DBFFFF
```

DESCRIPTION

THE CLOCK IS NOW TILED THROUGHOUT THE SPACE $D80000-DBFFFF. IF MORE PRECISE SELECTION TO $D8000[illegible] $D8FFFF IS REALLY NEEDED, THEN THIS MUST BE DONE EXTERNALLY USING THE /CS INPUT ON THE CLOCK CHIP.

DTACK FOR THE CLOCK IS HANDLED IN THE MYRAME EQUATION BECAUSE THE DTACK EQUATION ALREADY HAS 7 OR TERMS! THIS MEANS THAT CLOCK ACESSES WILL BE SYNCHRONIZED TO VIDEO CYCLES, BUT THIS SHOULD CREATE NO MAJOR PROBLEMS.

THE IMPLEMENTATION OF ROME/MYRAME MATCHES THE A1000 - IT MIGHT BE DESIRABLE TO HAVE THE ADDRESS RANGE IN MYRAME MATCH THE ROM SELECT ADDRESS RANGE...

MYRAME OUTPUT IS NOT USED EXTERNALLY - ONLY INTERNAL USAGE!!!

```
PAL16L8                                                PAL DESIGN SPECIFICATION
PART NO.: 380 XXX-01  DESCRPT.:PALCAS    REV.1    FRANK ULLMANN   08-29-86
RAM/ROM-DECODER FOR A2500 MAINBOARD REV.2 (U27) ASSY 380...
COMMODORE BSW         !! PRELIMENARY !!

ARW C1 C3 PRW UDS LDS RE RGAE CLKE GND
DBR CLKR RRW LCEN UCEN CDR CDW DAE CLKW  VCC


/CDR    = /RE*PRW*C1                                  ; ENABLE RAM READ
        + /RGAE*PRW*C1                                ; BUFFER
        + /CDR*/LDS
        + /CDR*/UDS

/CDW    = /RE*/PRW                                    ; ENABLE RAM WRITE
        + /RGAE*/PRW                                  ; BUFFER
        + /CDW*C1

/UCEN   = /DAE*RE*C1                                   ; GENERATE CAS
        + DAE*/RE*/UDS*C1                              ; SIGNALS
        + /UCEN*C1

/LCEN   = /DAE*RE*C1                                   ; GENERATE CAS
        + DAE*/RE*/LDS*C1                              ; SIGNALS
        + /LCEN*C1

/RRW    = /RE*/PRW                                    ; /WE FOR DRAMS CPU
        + /DAE*/ARW*/C1                               ; OR AGNUS ACCESS
        + /RRW*/DAE

/DAE    = /DBR*/C1*C3                                 ; CHIP RAM ADDR ENABLE
        + /DAE*/C1
        + /DAE*/C3

/CLKR   = /CLKE*/LDS*PRW                              ; CLOCK READ

/CLKW   = /CLKE*/LDS*/PRW                             ; CLOCK WRITE

DESCRIPTION
```

PAL20L8 PAL DESIGN SPECIFICATION

PART NO.: 380715-2 DESCRPT.: PAL BUFFER CONTROL REV.2 HEINZ ULLRICH 06-18-87

PAL BUFFER CONTROL FOR A2500 (U5) FOR PRODUCTION-PCB

COMMODORE BSW

SLV1 SLV2 SLV3 SLV4 SLV5 OVR RD BAS RESET A23 A22 GND

A21 A20 D2P A19 BERR OWN DS NCOLLIS PROC DBOE ASQ VCC

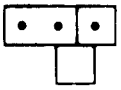
```
/NCOLLIS  =      SLV1*SLV2*SLV3*SLV4*SLV5
          + PROC*     SLV2*SLV3*SLV4*SLV5
          + PROC*SLV1*     SLV3*SLV4*SLV5
          + PROC*SLV1*SLV2*     SLV4*SLV5
          + PROC*SLV1*SLV2*SLV3*     SLV5
          + PROC*SLV1*SLV2*SLV3*SLV4

/PROC     = /BAS*/A23*/A22*/A21*          RESET*OVR
          + /BAS* A23*/A22* A21*          RESET*OVR
          + /BAS* A23* A22*/A21*          RESET*OVR
          + /BAS* A23* A22* A21*/A20*/A19*RESET*OVR
          + /BAS* A23* A22* A21* A20* A19*RESET*OVR

/D2P      = OWN*/SLV1*RD            ; DOWNSTREAM READS UPSTREAM SLAVE
          + OWN*/SLV2*RD            ;          -"-
          + OWN*/SLV3*RD            ;          -"-
          + OWN*/SLV4*RD            ;          -"-
          + OWN*/SLV5*RD            ;          -"-
          + /OWN*SLV1*SLV2*SLV3*SLV4*SLV5*/RD ; UPSTREAM WRITES DOWNSTREAM SL

/DBOE     = /BAS*/RD    *BERR*OWN      ; CPU READS FROM SLAVE
          + /D5*RD*/ASQ*BERR*OWN       ; CPU WRITES TO SLAVE
          + /BAS*/RD    *BERR*/OWN*SLV1*SLV2*SLV3*SLV4*SLV5; DMA READS CPU RAM
          + /DS*RD*/ASQ*BERR*/OWN*SLV1*SLV2*SLV3*SLV4*SLV5; DMA WRITES TO CPU RAM

IF (RESET*NCOLLIS) /BERR = RESET

DESCRIPTION
```

```
PAL16R6                                                PAL DESIGN SPECIFICATION
PART NO.:          DESCRPT.:PAL ARBITRATE A2500 REV.1 FRANK ULLMANN 05-22-86
PAL ARBITRATE FOR A2500 (U  ) FOR PRODUCTION-PCB
COMMODORE BSW

7M BAS RES BGIN BR5 BR4 BR3 BR2 BR1 GND
NC BASD BGOLD BG5 BG4 BG3 BG2 BG1 BR VCC


/BG1    = RES*/BGIN*BGOLD*/BR1          ; GENERATE BG1
        + RES*/BGIN*/BG1                ; HOLD UNTIL /BG

/BG2    = RES*/BGIN*BGOLD*/BR2*BR1
        + RES*/BGIN*/BG2

/BG3    = RES*/BGIN*BGOLD*/BR3*BR1*BR2
        + RES*/BGIN*/BG3

/BG4    = RES*/BGIN*BGOLD*/BR4*BR1*BR2*BR3
        + RES*/BGIN*/BG4

/BG5    = RES*/BGIN*BGOLD*/BR5*BR1*BR2*BR3*BR4
        + RES*/BGIN*/BG5

/BGOLD  = /BGIN                         ; STORE OLD STATE OF BG

/BR     = RES*/BR5                      ; BR IS REQUEST TO 68K
        + RES*/BR1
        + RES*/BR2
        + RES*/BR3
        + RES*/BR4

/BASD   = BAS


DESCRIPTION

BG1 IS HIGHEST PRIORITY
```

List of B2000 Motherboard Jumpers

J101

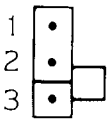
3 2 1



This jumper determines the high-order address bit for Fat Agnus. In its normal position shown the high-order bit is A23; in its other position, this bit is A19. The current Fat Agnus chip requires the A23 signal for proper management of the memory at $C00000. Future Fat Agnus chips may map things differently.

J200

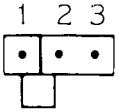
1
2
3



This jumper is used to set the light-pen port number. In the normal position shown, the light pen input will be the FIRE input of mouse/joystick port 1, as with the A500. With the jumper in the other position, the light pen input will be the FIRE input of mouse/joystick port 0, which is the scheme used on the A1000 machine.

J300

1 2 3



This jumper determines the time base used for the 50/60Hz CIA timer chip. In the normal position, the 50/60Hz TICK clock, based on AC line frequency, is used as a time base. In the alternate position, the vertical sync pulse from the video section is used. The system will not operate properly without one of these clocks.

J301

X X

This jumper is closed to add a second internal floppy drive, open to leave the second floppy out of the main unit box.

J500

X-X

This jumper is used to enable the 512K of RAM at $C00000. It is normally closed; opening it will disable this extra RAM.

Appendix A

Diagrams

CONTENTS

This appendix contains the following figures:

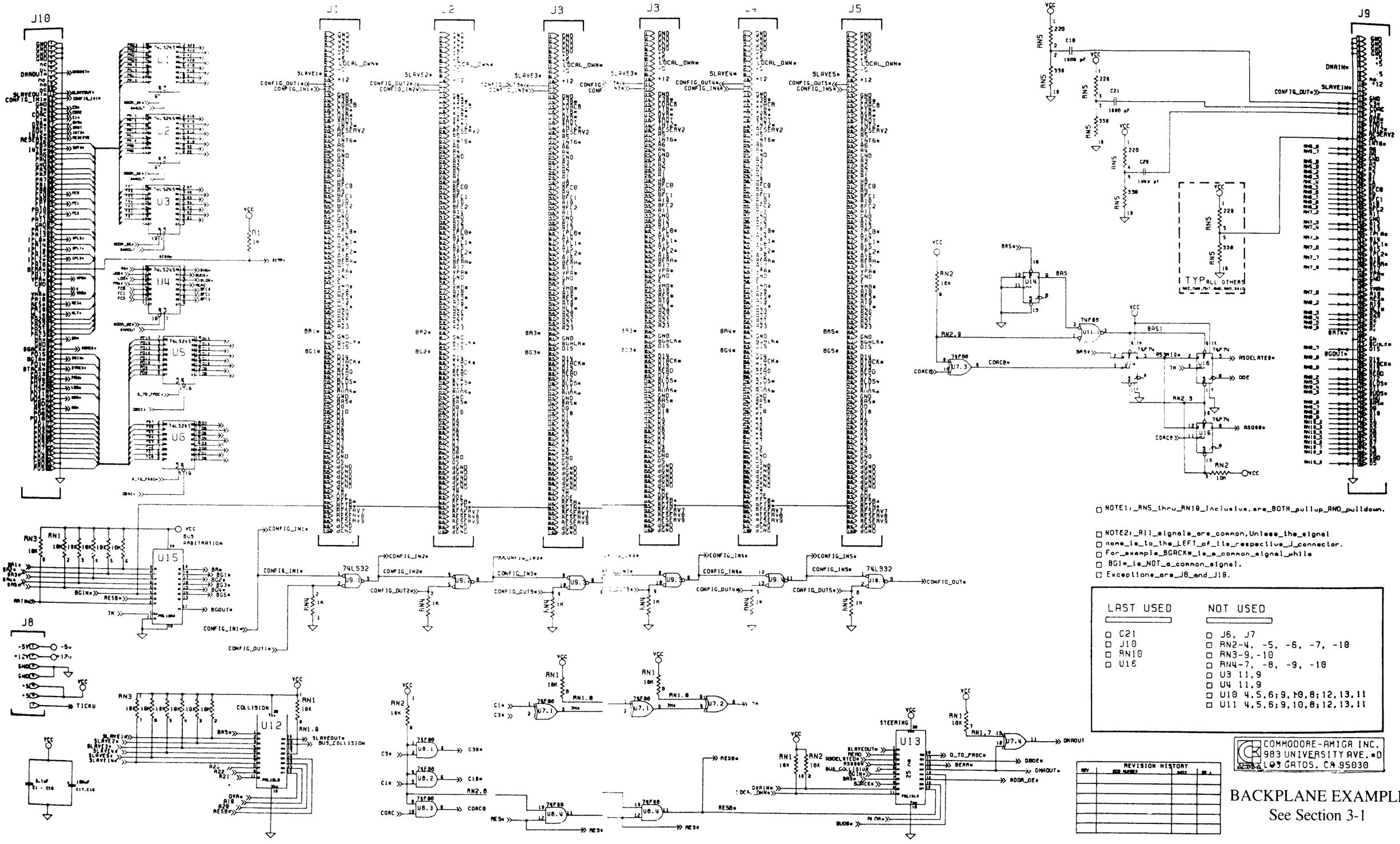
The example backplane (discussed in Section 3.1)	A-1
The example PIC (discussed in Section 3.1)	A-2
A500 exterior, featuring the 86-pin expansion connector	A-3
Amiga 2000 expansion board layout	A-4
Amiga 2000 form factor (including 100-pin connector)	A-5
Amiga 2000 video card	A-6
86-Pin slot expansion board	A-7
A2000/B2000 keyboard connector pinout	A-8
Amiga 500/2000 mouse diagram and pinout	A-9

Appendix B

Schematics

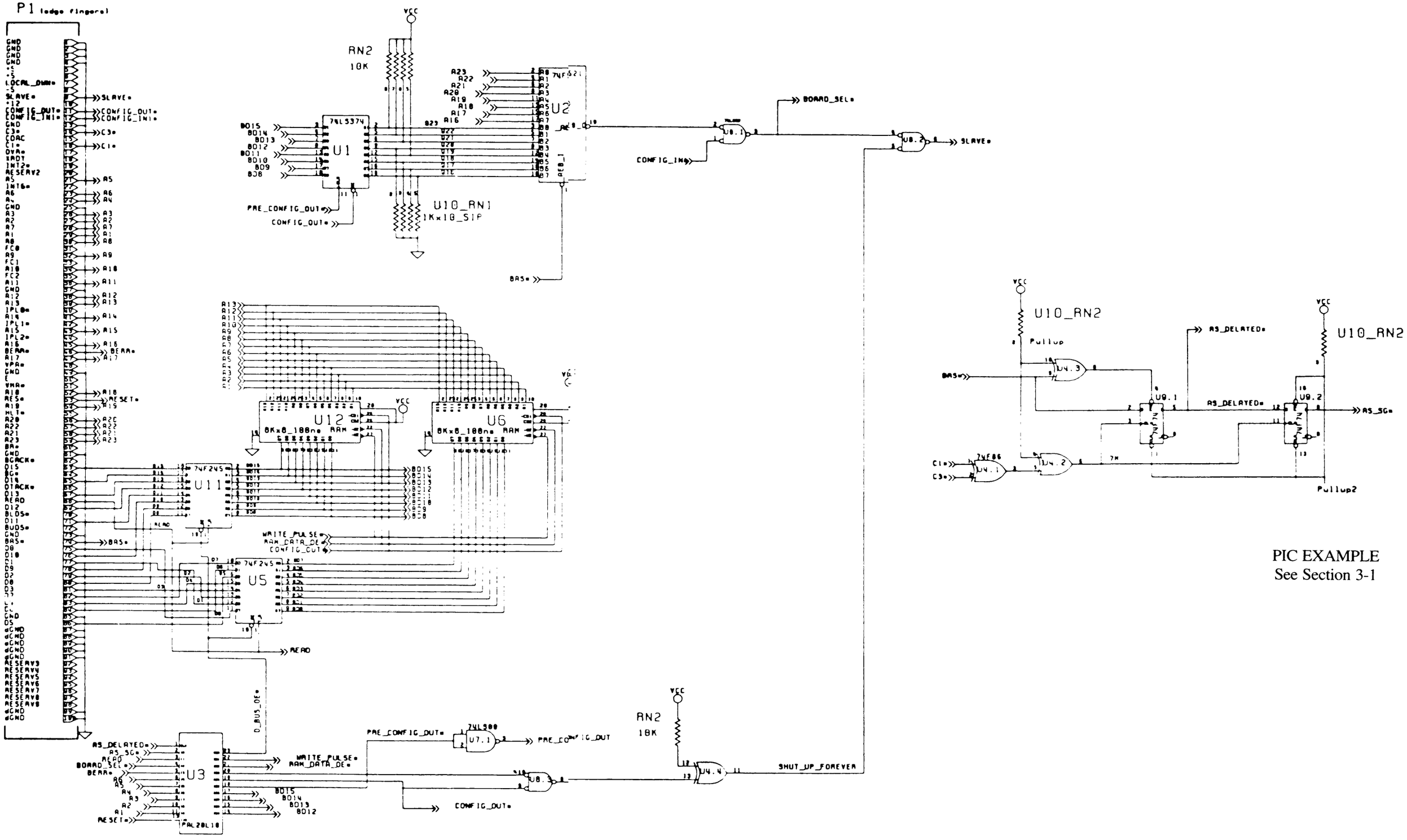
CONTENTS

This appendix contains schematics for each new model Amiga. Note that these schematics are representative of the engineering design, but may not reflect the current production board design in all details.



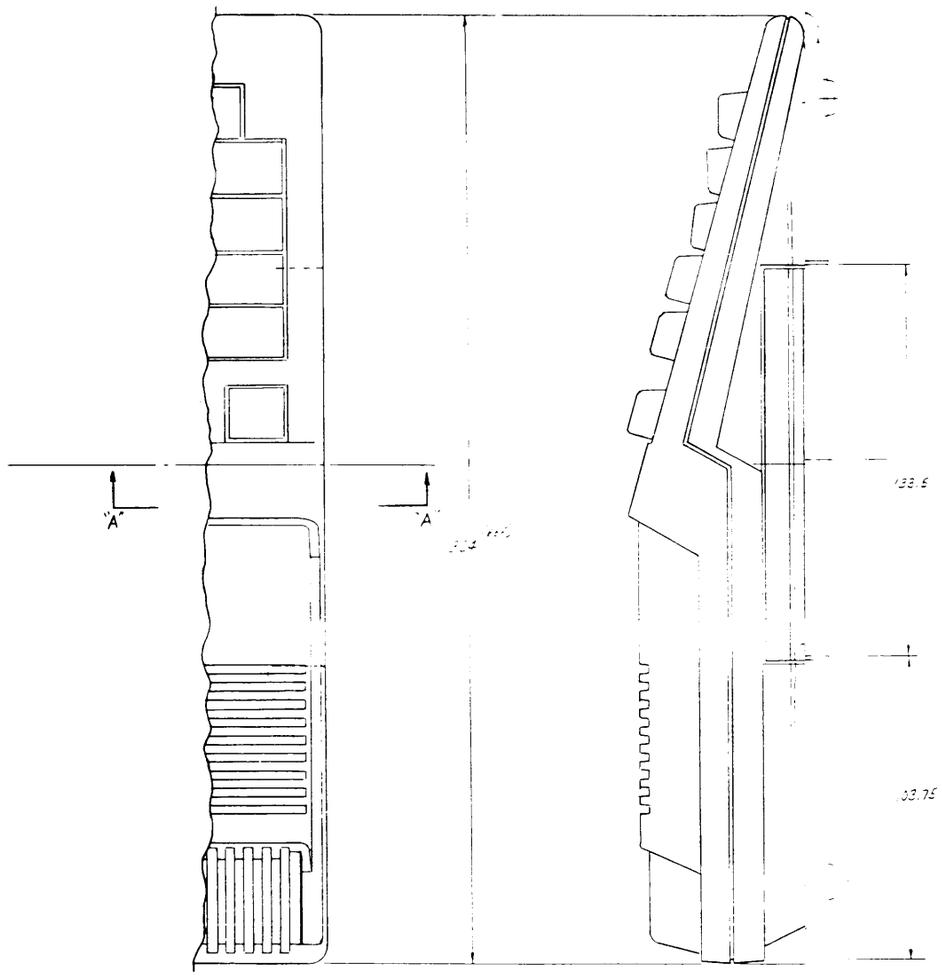
BACKPLANE EXAMPLE

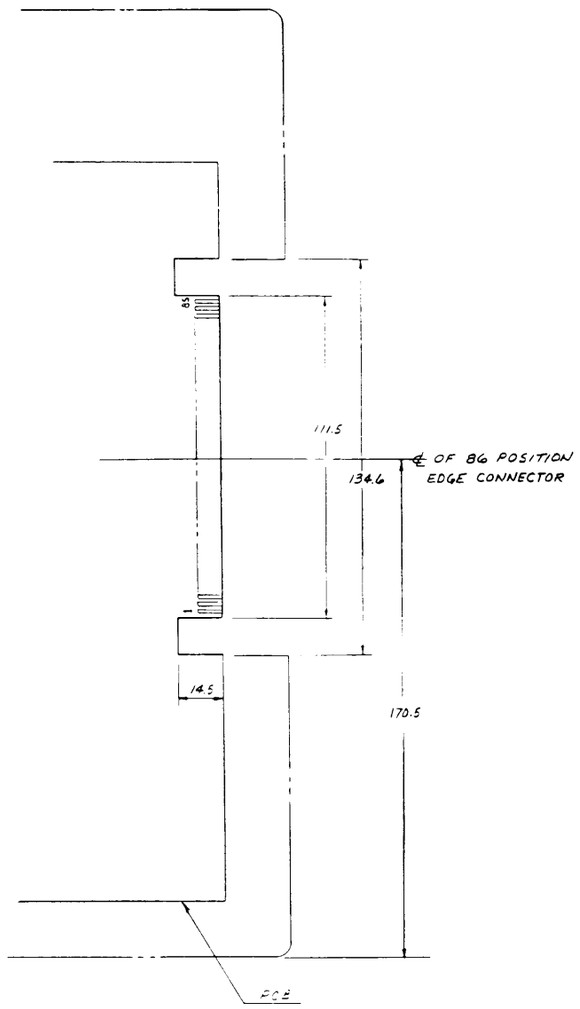
See Section 3-1

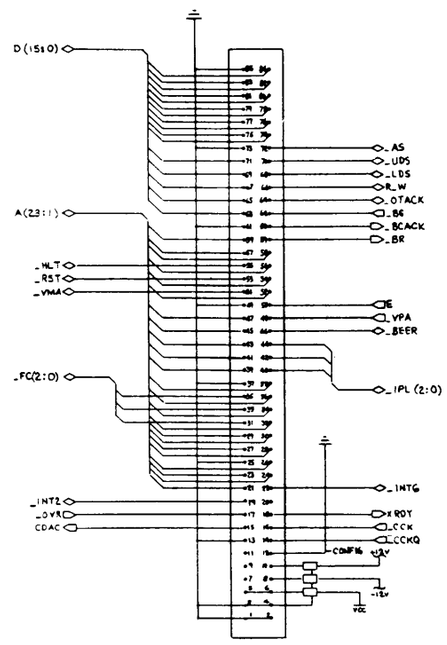


PIC EXAMPLE
See Section 3-1

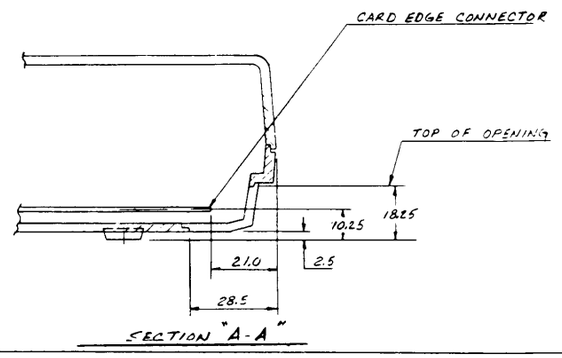
REVISIONS				
ZONE	LTR	DESCRIPTION	DATE	APPROVED
	1	PILOT PRODUCTION RELEASE	4-23-87	[illegible]







EXPANSION SCHEMATIC

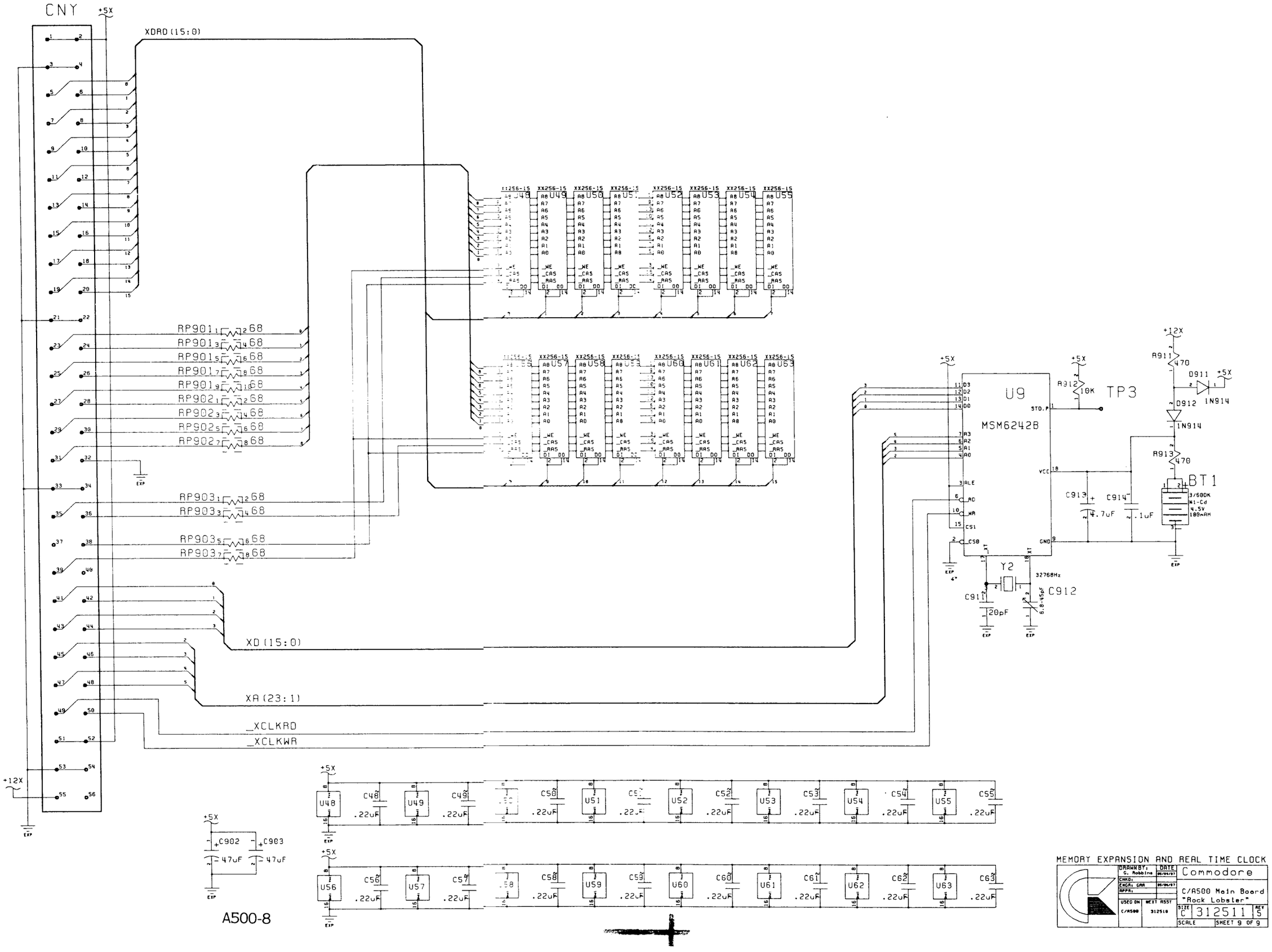


SECTION "A-A"

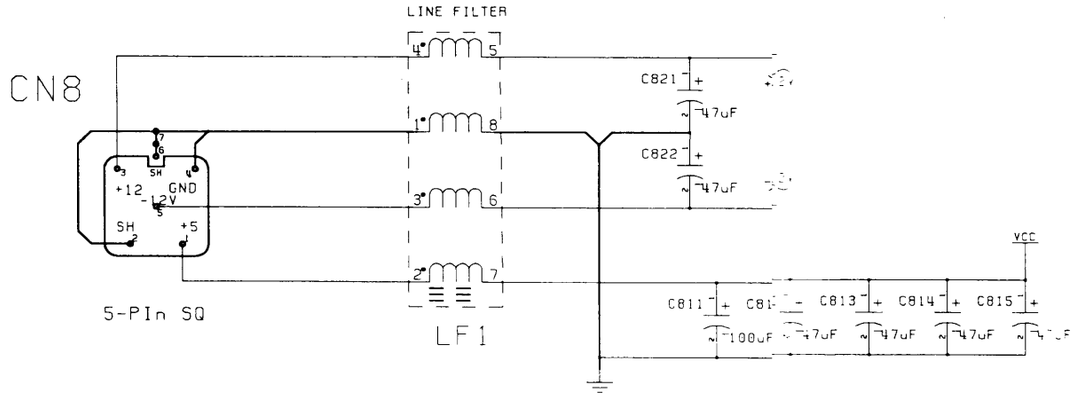
NOTE: UNLESS OTHERWISE SPECIFIED
ALL DIMENSIONS ARE MILLIMETERS.

UNLESS OTHERWISE SPECIFIED TOLERANCES ON DECIMALS X XX XXX ∠	DRAWN BY T. COLEY	DATE 4-20-87	commodore
	CHKD		A500 EXPANSION CONNECTOR LOCATION
	ENGR H.MOSTELLER	4-20-87	
	APPR		
MATERIAL	USED ON AMIGA A500	NEXT ASSY	SIZE D 312613 REV 1
FINISH			SCALE FULL SHEET 1 OF 1

1

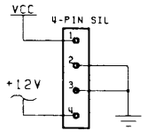

CNY
+5X
XDRD(15:0)
RP901 68
RP903 68
XD(15:0)
XA(23:1)
_XCLKRD
_XCLKWR
+12X
U9
MSM6242B
TP3
R911 470
D911 1N914
D912 1N914
R913 470
BT1
3/600K
N1-Cd
4.5V
180mAH
C913 4.7uF
C914 .1uF
Y2
32768Hz
C911 20pF
C912 6.8-45pF
C902 47uF
C903 47uF
U48 C48 .22uF
U49 C49 .22uF
U51 C51 .22uF
U52 C52 .22uF
U53 C53 .22uF
U54 C54 .22uF
U55 C55 .22uF
U56 C56 .22uF
U57 C57 .22uF
U59 C59 .22uF
U60 C60 .22uF
U61 C61 .22uF
U62 C62 .22uF
U63 C63 .22uF
A500-8
MEMORY EXPANSION AND REAL TIME CLOCK
Commodore
C/A500 Main Board
"Rock Lobster"
C 312511 5
SHEET 9 OF 9

POWER INPUT



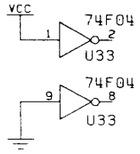
NOTE: HEAVY LINES INDICATE A
SINGLE POINT CONNECTION

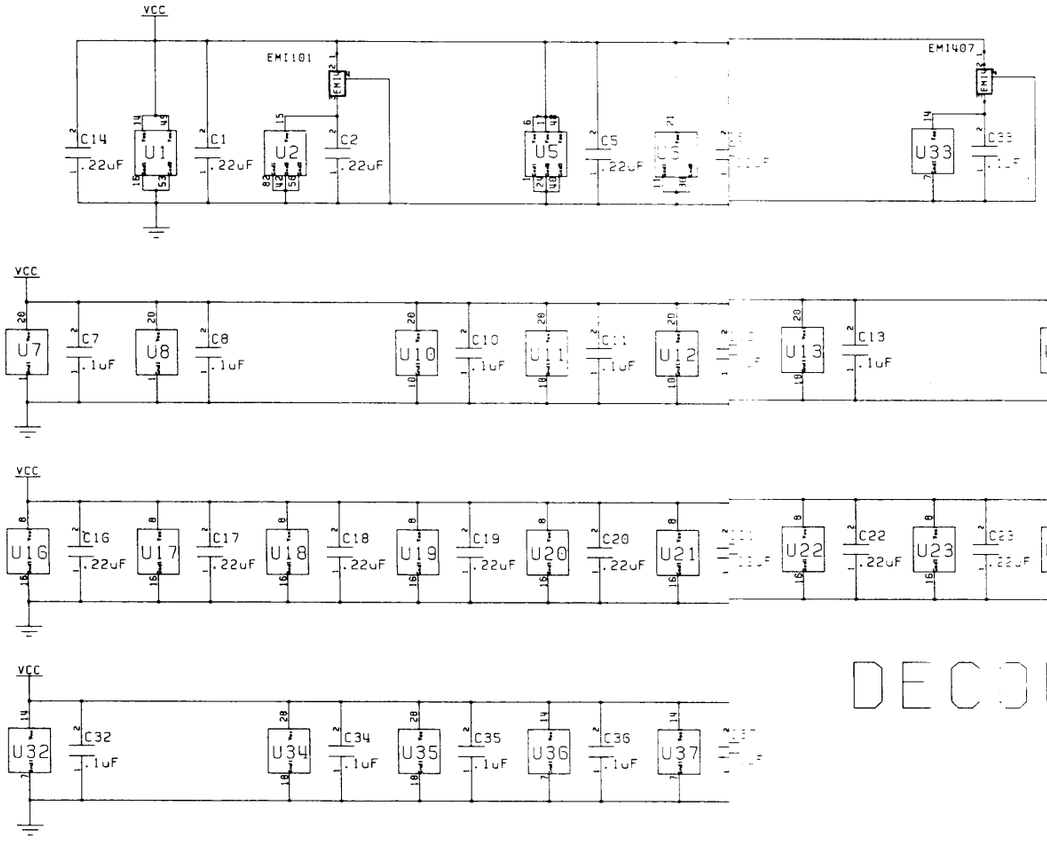
FLOPPY POWER



CN12

SPARES





DECOUPLING

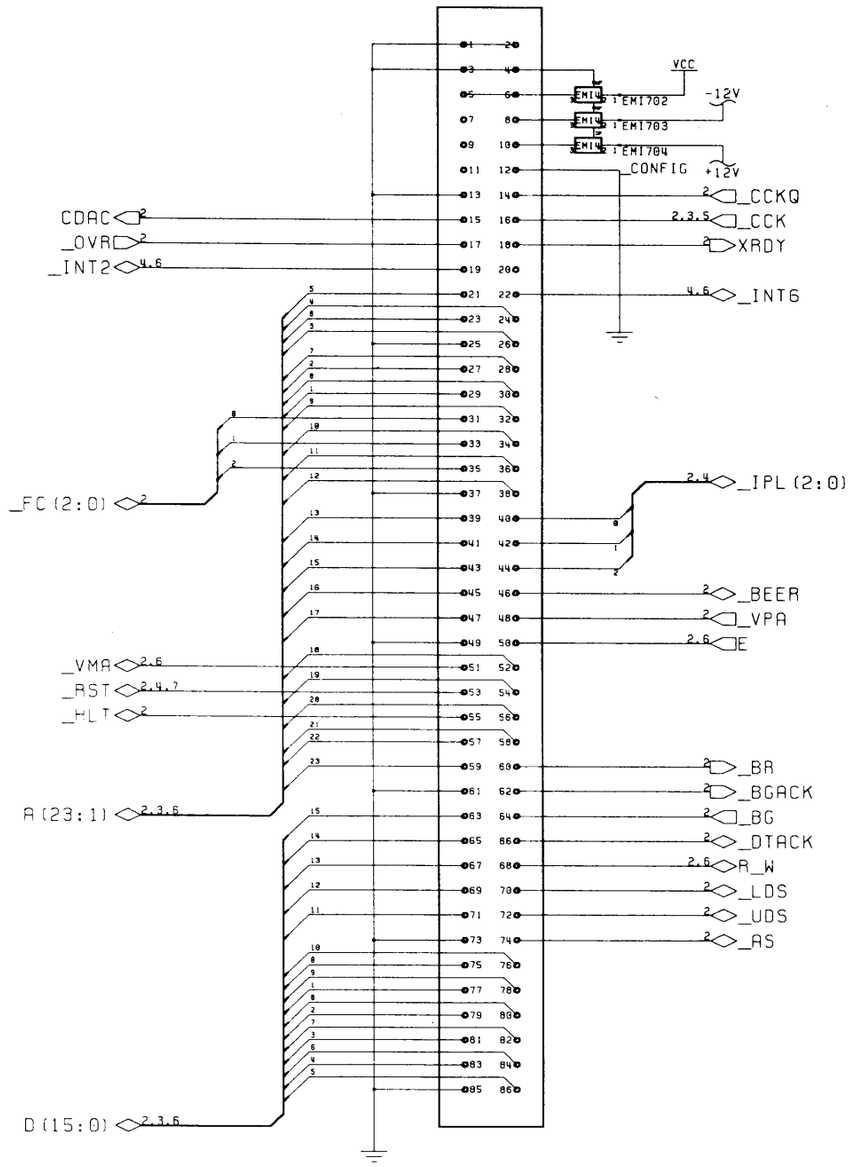
A500-7

POWER DISTRIBUTION AND DECOUPLING

DRAWN BY: G. Robbins	DATE 05/24/87	Commodore	
CHKD:			
ENGR: GRR	05/24/87		
APPR:		C/A500 Main Board "Rock Lobster"	
USED ON C/A500	NEXT ASST 312518	SIZE C 312511	REV 5
		SCALE	SHEET 8 OF 9

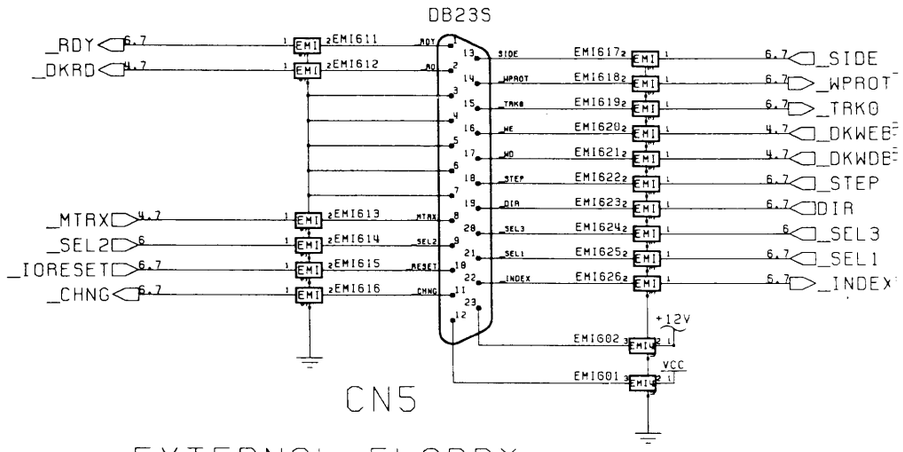
EXPANSION

P1



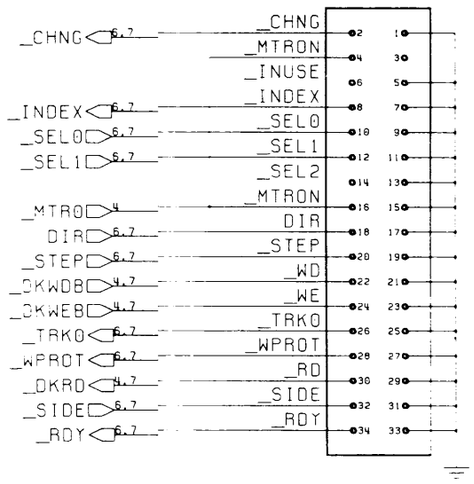
CN5

EXTERNAL FLOPPY

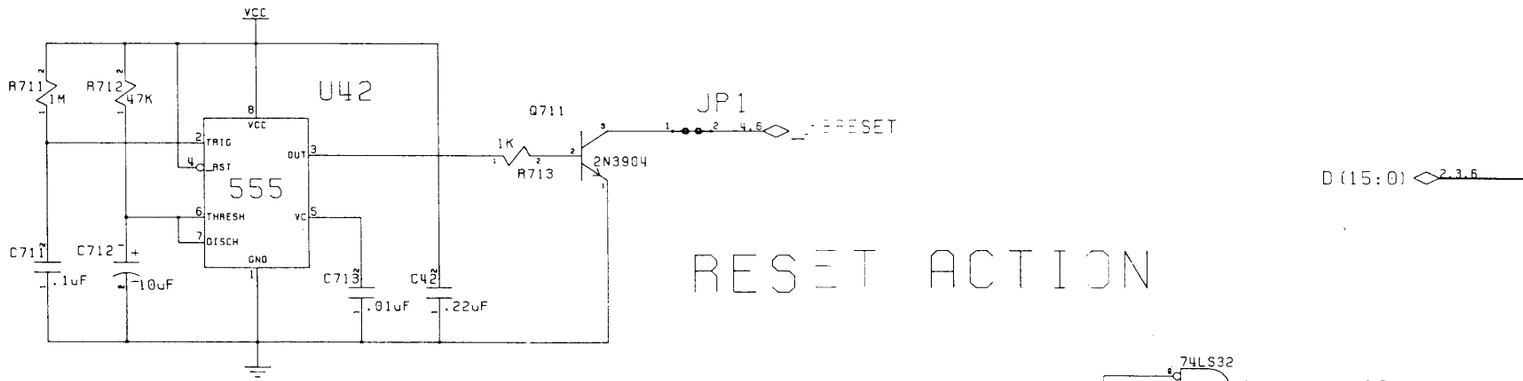


INTERNAL FLOPPY

CN11



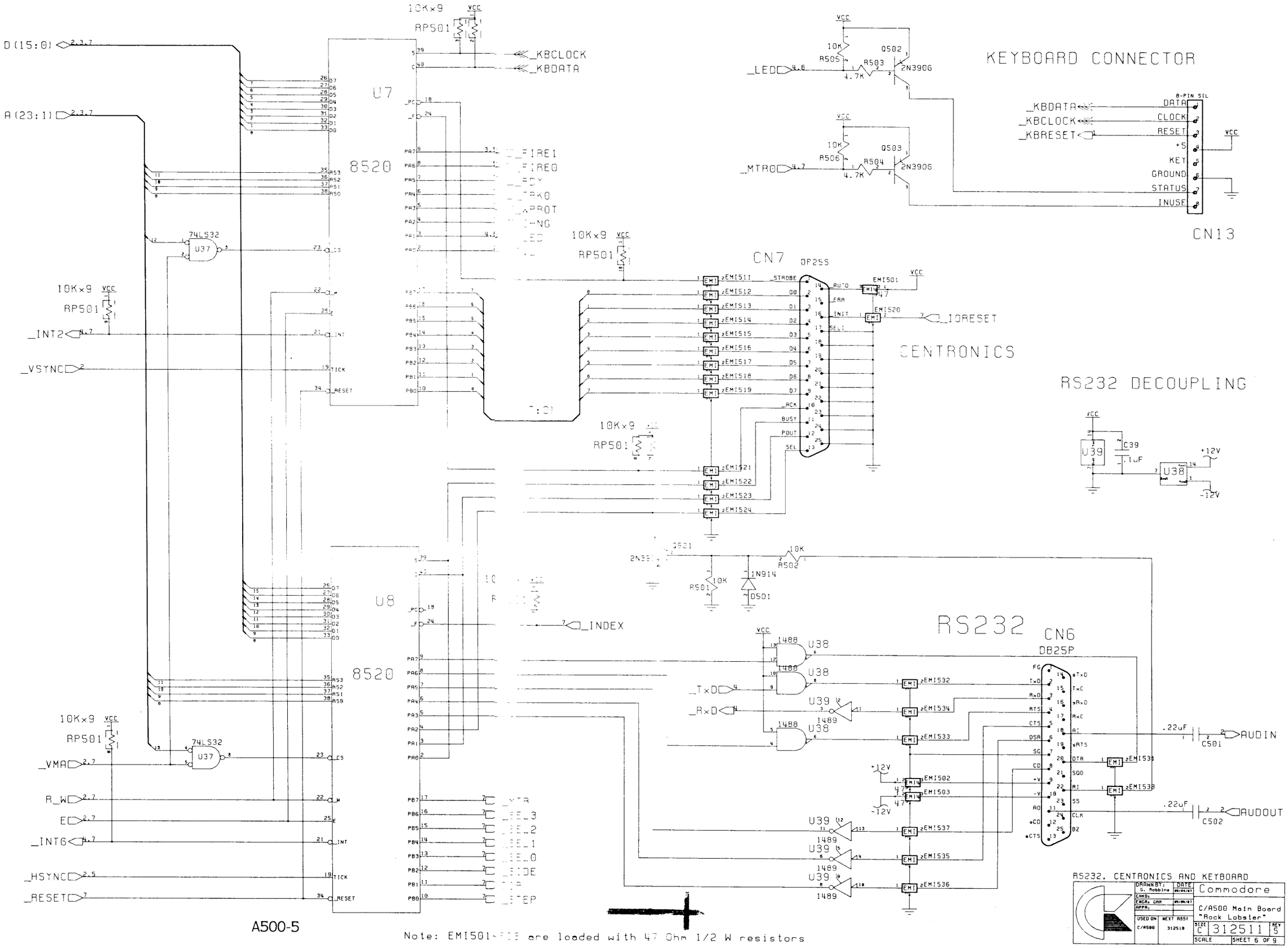
RESET ACTION



A500-6

FLOPPY DISK AND EXXPASION CONNECTORS

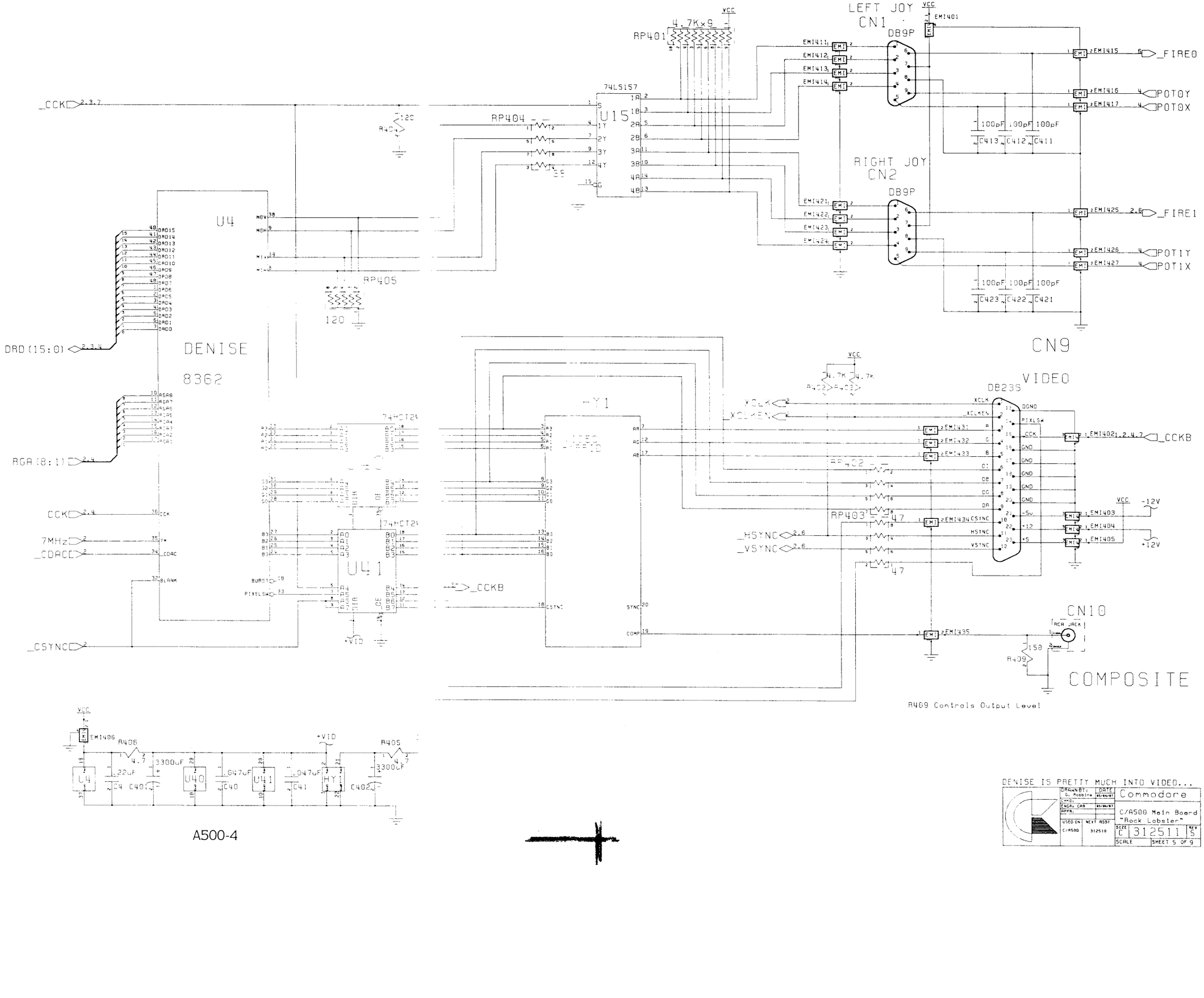
DRAWN BY: G. Mabbine	DATE: 05/06/87	Commodore	
CHKD:			
ENGR: GMM	05/06/87	C/A500 Main Board "Rock Lobster"	
APPR:			
USED ON: C/A500	NEXT ASST: 312510	SIZE C	312511 REV S
		SCALE	SHEET 7 OF 9



A500-5

Note: EMI501~EMI539 are loaded with 47 Ohm 1/2 W resistors

note: EMI401 is loaded with 4.7 Ohm Resistor

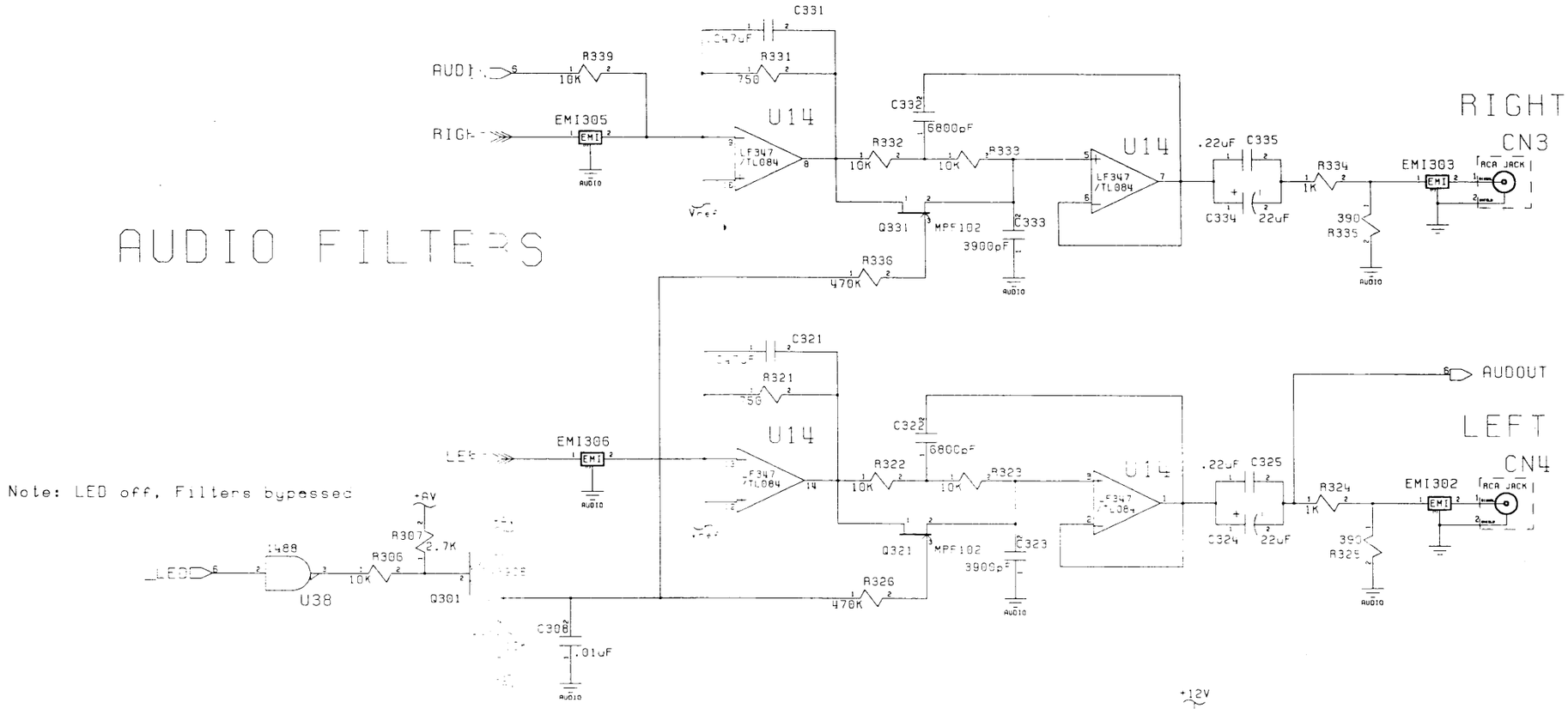


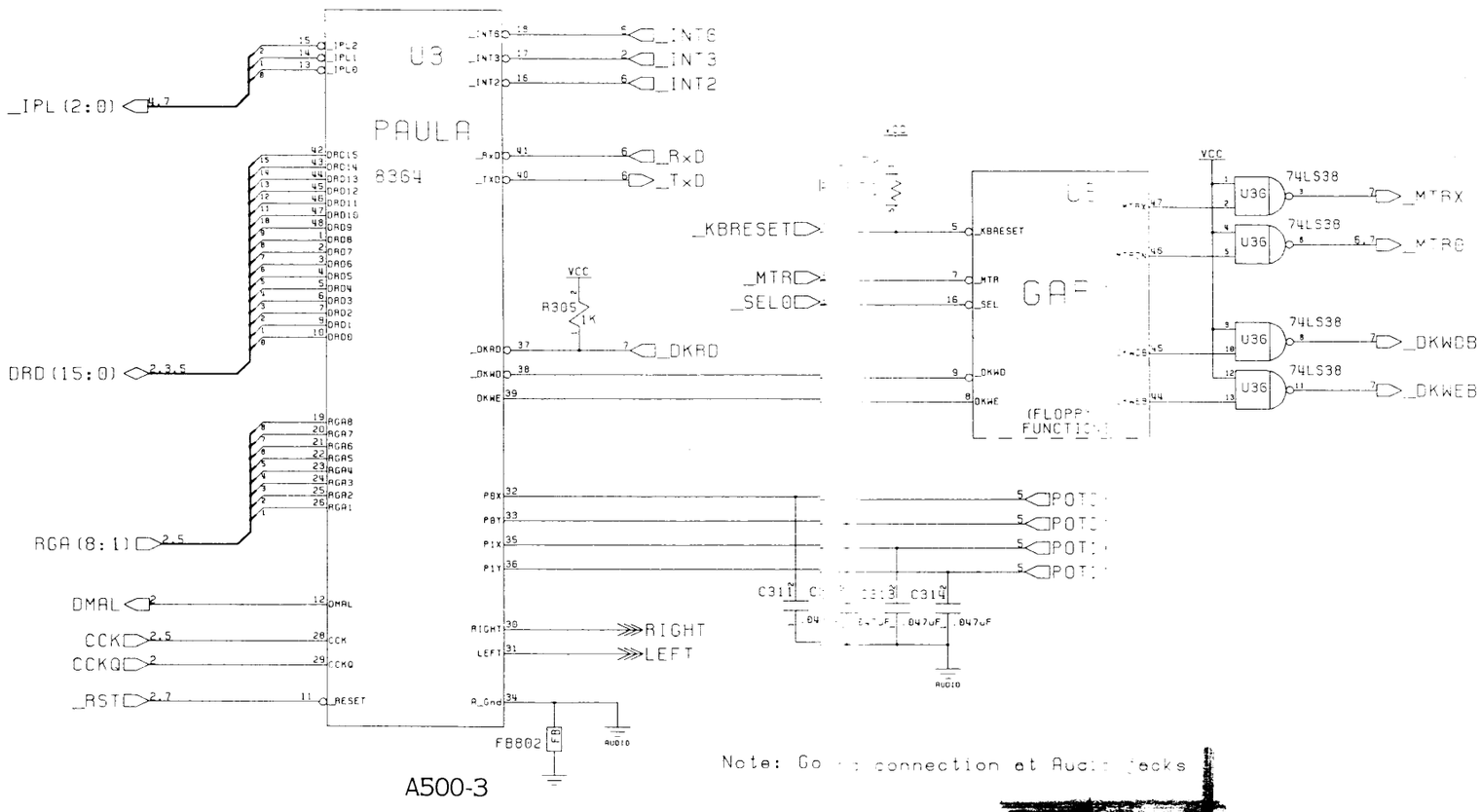
A500-4

DENISE IS PRETTY MUCH INTO VIDEO...

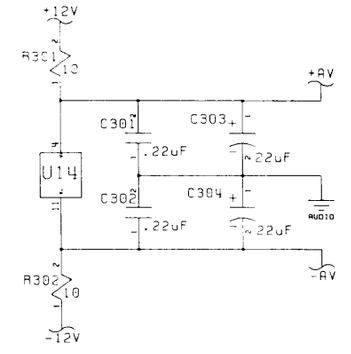
DRAWN BY: G. Robbins	DATE 85/04/07	Commodore	
CHKD:			
ENGR: CMR	85/04/07	C/A500 Main Board "Rock Lobster"	
APPR:			
USED ON C/A500	NEXT ASST 312510	SIZE C	312511 REV 5
		SCALE	SHEET 5 OF 9

AUDIO FILTERS

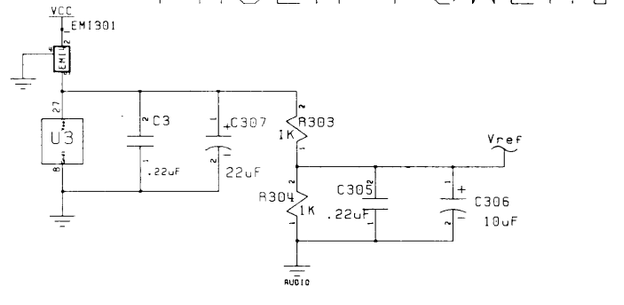




A500-3

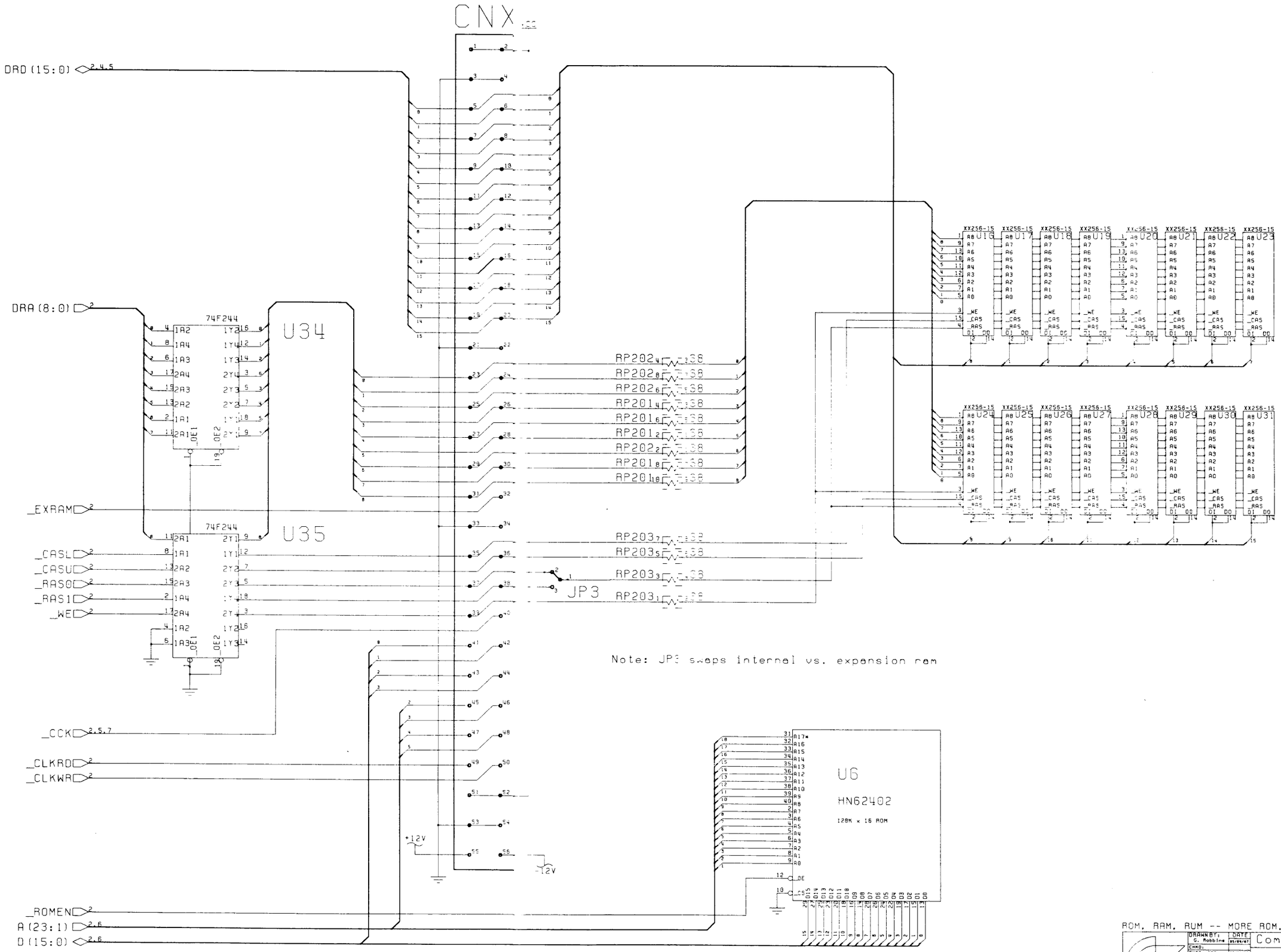


PAULA POWER!



PAULA DOES THINGS THAT DENISE DOESN'T

DRAWN BY: G. Robbins	DATE: 05/06/87	Commodore	
CHKD:		C/A500 Main Board "Rock Lobster"	
ENGR: GMR	05/06/87		
APPR:			
USED ON: C/A500	NEXT ASST: 312510	SIZE: C	312511 REV: 5
		SCALE	SHEET 4 OF 9



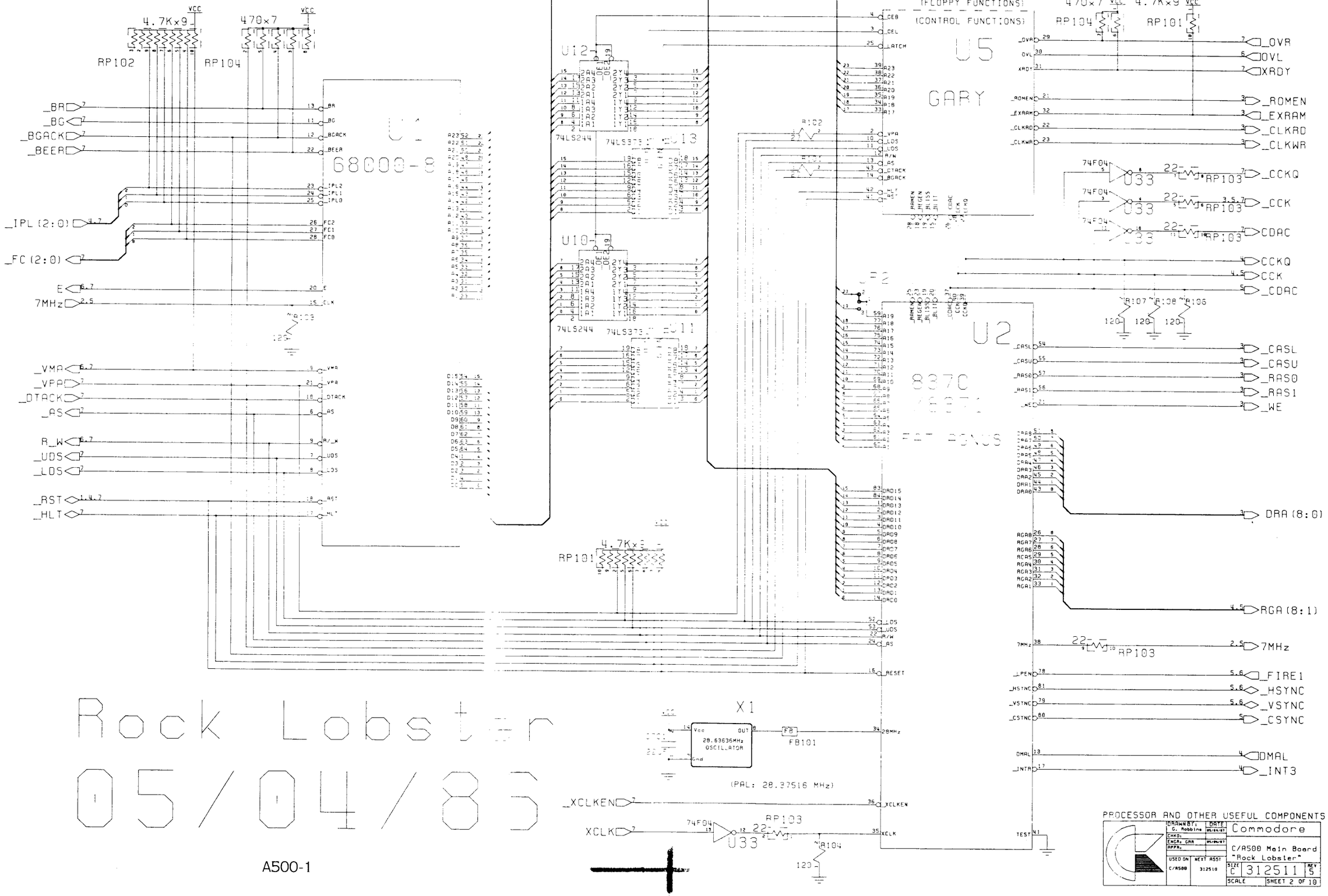
A500-2

ROM, RAM, RUM -- MORE ROM, RAM, RUM!

DRAWN BY: G. Robbins	DATE 05/04/87	Commodore
CHKD:		
ENGR: GRR	05/04/87	C/A500 Main Board "Rock Lobster"
APPR:		
USED ON C/A500	NEXT ASSY 312510	SIZE C 312511 REV 5
		SCALE SHEET 3 OF 9

Note: R103-104,106-108 are from EMI control
and may not be loaded in all cases

JP2 controls where expansion ram maps to:
A23 -> C00000 (default), A19 -> 080000



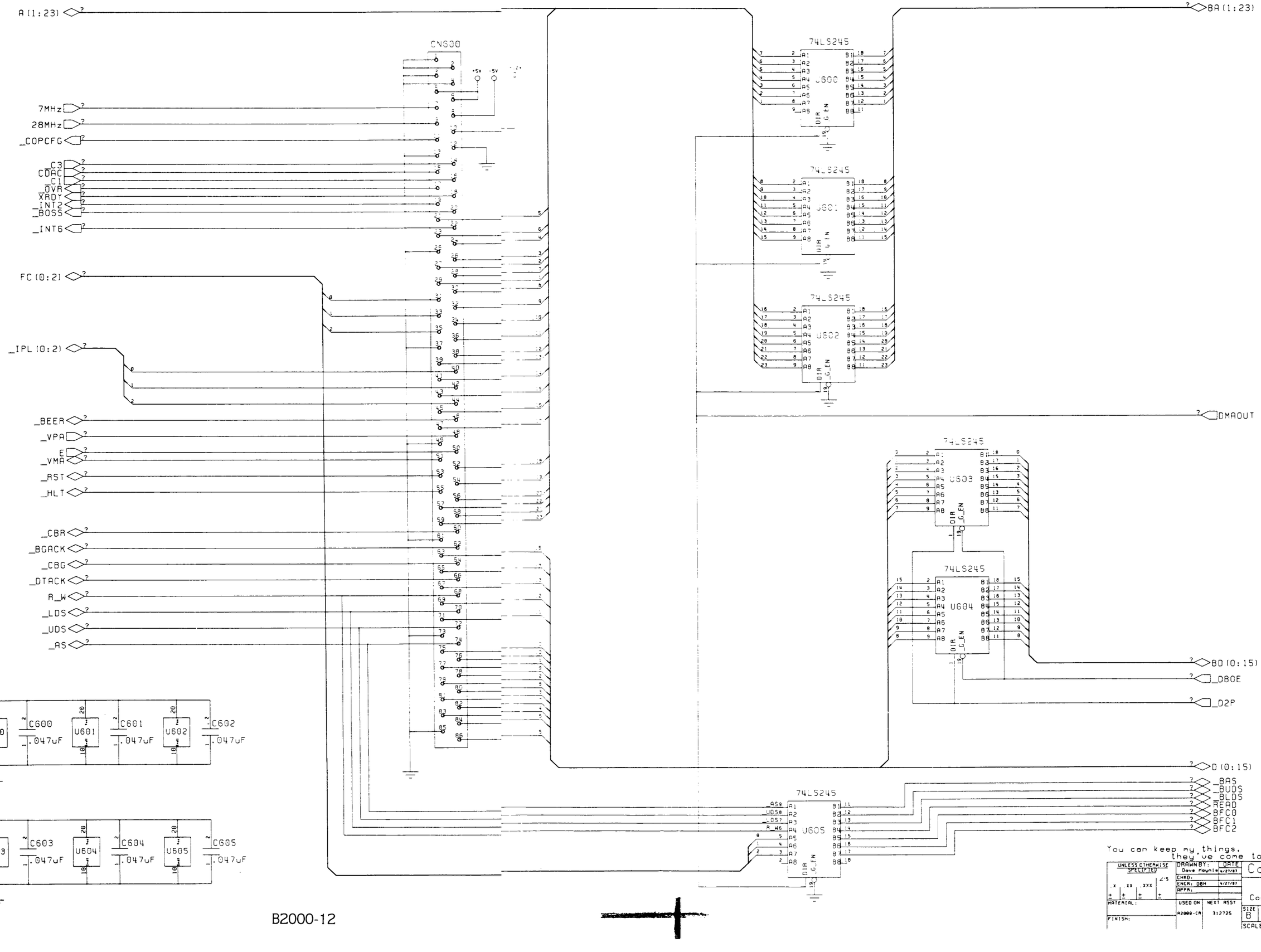
Rock Lobster

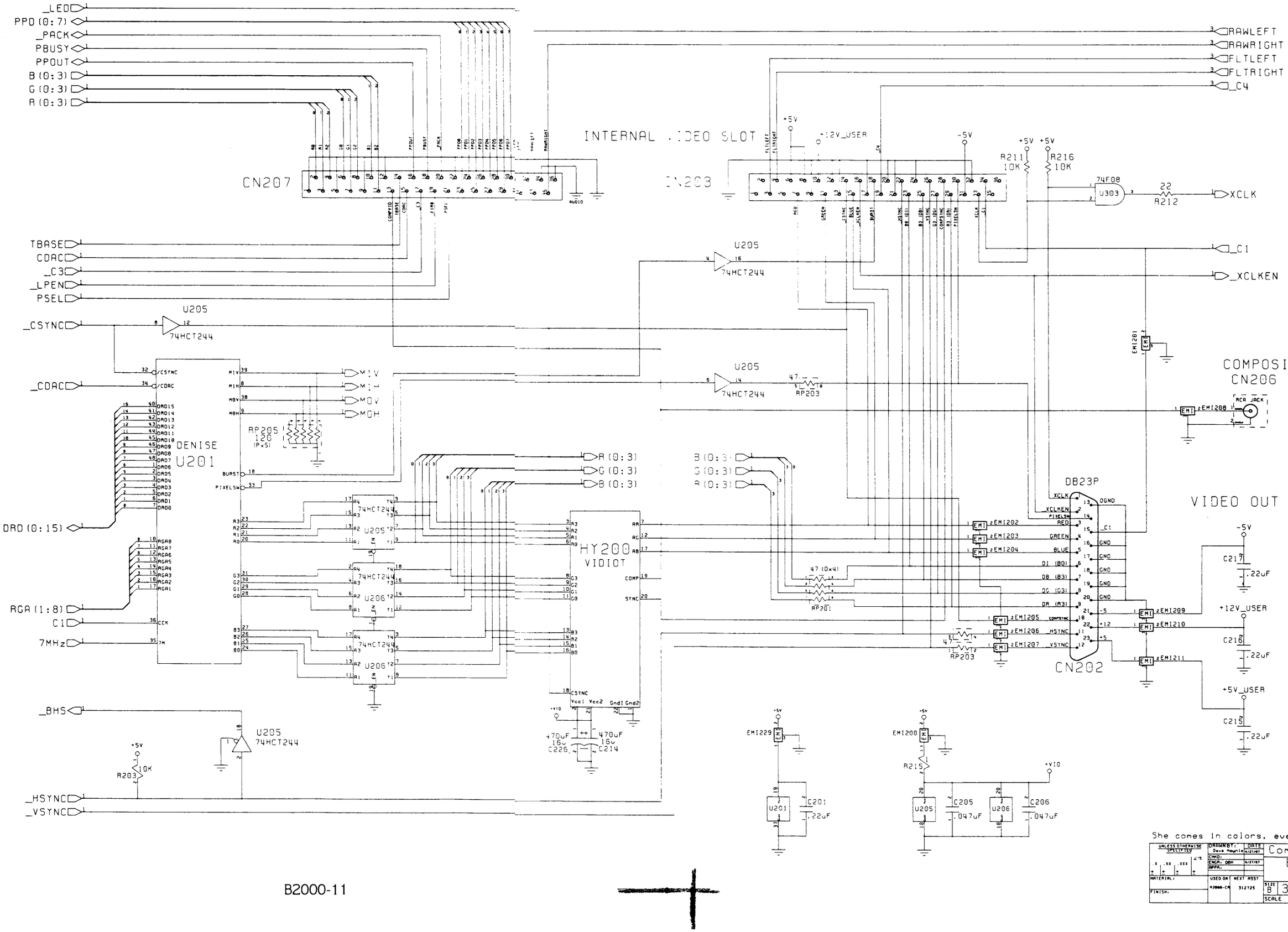
05/04/86

A500-1

PROCESSOR AND OTHER USEFUL COMPONENTS

DRAWN BY: G. Mobbins	DATE 05/04/87	Commodore	
CHKD:			
ENGR: GMR	05/04/87	C/A500 Main Board "Rock Lobster"	
APPR:			
USED ON C/A500	NEXT ASSY 312510	SIZE C 312511	REV 5
		SCALE	SHEET 2 OF 10

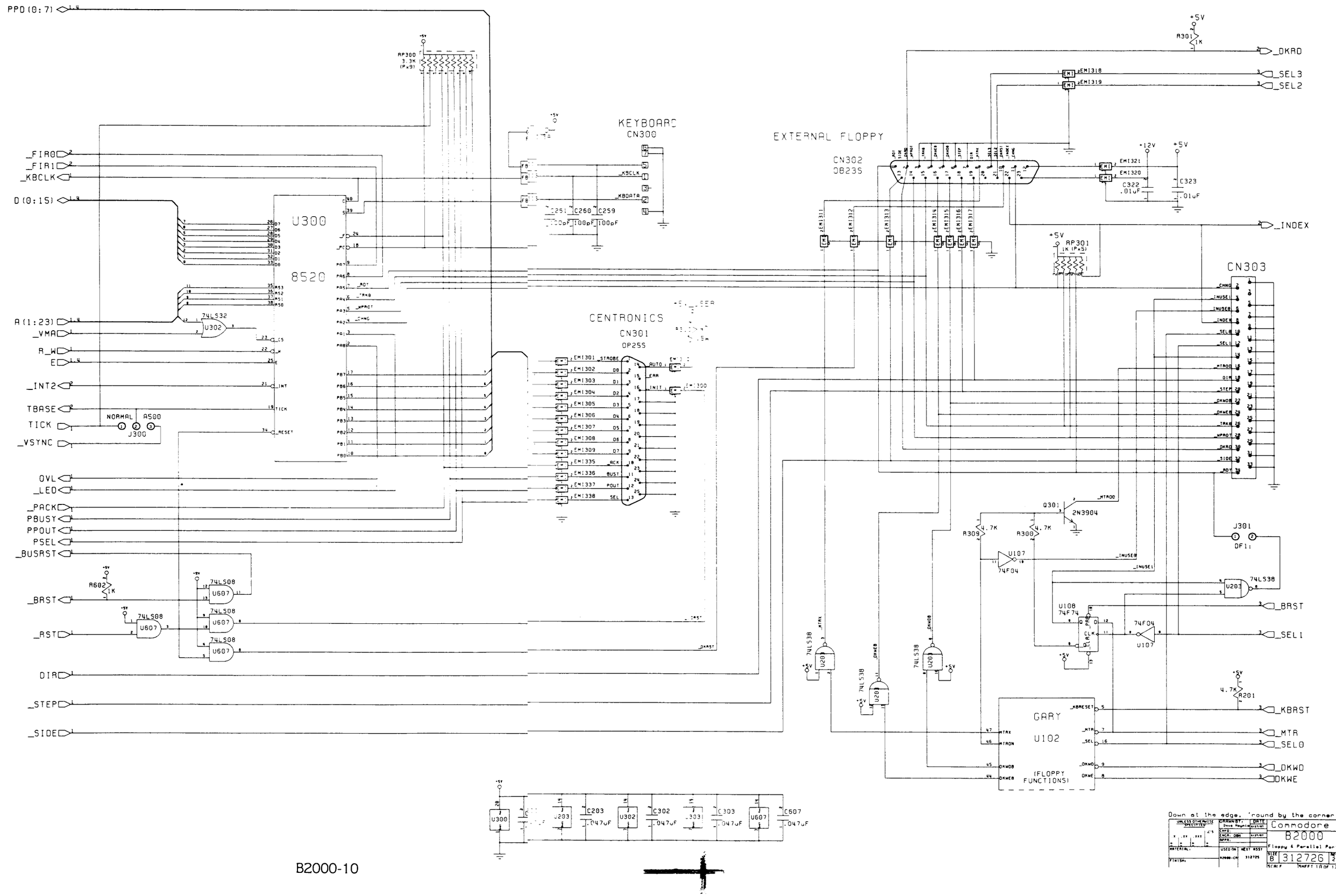

A(1:23)
BA(1:23)
CN600
7MHz
28MHz
_COPCFG
_C3
COAC
_C1
_OVR
XRDY
_INT2
_BOSS
_INT6
FC(0:2)
_IPL(0:2)
_BEER
_VPA
E
_VMA
_RST
_HLT
_CBR
_BGACK
_CBG
_DTACK
R_W
_LDS
_UDS
_AS
74LS245
U600
U601
U602
U603
U604
U605
DMAOUT
BD(0:15)
_DBOE
_D2P
D(0:15)
_BAS
_BUDS
_BLDS
READ
BFC0
BFC1
BFC2
C600
C601
C602
C603
C604
C605
.047uF
+5V
You can keep my things.
they've come to take me hom
UNLESS OTHERWISE SPECIFIED
DRAWN BY: Dave Maynie
DATE 4/27/87
CHKD:
ENGR: DBM 4/27/87
APPR:
MATERIAL:
FINISH:
USED ON A2000-CR
NEXT ASSY 312725
Commodore
B2000
Coprocessor Slot
SIZE B
312726
SCALE
SHEET 12 OF 1

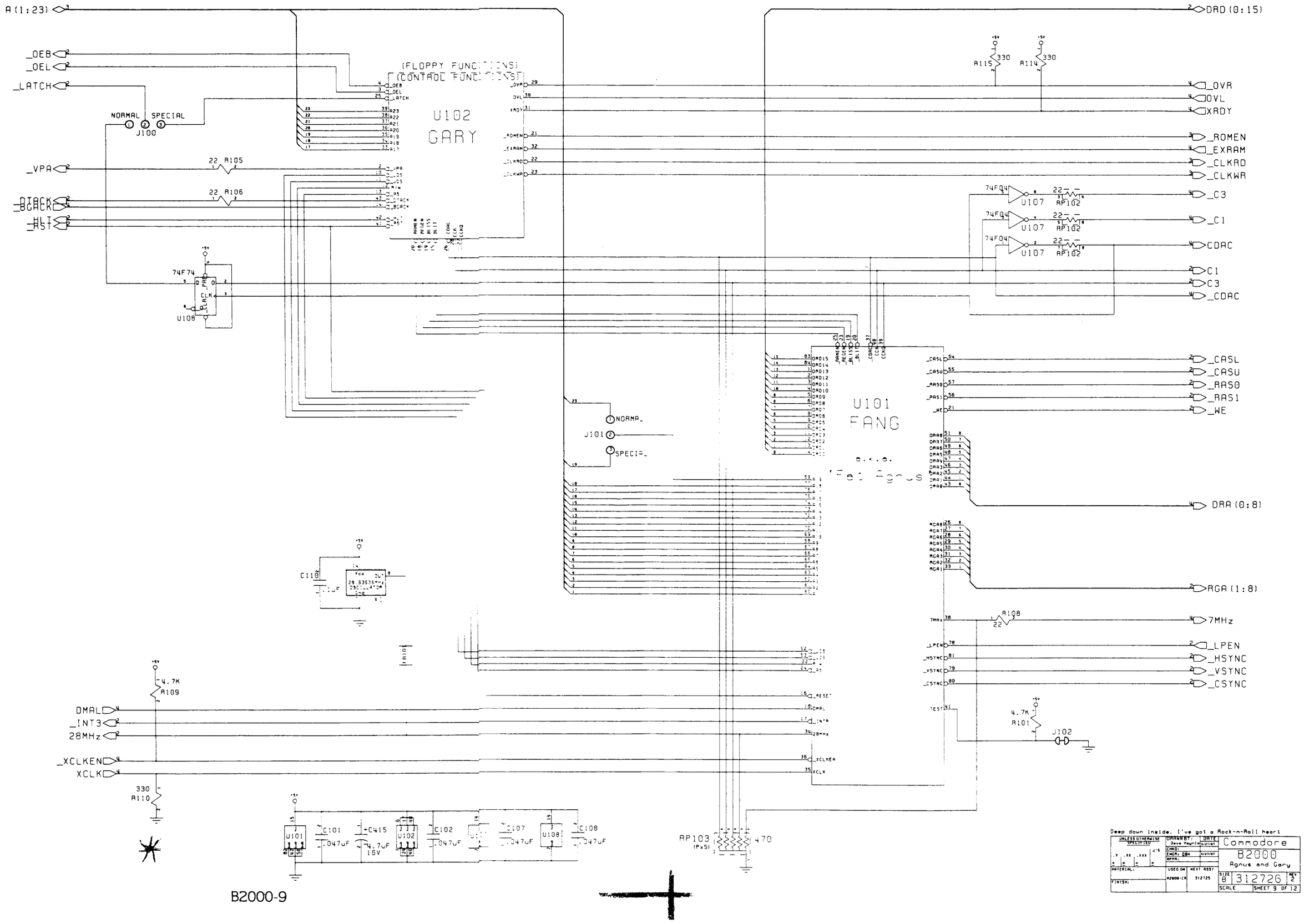


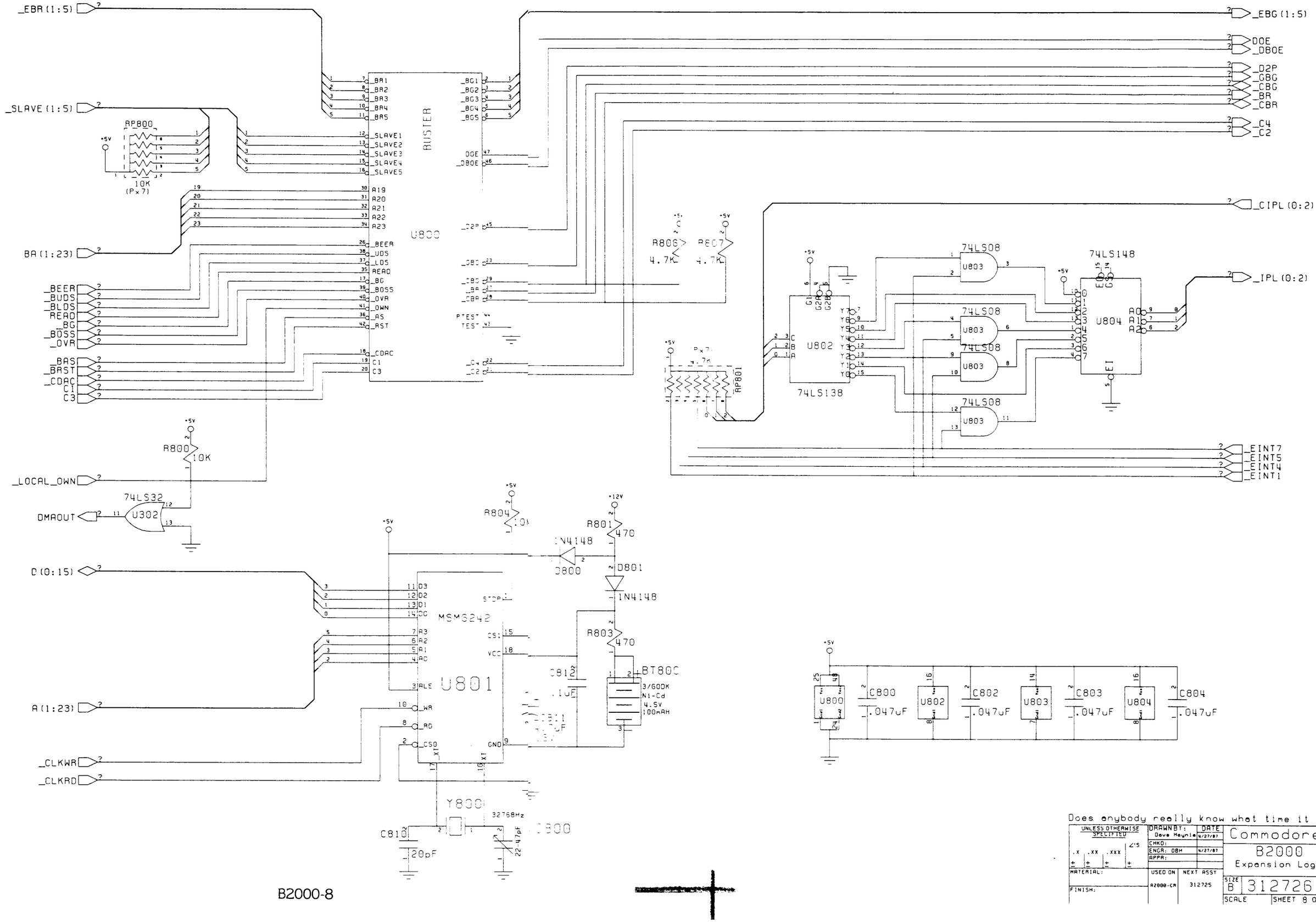
B2000-11

She comes in colors, everywhere

UNLESS OTHERWISE SPECIFIED	DRAWN BY: Dave Meynie	DATE 4/27/87	Commodore
	CHKD:		B2000
	ENGR: DBM	4/27/87	Video
	APPR:		
MATERIAL:	USED ON A2000-CR	NEXT ASSY 312725	SIZE B 312726
FINISH:			SCALE SHEET 11 OF


PPD(0:7)
_FIR0
_FIR1
_KBCLK
D(0:15)
A(1:23)
_VMA
R_W
E
_INT2
TBASE
TICK
_VSYNC
OVL
_LED
_PACK
PBUSY
PPOUT
PSEL
_BUSRST
_BRST
_RST
DIR
_STEP
_SIDE
U300
8520
74LS32
U302
NORMAL
A500
J300
RP300
3.3K
(Px9)
KEYBOARD
CN300
_KBCLK
_KBDATA
C261
C260
C259
100pF
CENTRONICS
CN301
DP25S
EXTERNAL FLOPPY
CN302
DB23S
EMI321
EMI320
C322
.01uF
C323
.01uF
RP301
1K (Px5)
CN303
_DKRD
_SEL3
_SEL2
_INDEX
Q301
2N3904
R309
4.7K
R300
4.7K
U107
74F04
J301
DF1:
U203
74LS38
U108
74F74
74F04
U107
_BRST
_SEL1
R201
4.7K
_KBRST
_MTR
_SEL0
_DKWD
_DKWE
GARY
U102
(FLOPPY FUNCTIONS)
74LS38
U203
74LS38
U203
74LS38
U203
74LS08
U607
R602
1K
U300
U203
C203
.047uF
U302
C302
.047uF
U303
C303
.047uF
U607
C607
.047uF
Down at the edge, 'round by the corner
Commodore
B2000
Floppy & Parallel Por
312726


A(1:23)
_OEB
_OEL
_LATCH
NORMAL
SPECIAL
J100
_VPA
_DTACK
_BGACK
_HLT
_RST
22 R105
22 R106
74F74
U108
(FLOPPY FUNCTIONS)
(CONTROL FUNCTIONS)
U102
GARY
DRD(0:15)
R115 330
R114 330
_OVR
OVL
XRDY
_ROMEN
_EXRAM
_CLKRD
_CLKWR
74F04
U107
RP102
_C3
_C1
CDAC
C1
C3
_CDAC
U101
FANG
e.k.a.
'Fat Agnus'
_CASL
_CASU
_RAS0
_RAS1
_WE
DRA(0:8)
RGA(1:8)
7MHz
_LPEN
_HSYNC
_VSYNC
_CSYNC
R108
22
R101
J102
NORMAL
J101
SPECIAL
C110
4.7K
R109
DMAL
_INT3
28MHz
_XCLKEN
XCLK
330
R110
U101
C101
.047uF
C415
4.7uF
16V
U102
C102
.047uF
C107
.047uF
U108
C108
.047uF
RP103
(PxS)
470
Deep down inside. I've got a Rock-n-Roll heart
UNLESS OTHERWISE SPECIFIED
DRAWN BY: Dave Haynie
DATE
CHKD:
ENGR: DBH
APPR:
MATERIAL:
FINISH:
USED ON
A2000-CR
NEXT ASSY
312725
Commodore
B2000
Agnus and Gary
SIZE
B
312726
REV
2
SCALE
SHEET 9 OF 12

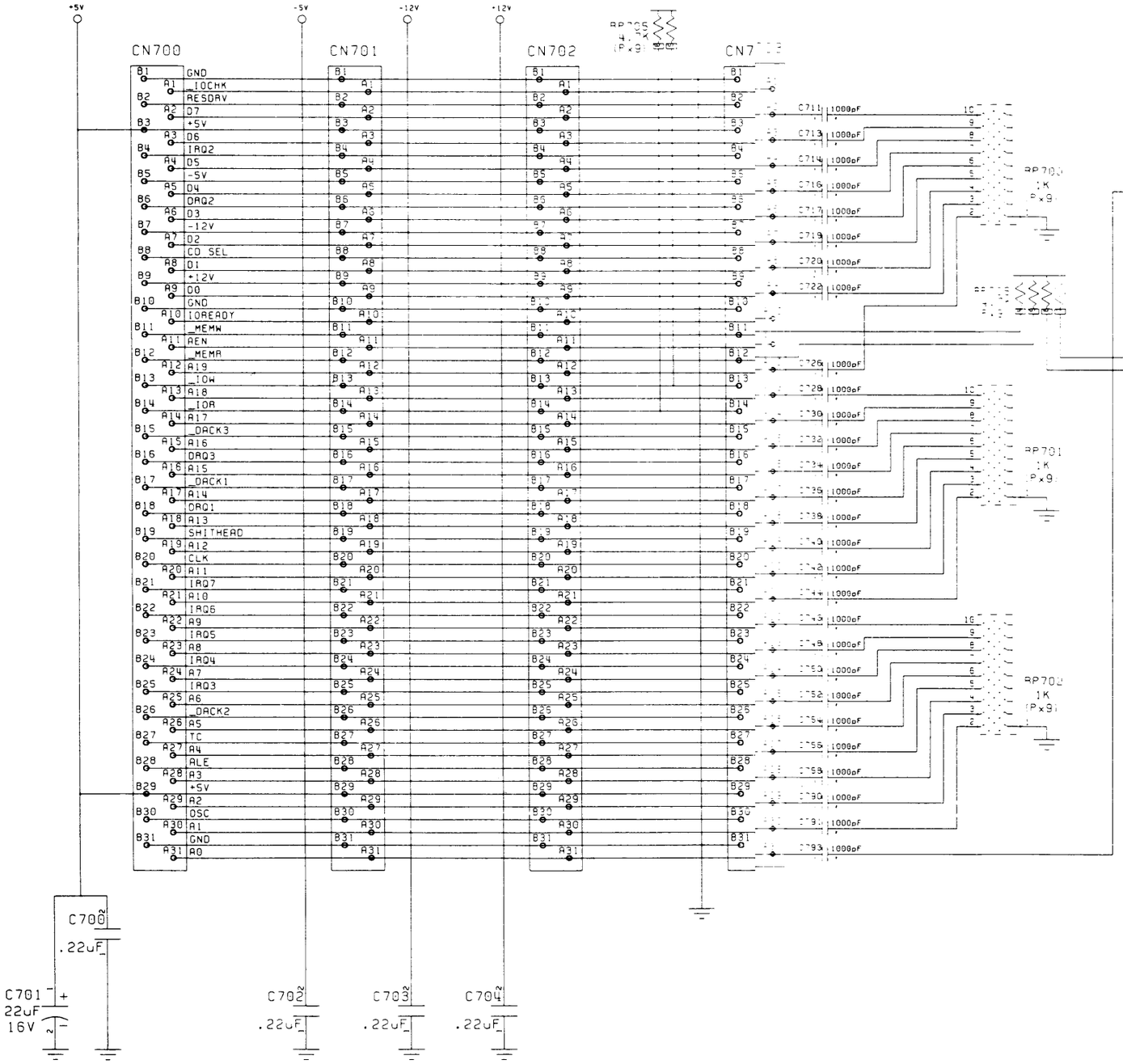


B2000-8

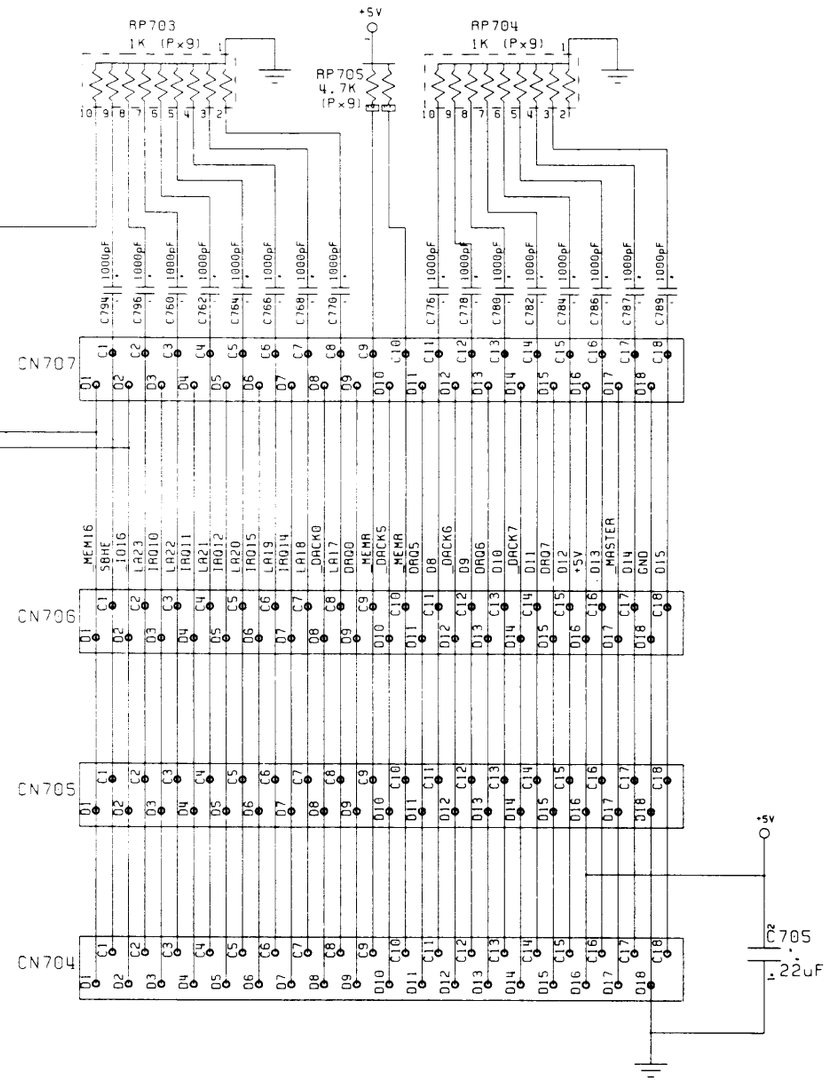
Does anybody really know what time it is?

UNLESS OTHERWISE SPECIFIED: .X ±, .XX ±, .XXX ±, ∠'S	DRAWN BY: Dave Haynie	DATE 4/27/87	Commodore
	CHKD:		B2000
	ENGR: DBH	4/27/87	Expansion Logic
	APPR:		
MATERIAL:	USED ON A2000-CM	NEXT ASSY 312725	SIZE B 312726 REV 2
FINISH:			SCALE SHEET 8 OF 12

IBM 8 BIT EXPANSION BUS

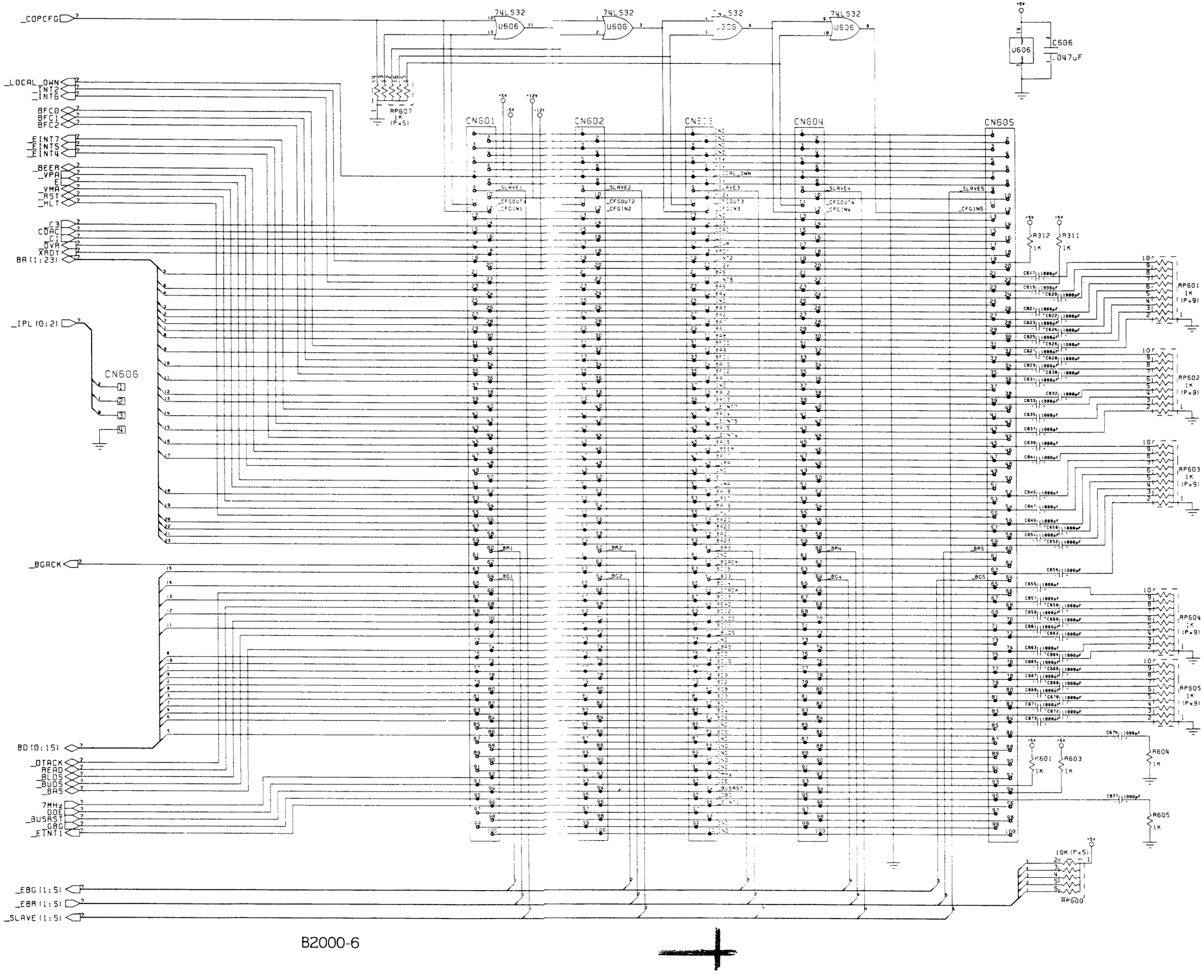


IBM 16 BIT EXTENDED BUS



I wait in this place,
where the sun never shines

UNLESS OTHERWISE SPECIFIED: .X ±, .XX ±, .XXX ±, ∠'S ±	DRAWN BY: Dave Haynie	DATE 4/27/87	Commodore
	CHKD:		B2000
	ENGR: DBH	4/27/87	IBM PC Bus
	APPR:		
MATERIAL:	USED ON: B2000-CM	NEXT ASSY: 312725	SIZE B 312726 REV 2
FINISH:			SCALE SHEET 7 OF 12



B2000-6

Goes like thunder; its a Bus-Age wonder

UNLESS OTHERWISE SPECIFIED	DRAWN BY: Dave Haynie	DATE 6/27/87	Commodore
.X ± .XX ± .XXX ± ∠ 5°	CHK'D:		B2000
	ENGR: DBH	6/27/87	
	APPR:		Amiga Expansion
MATERIAL:	USED ON B2000-CA	NEXT ASST 312725	SIZE B 312726 REV 2
FINISH:			SCALE SHEET 6 OF 12

DRD(0:15)

DRA(0:8)

DRB(0:8)

_CASL
_CASU
_RAS0
_RAS1
_WE
_EXRAM
A(1:23)
_ROMEN
D(0:15)

74F244 U540
74F244 U540
74F244 U541
74F244 U541
74F244 U541
74F244 U541

47(0x5) RP500
47(0x5) RP501
22(0x5) RP502
470(Px9) RP503

R500 4.7K
J500

U501 U502 U503 U504 U505 U506 U507 U508
U509 U510 U511 U512 U513 U514 U515 U516
U517 U518 U519 U520 U521 U522 U523 U524
U525 U526 U527 U528 U529 U530 U531 U532

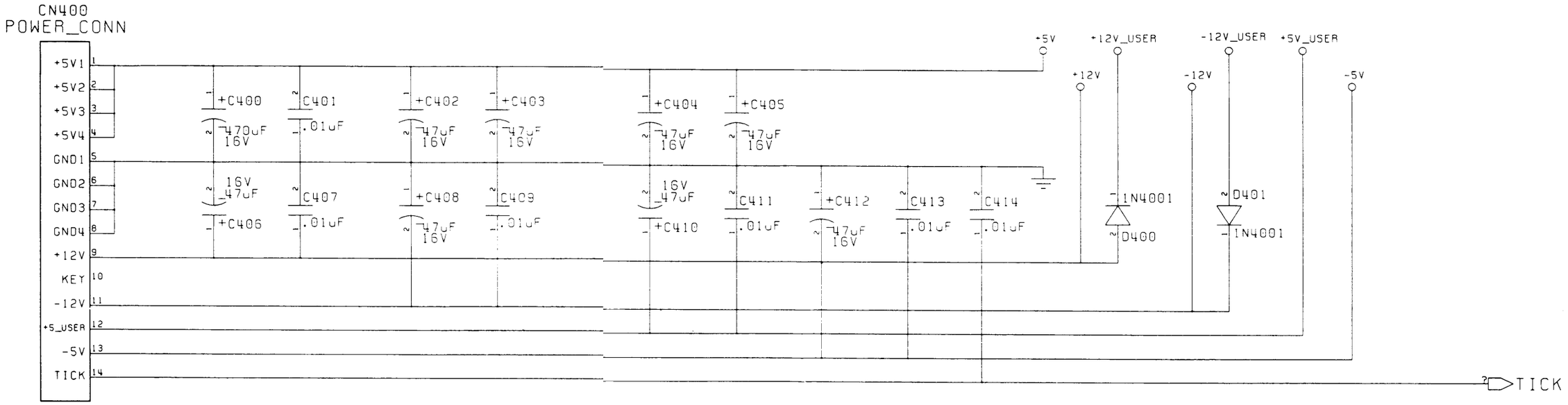
XX256-15

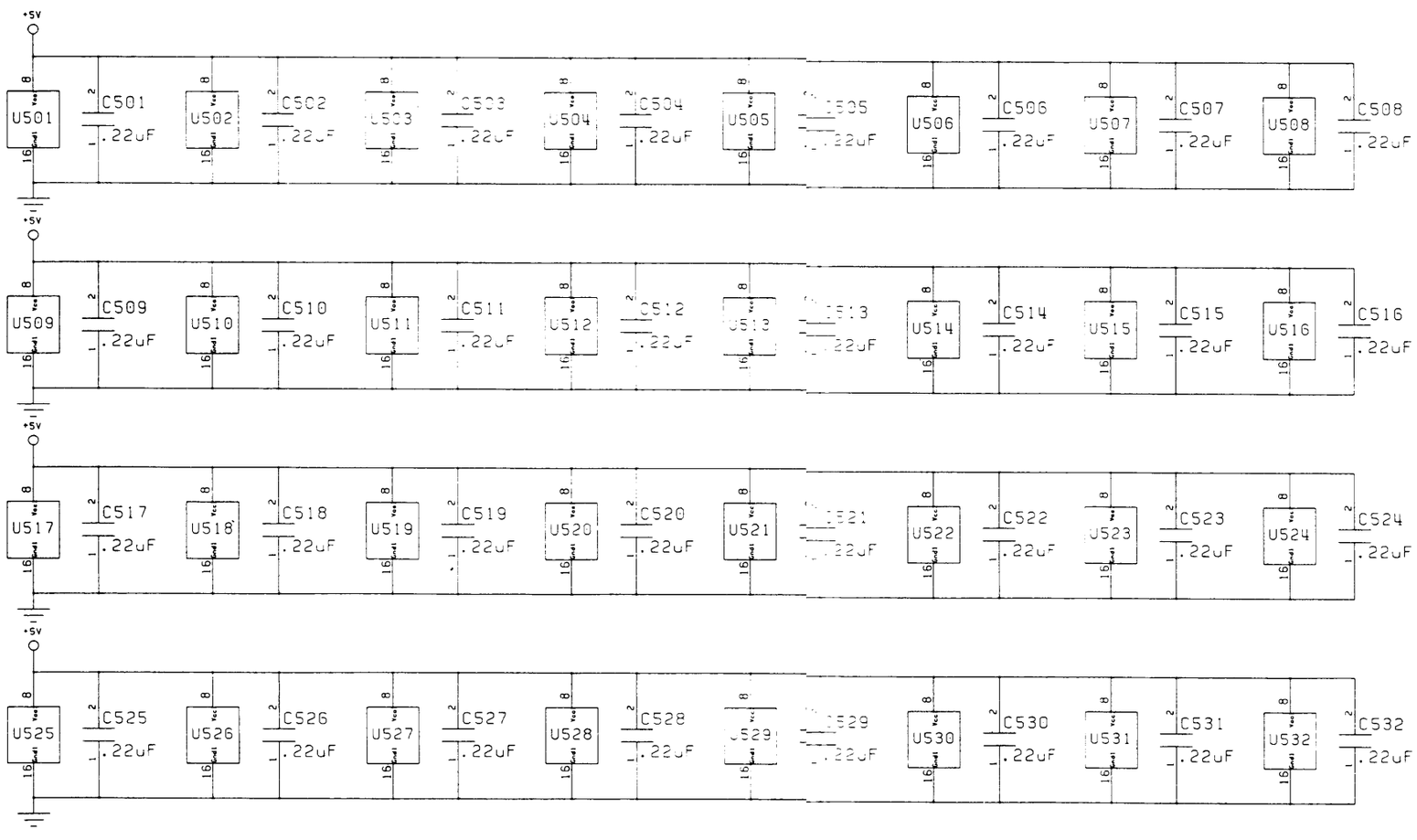
U500 C500 .22uF
U540 C540 .047uF
U541 C541 .047uF

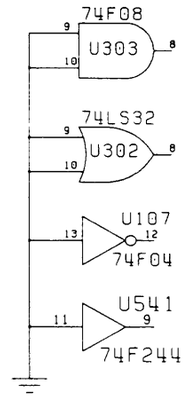
B2000-5

Memories (all alone in the moonlight)

UNLESS OTHERWISE SPECIFIED: .x ±, .xx ±, .xxx ±, ∠ ±	DRAWN BY: Dave Haynie	DATE 4/27/87	Commodore
	CHKD:		B2000 Memory
	ENGR: GBM	4/27/87	
	APPR:		
MATERIAL:	USED ON	NEXT ASSY	SIZE B 312726 REV 2
FINISH:	B2000-CR	312725	SCALE SHEET 5 OF 12





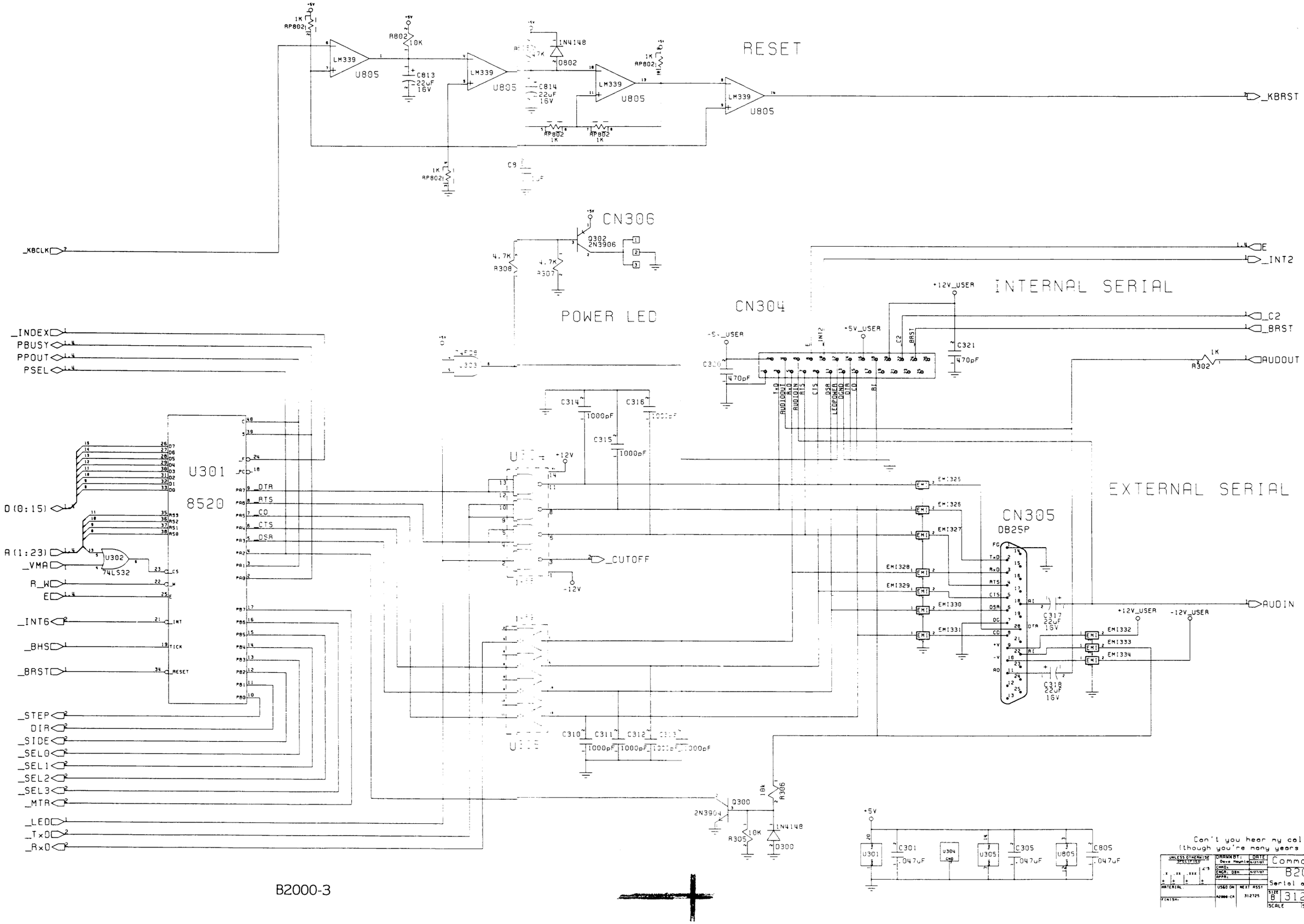


SPARES

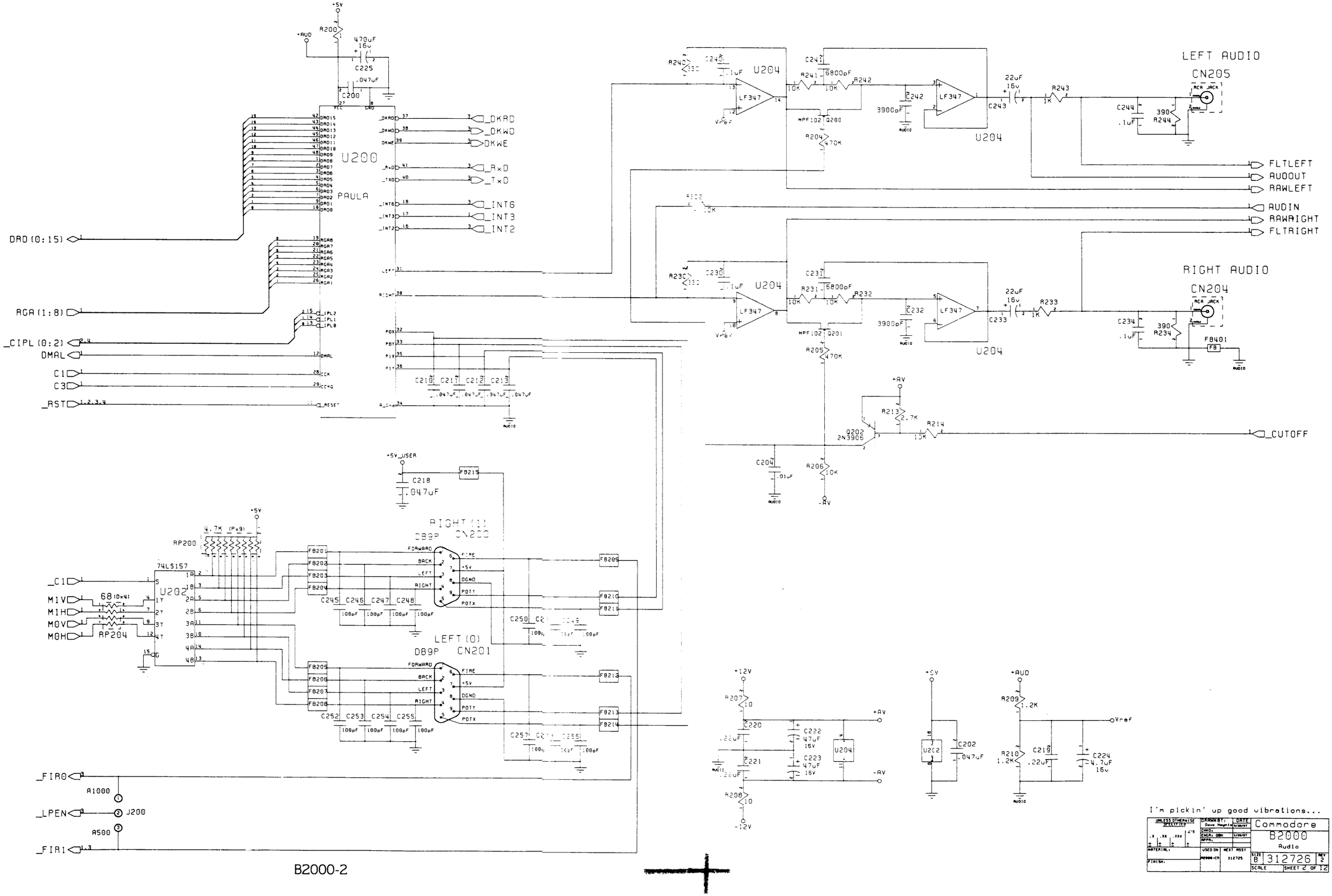
B2000-4

Baby, we were born to run...

UNLESS OTHERWISE SPECIFIED	DRAWN BY: Dave Haynie	DATE 4/27/87	Commodore
.X ± .XX ± .XXX ± ∠'S ±	CHKD:		B2000
	ENGR: DBH	4/27/87	Power
	APPR:		
MATERIAL:	USED ON	NEXT ASSY	SIZE B 312726 REV 2
FINISH:	A2000-CM	312725	SCALE SHEET 4 OF 1


RESET
LM339
U805
1K
RP802
R802
10K
C813
22uF
16V
1N4148
D802
C814
22uF
16V
_KBRST
_KBCLK
CN306
Q302
2N3906
4.7K
R308
R307
POWER LED
CN304
INTERNAL SERIAL
+12V_USER
-5V_USER
+5V_USER
C321
470pF
C300
470pF
E
_INT2
_C2
_BRST
AUDOUT
R302
_INDEX
PBUSY
PPOUT
PSEL
U301
8520
D(0:15)
A(1:23)
_VMA
R_W
E
_INT6
_BHS
_BRST
_STEP
DIR
_SIDE
_SEL0
_SEL1
_SEL2
_SEL3
_MTR
_LED
_TxD
_RxD
U302
74LS32
_DTR
_RTS
_CD
_CTS
_DSR
C314
1000pF
C316
C315
1000pF
+12V
-12V
_CUTOFF
EXTERNAL SERIAL
CN305
DB25P
EMI325
EMI326
EMI327
EMI328
EMI329
EMI330
EMI331
EMI332
EMI333
EMI334
C317
22uF
16V
C318
22uF
16V
AUDIN
+12V_USER
-12V_USER
C310
C311
C312
1000pF
Q300
2N3904
R306
10k
R305
10K
1N4148
D300
+5V
U301
C301
.047uF
U304
U305
C305
.047uF
U805
C805
.047uF
Can't you hear my call
(though you're many years away)
Commodore
B2000
Serial and Reset
B
312726
SCALE
SHEET 3 OF 1

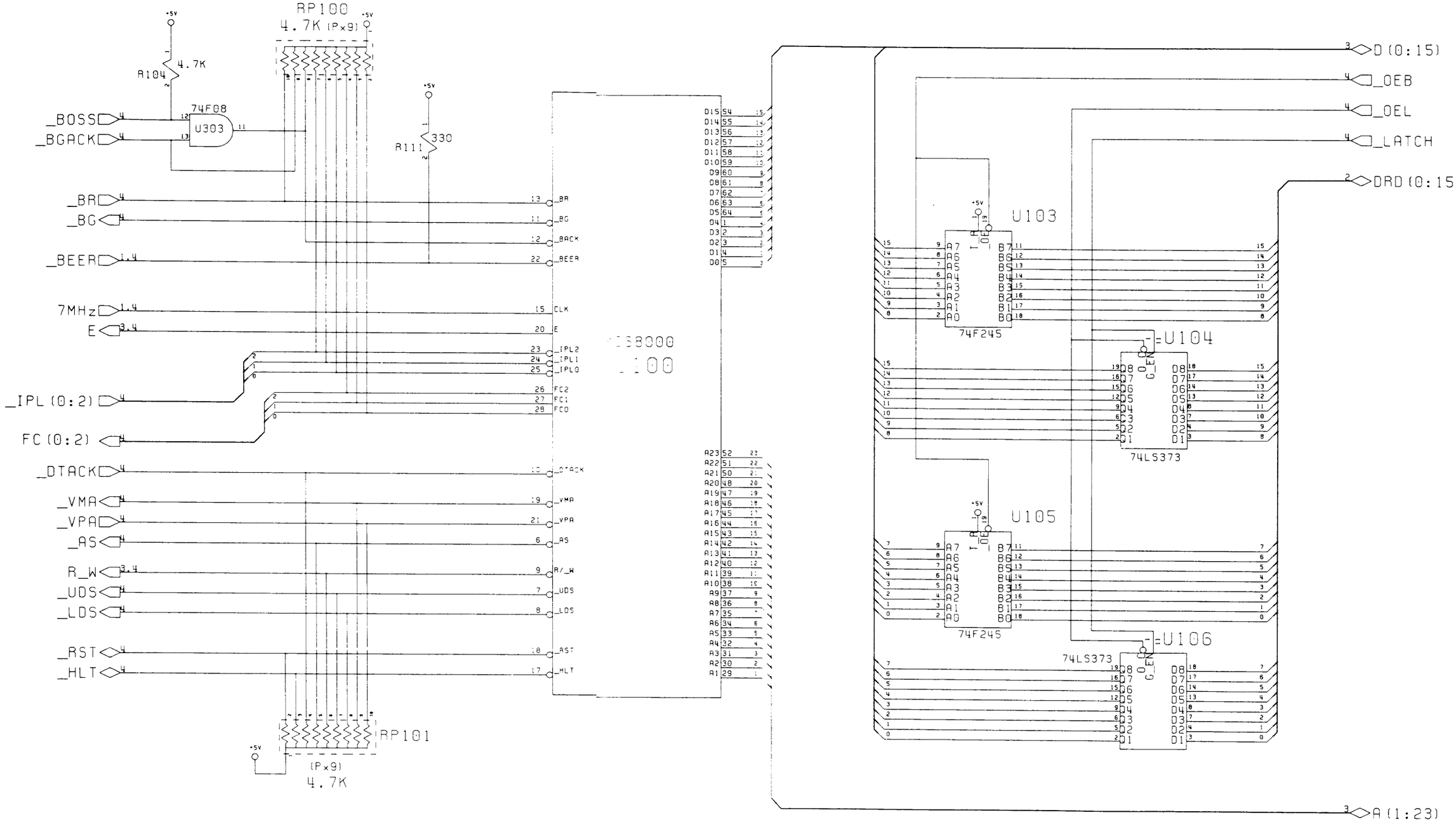
B2000-3

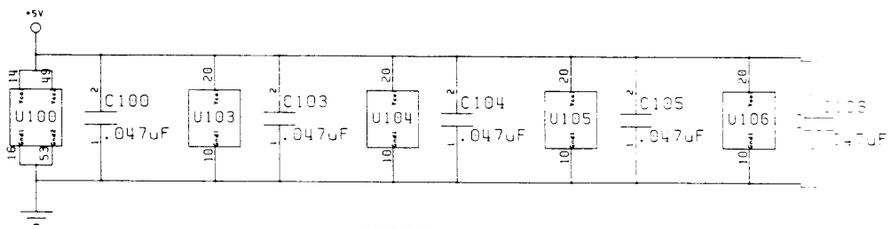


B2000-2

I'm pickin' up good vibrations...

UNLESS OTHERWISE SPECIFIED	DRAWN BY: Dave Maynie	DATE	Commodore
.X .XX .XXX ∠'S	CHKD:		B2000
	ENGR: DBM		Audio
MATERIAL:	USED ON B2000-CA	NEXT ASSY 312725	SIZE B 312726 REV 2
FINISH:			SCALE SHEET 2 OF 12





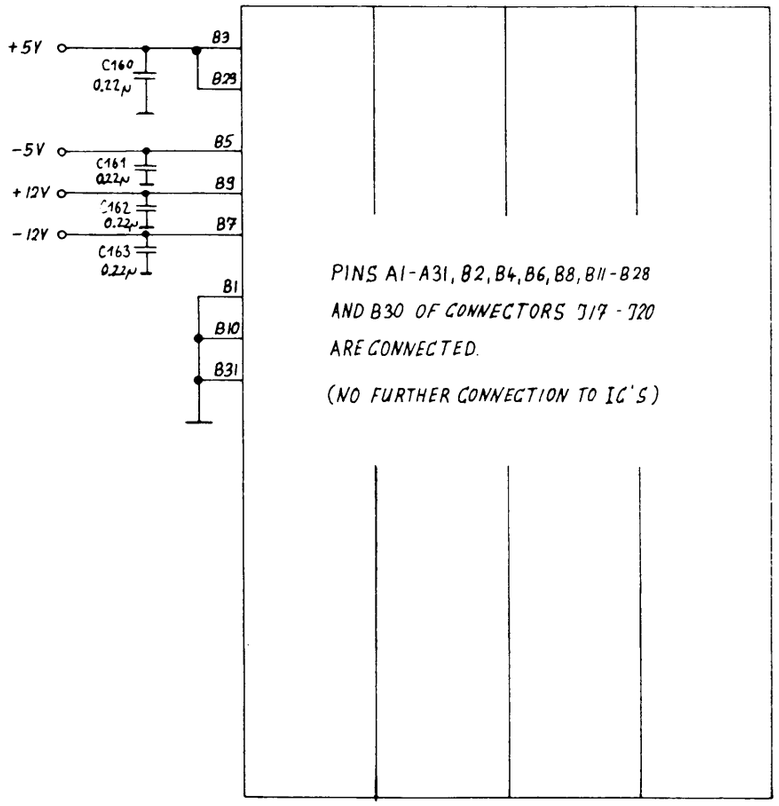
B2000-1

Something in the way she moves..

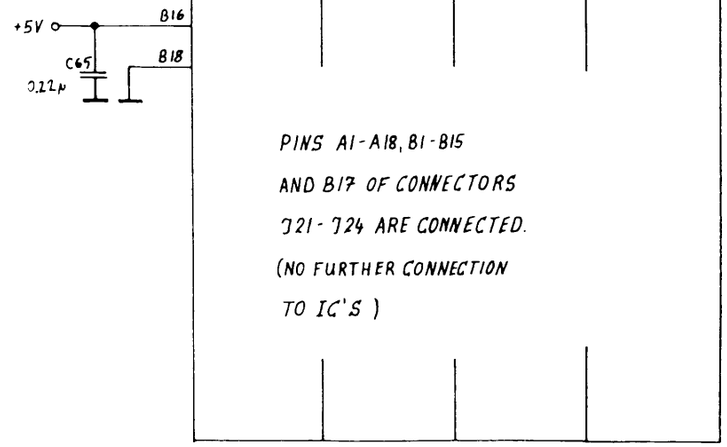
UNLESS OTHERWISE SPECIFIED: .X ±, .XX ±, .XXX ±, ∠'S ±	DRAWN BY: Dave Haynie	DATE 4/29/87	Commodore
	CHKD:		B2000
	ENGR: DBH	4/29/87	Processor
	APPR:		
MATERIAL:	USED ON: B2000-CM	NEXT ASST 312725	SIZE B 312720 REV 2
FINISH:			SCALE SHEET 1 OF 12

REVISIONS				
LTR	ZONE	DESCRIPTION	DATE	APPROVED
		SEE SHEET 1		

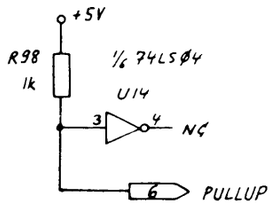
J29 - J32
2 x 31 PINS

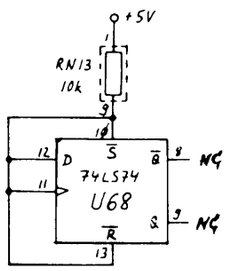


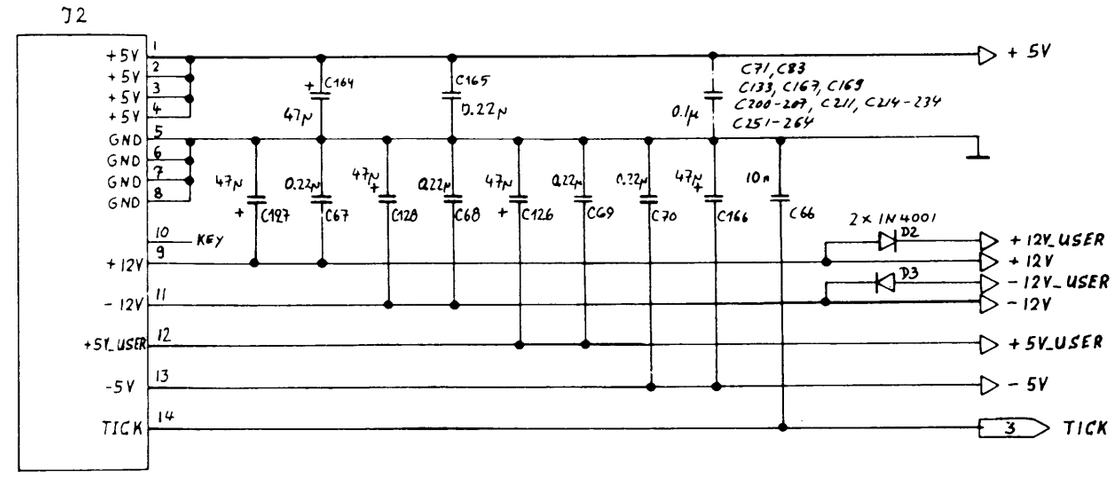
J25 - J28
2 x 18 PINS

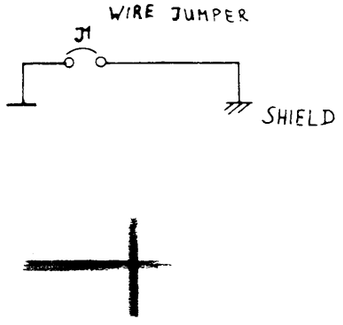


SPARE:





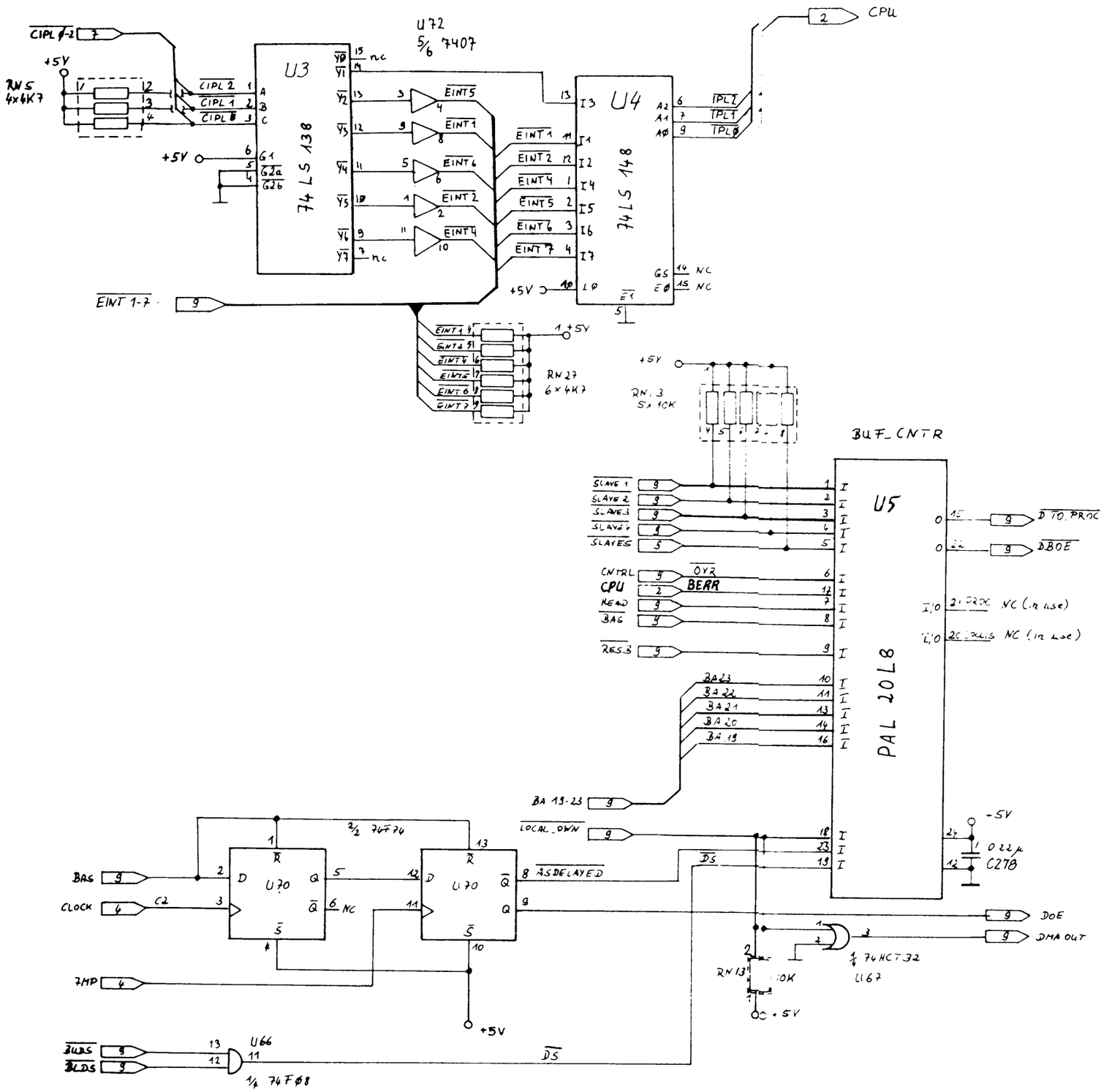


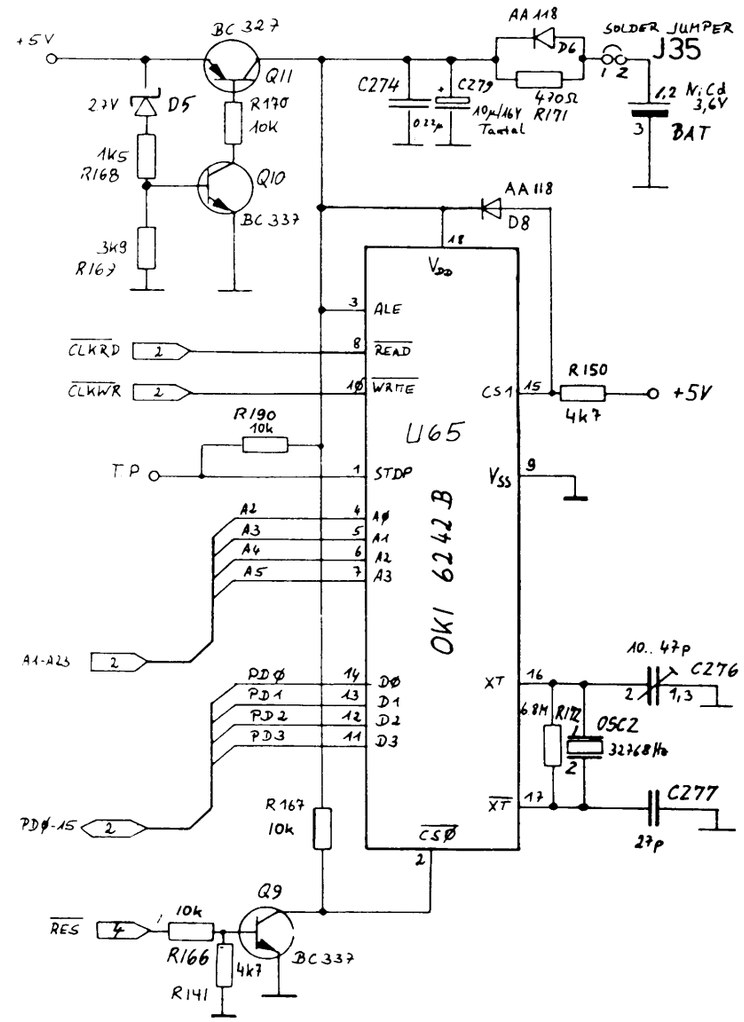


A2000-10

UNLESS OTHERWISE SPECIFIED TOLERANCES ON DECIMALS .X ± .XX ± .XXX ± ∠'S ±	DRAWN BY:	DATE 05-28-86	commodore
	CHKD:		SCHEMATIC AMIGA 2000 3-58
	ENGR: [illegible]	05-28-86	
	APPR:		
MATERIAL: FINISH	USED ON A2000	NEXT ASSY	SIZE C 380710 REV 6
			SCALE NONE SHEET 11 OF 17

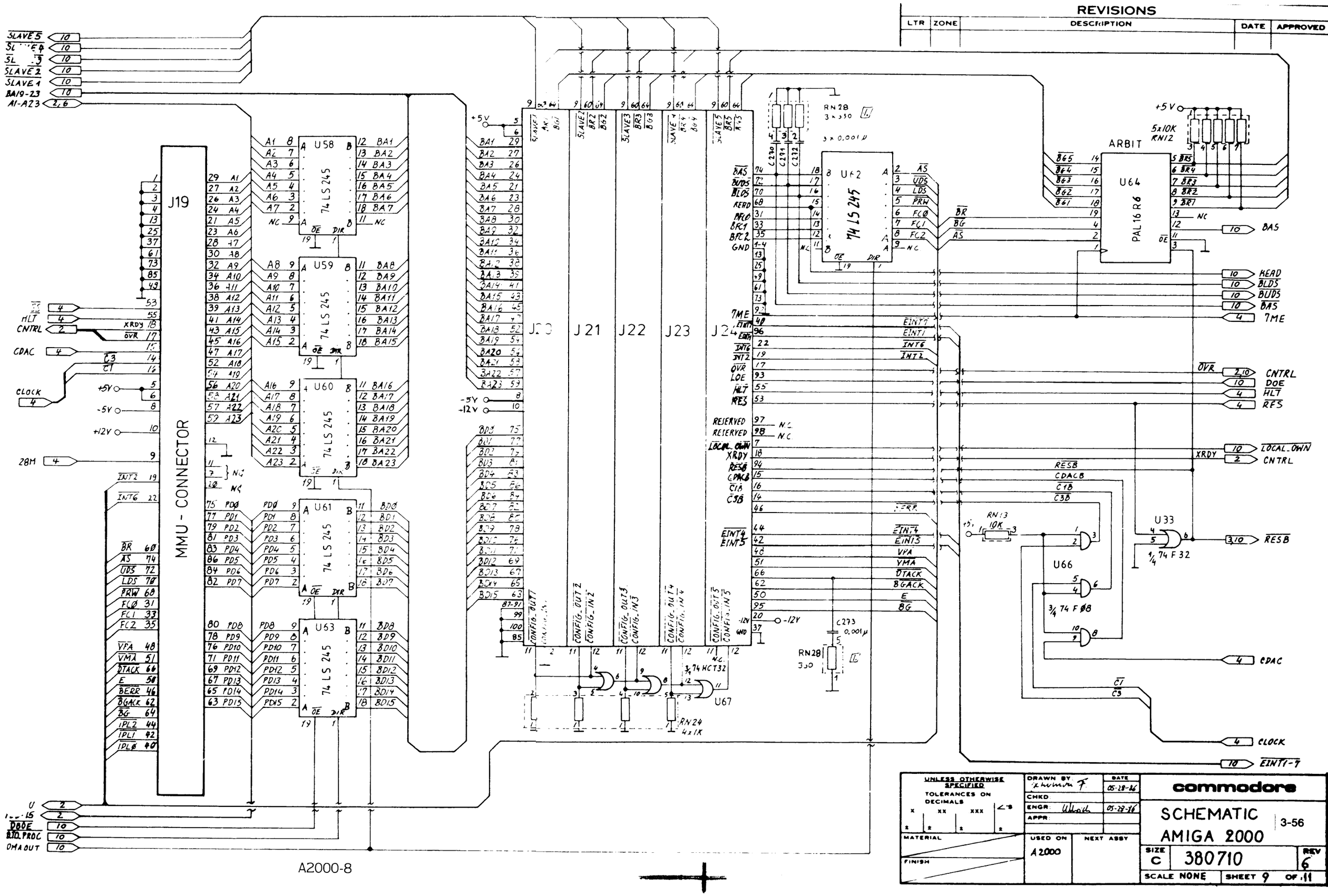
REVISIONS				
LTR	ZONE	DESCRIPTION	DATE	APPROVED



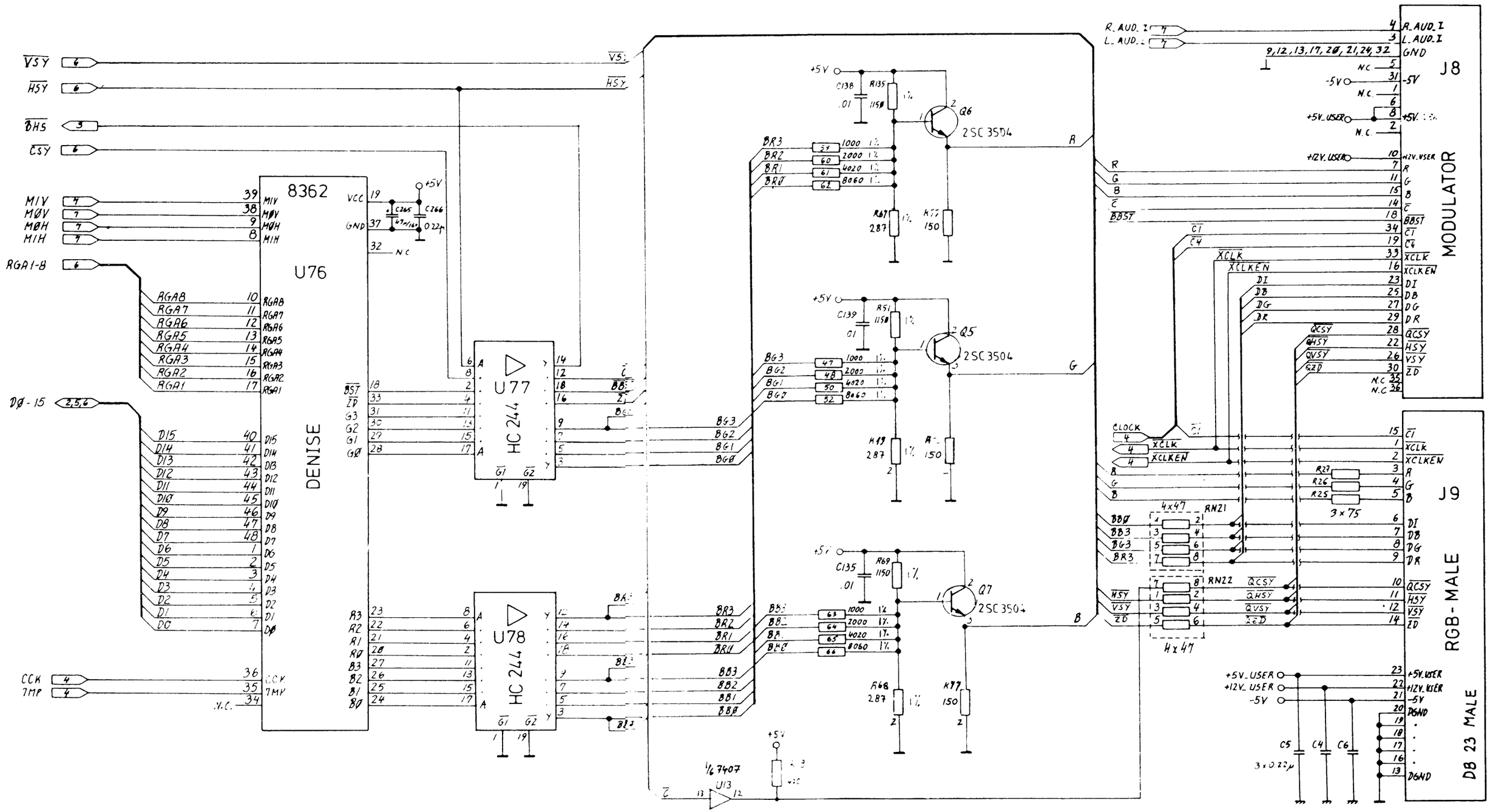


A2000-9

UNLESS OTHERWISE SPECIFIED TOLERANCES ON DECIMALS .X .XX .XXX ∠'S	DRAWN BY:	DATE 05-28-86	commodore
	CHKD:		SCHEMATIC AMIGA 2000 3-57
	ENGR:	05-28-86	
	APPR:		
MATERIAL	USED ON A2000	NEXT ASSY	SIZE C 380 710 REV 6
FINISH			SCALE NONE SHEET 10 OF 11


REVISIONS
LTR
ZONE
DESCRIPTION
DATE
APPROVED
MMU - CONNECTOR
J19
J20
J21
J22
J23
J24
U58
U59
U60
U61
U63
74 LS 245
U62
74 LS 245
U64
PAL 16 R6
ARBIT
U33
1/4 74 F 32
U66
3/4 74 F Ø8
U67
3/4 74 HCT32
RN28
3 x 330
3 x 0,001 µ
5x10K
RN12
RN13
10K
RN24
4x1K
C273
0,001µ
SLAVE 5
SLAVE 4
SLAVE 3
SLAVE 2
SLAVE 1
BA19-23
A1-A23
HLT
CNTRL
CDAC
CLOCK
28M
BAS
HEAD
BLDS
BUDS
7ME
CNTRL
DOE
HLT
RES
LOCAL_OWN
XRDY
RESB
CDAC
CLOCK
EINT1-7
DBOE
DMA OUT
A2000-8
UNLESS OTHERWISE SPECIFIED
TOLERANCES ON DECIMALS
MATERIAL
FINISH
DRAWN BY
CHKD
ENGR
APPR
DATE
05-28-86
05-28-86
USED ON
A2000
NEXT ASSY
commodore
SCHEMATIC
AMIGA 2000
3-56
SIZE
C
380710
REV
6
SCALE NONE
SHEET 9 OF 11

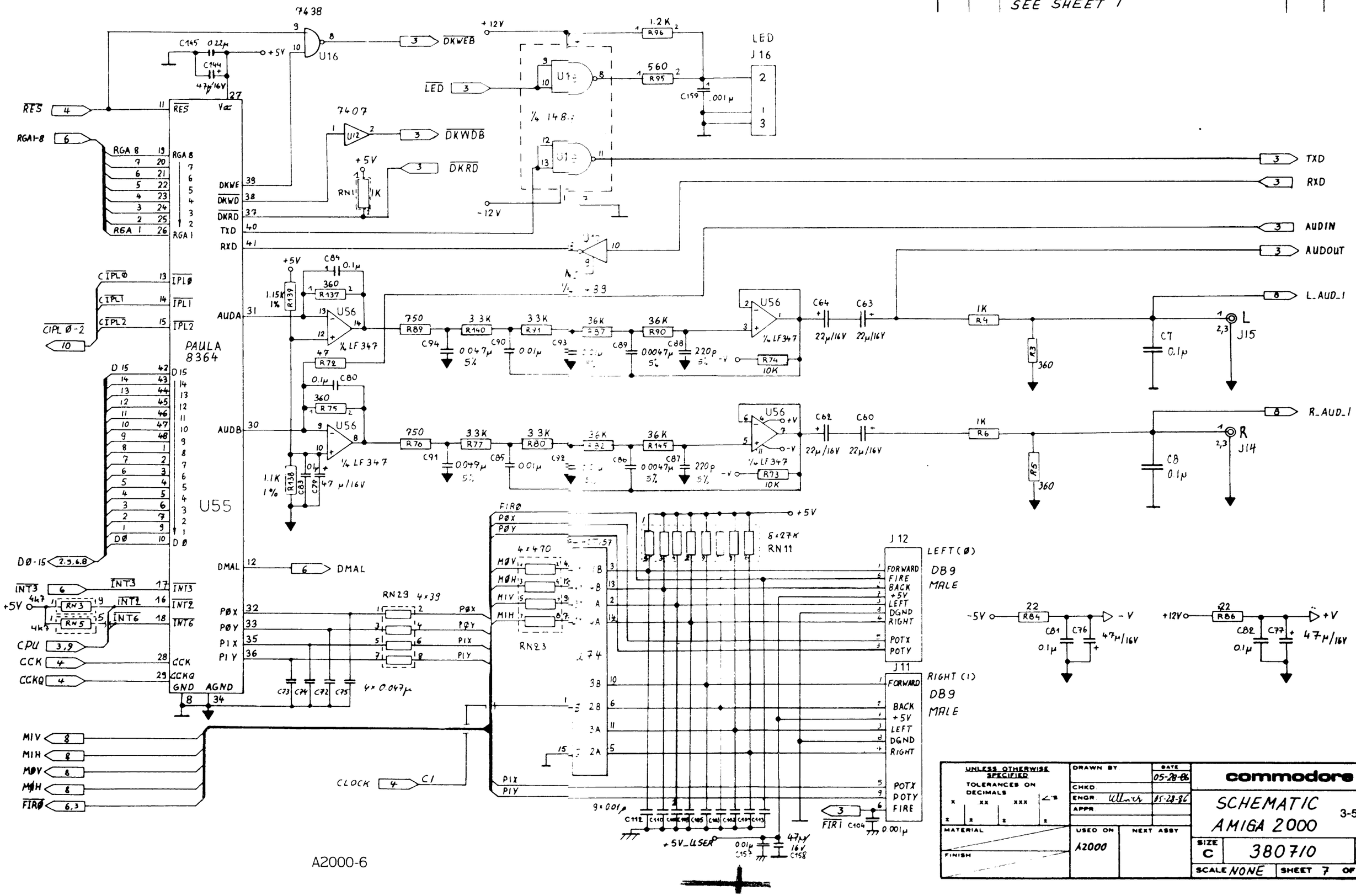
REVISIONS				
LTR	ZONE	DESCRIPTION	DATE	APPROVED



UNLESS OTHERWISE SPECIFIED TOLERANCES ON DECIMALS	DRAWN BY Zimmiri F	DATE 05.28.86	commodore
	CHKD		SCHEMATIC AMIGA 2000 3-55
	ENGR Ulrich	05-28-86	
	APPR		
MATERIAL	USED ON A2000	NEXT ASSY	SIZE C 380 710 REV 6
FINISH			SCALE NONE SHEET 8 OF 11

A2000-7

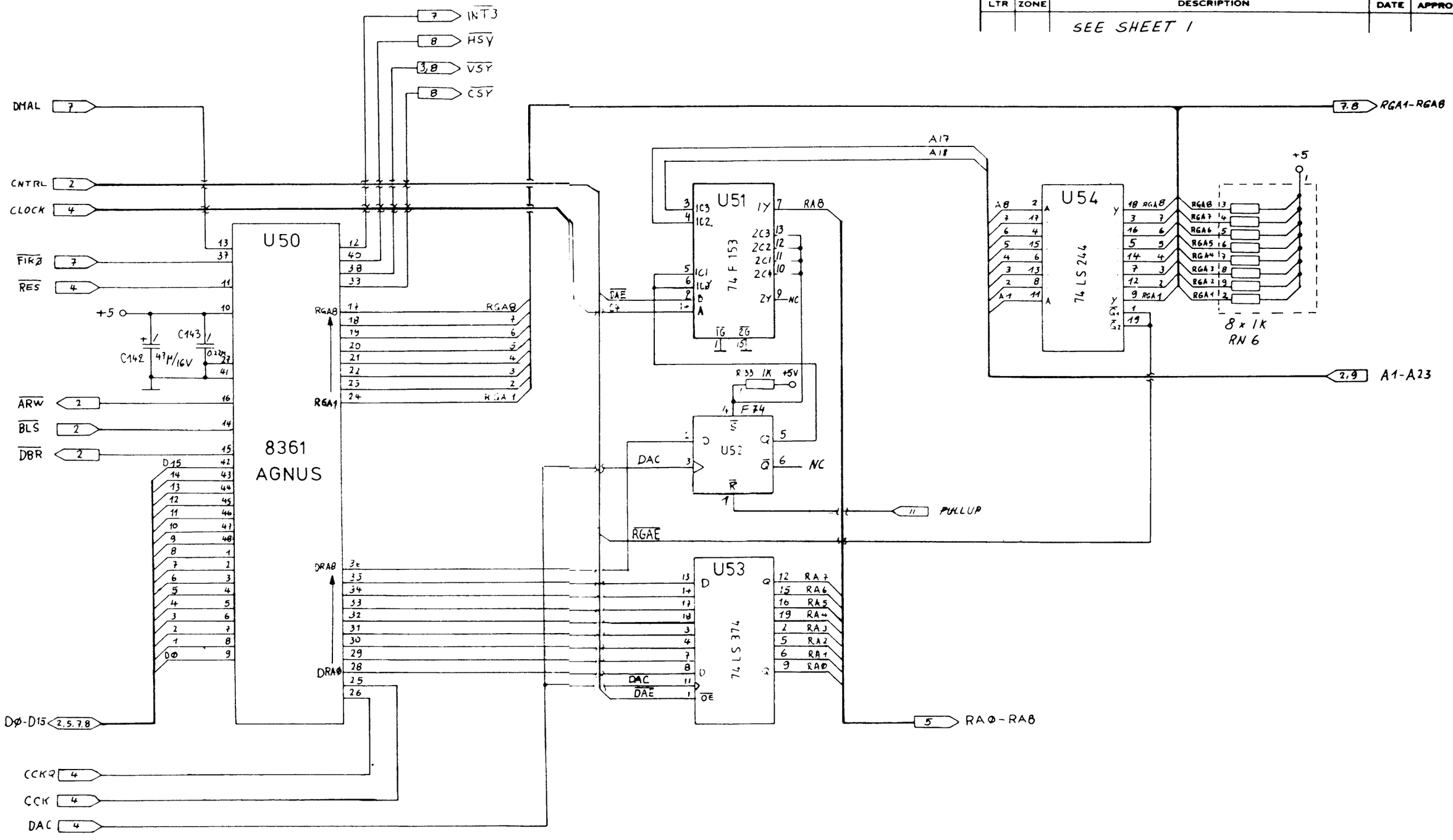
REVISIONS				
LTR	ZONE	DESCRIPTION	DATE	APPROVED
		SEE SHEET 1		



A2000-6

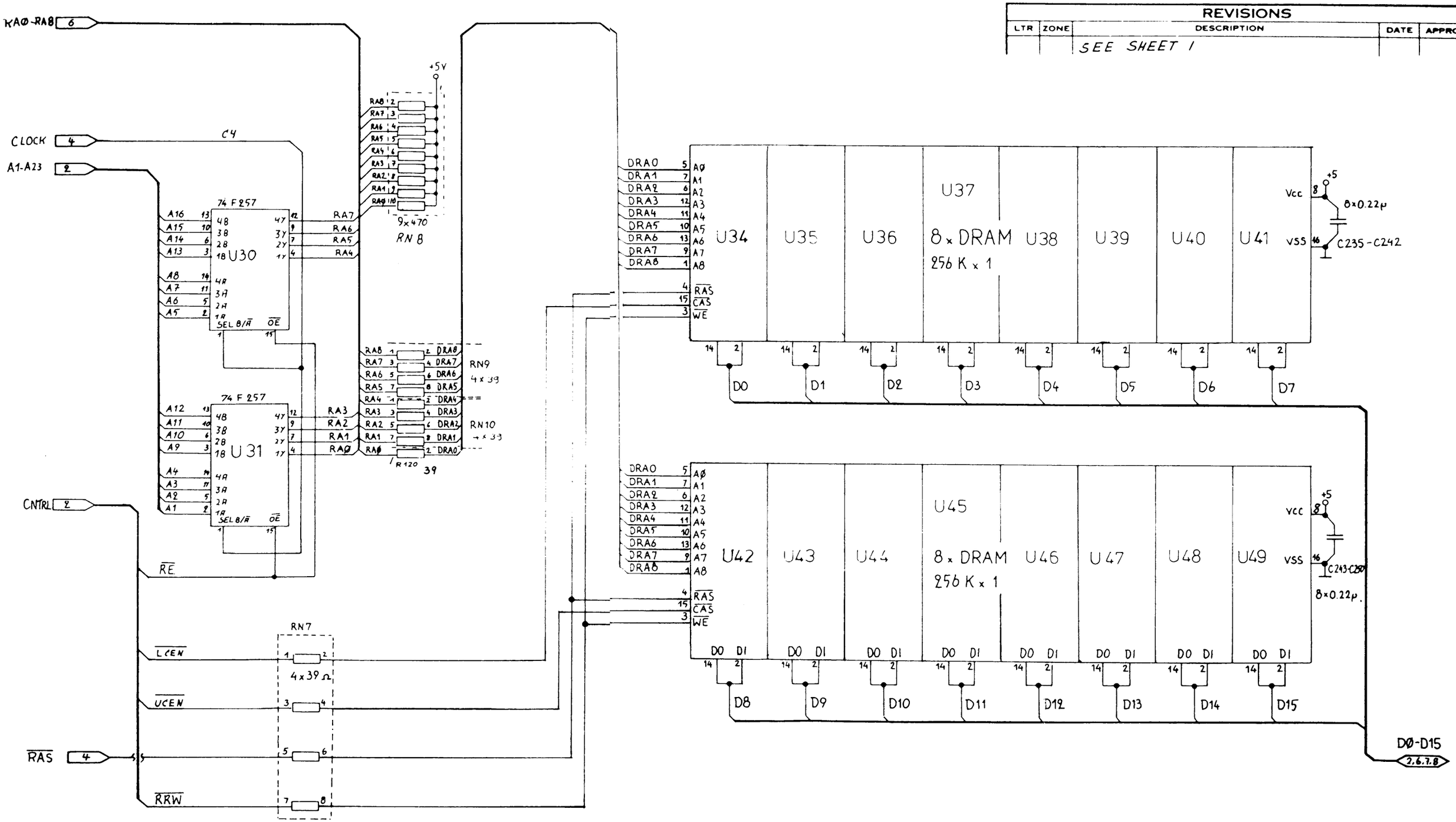
UNLESS OTHERWISE SPECIFIED TOLERANCES ON DECIMALS	DRAWN BY	DATE 05-28-86	commodore
	CHKD		SCHEMATIC AMIGA 2000 3-54
	ENGR Ullrich	05-28-86	
	APPR		
MATERIAL	USED ON A2000	NEXT ASSY	SIZE C 380710 REV 6
FINISH			SCALE NONE SHEET 7 OF 11

REVISIONS				
LTR	ZONE	DESCRIPTION	DATE	APPROVED
		SEE SHEET 1		



UNLESS OTHERWISE SPECIFIED TOLERANCES ON DECIMALS X ± XX ± XXX ± ∠° ±	DRAWN BY	DATE 05-28-86	commodore
	CHKD		SCHEMATIC AMIGA 2000 3-53
	ENGR Tillock	05-28-86	
	APPR		
MATERIAL / FINISH	USED ON A2000	NEXT ASSY	SIZE C — 380 710 — REV 6
			SCALE NONE — SHEET 6 OF 11

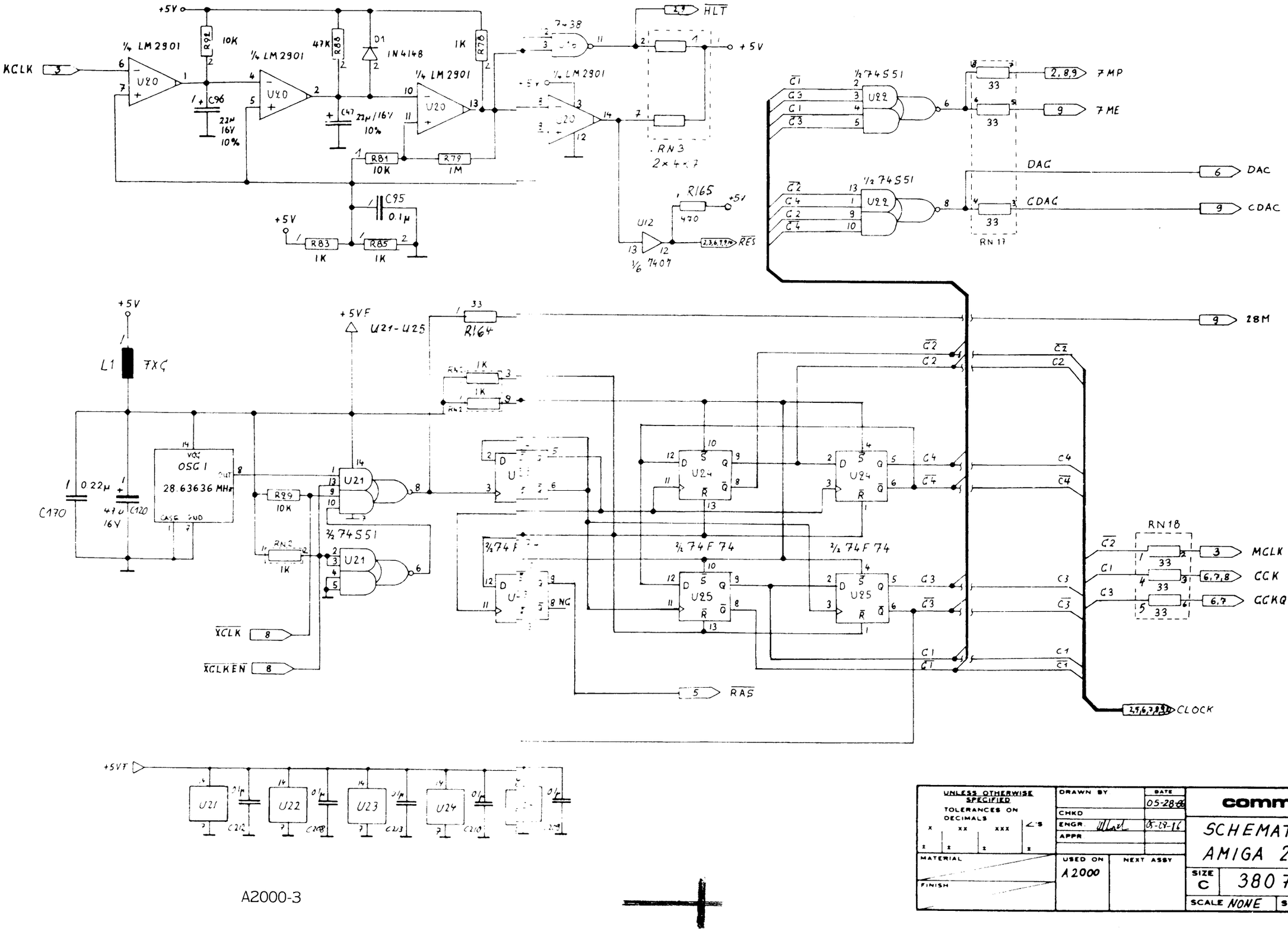
REVISIONS				
LTR	ZONE	DESCRIPTION	DATE	APPROVED
		SEE SHEET 1		



UNLESS OTHERWISE SPECIFIED TOLERANCES ON DECIMALS	DRAWN BY	DATE	commodore
	CHKD	05-28-86	SCHEMATIC AMIGA 2000 3-52
	ENGR	05-28-86	
	APPR		
MATERIAL	USED ON A2000	NEXT ASSY	SIZE C 380 710 REV 6
FINISH			SCALE NONE SHEET 5 OF 11

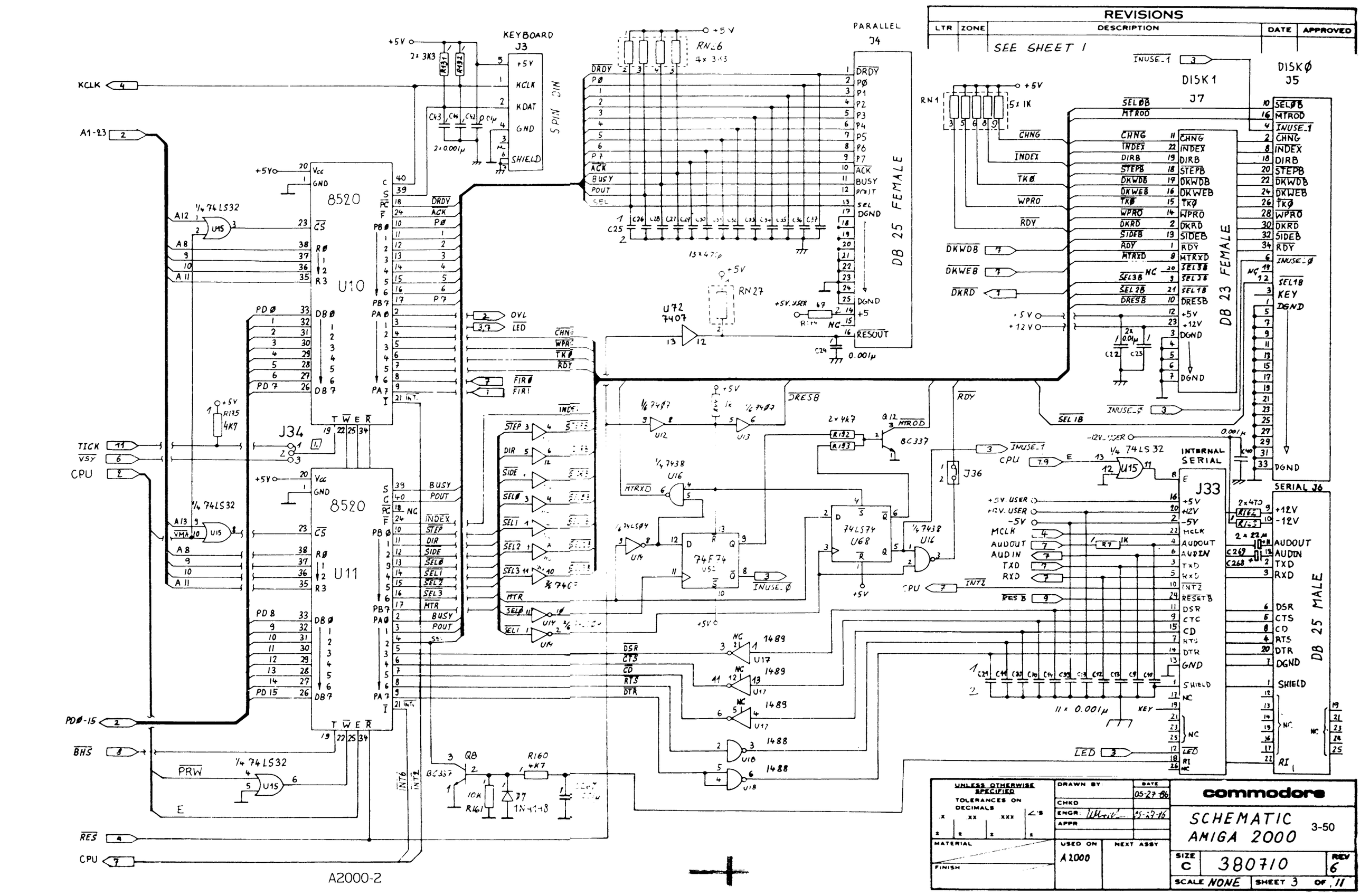
A2000-4

REVISIONS				
LTR	ZONE	DESCRIPTION	DATE	APPROVED
		SEE SHEET 1		

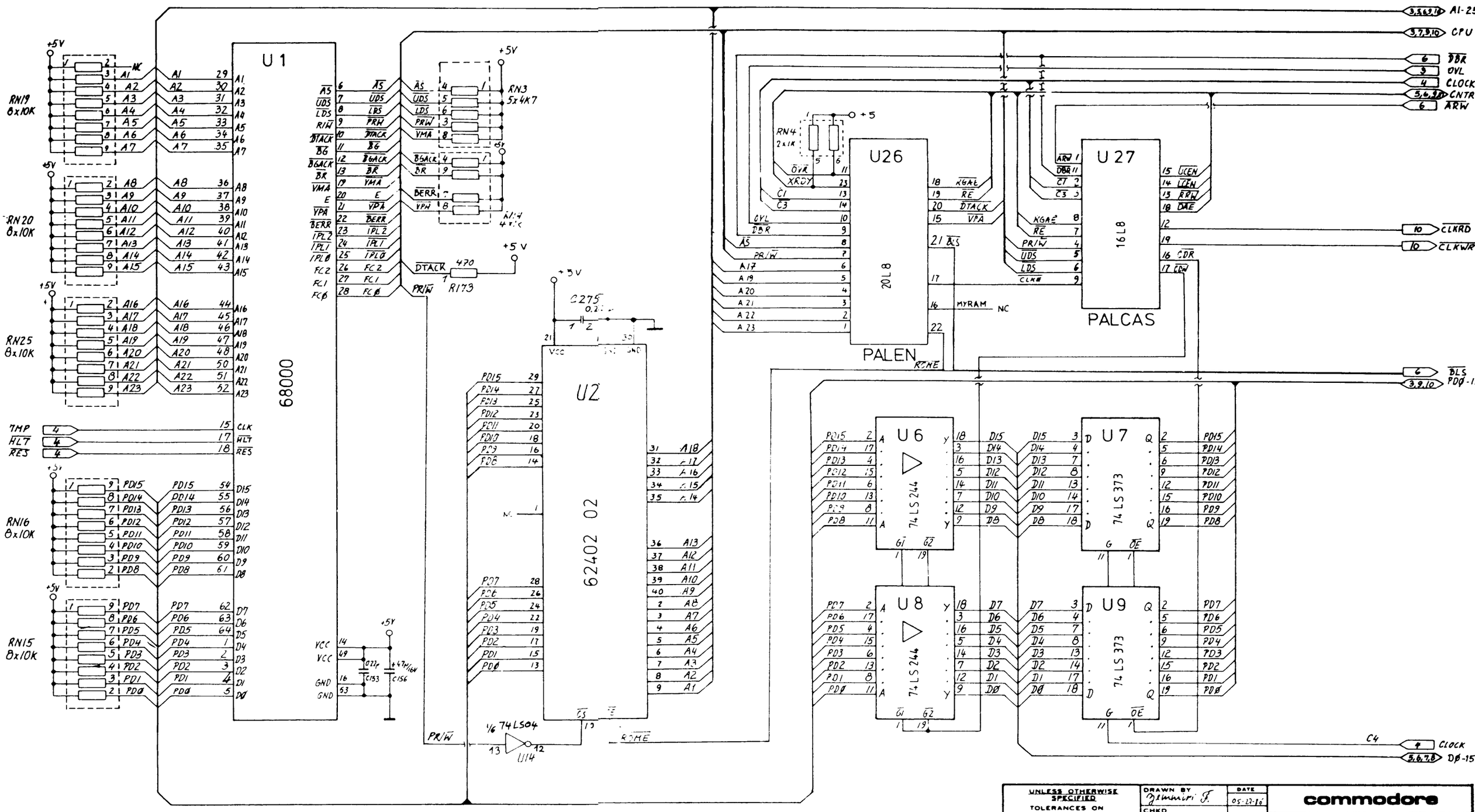


A2000-3

UNLESS OTHERWISE SPECIFIED TOLERANCES ON DECIMALS x ± xx ± xxx ± ∠ ±	DRAWN BY	DATE 05-28-86	commodore
	CHKD		SCHEMATIC AMIGA 2000 3-51
	ENGR	05-29-86	
	APPR		
MATERIAL / FINISH	USED ON A2000	NEXT ASSY	SIZE C · 380710 · REV 6
			SCALE NONE · SHEET 4 OF 11


REVISIONS
LTR
ZONE
DESCRIPTION
DATE
APPROVED
SEE SHEET 1
KEYBOARD
J3
5 PIN DIN
PARALLEL
J4
DB 25 FEMALE
DISK 1
J7
DISKØ
J5
DB 23 FEMALE
8520
U10
U11
J34
J36
INTERNAL SERIAL
J33
SERIAL J6
DB 25 MALE
74F74
U52
74LS74
U68
U72
7407
RN26
RN27
RN1
KCLK
A1-23
TICK
VSY
CPU
PDØ-15
BHS
RES
A2000-2
UNLESS OTHERWISE SPECIFIED
TOLERANCES ON DECIMALS
MATERIAL
FINISH
DRAWN BY
DATE
05-27-86
CHKD
ENGR
APPR
USED ON
A2000
NEXT ASSY
commodore
SCHEMATIC
AMIGA 2000
3-50
SIZE
C
380710
REV
6
SCALE NONE
SHEET 3 OF 11

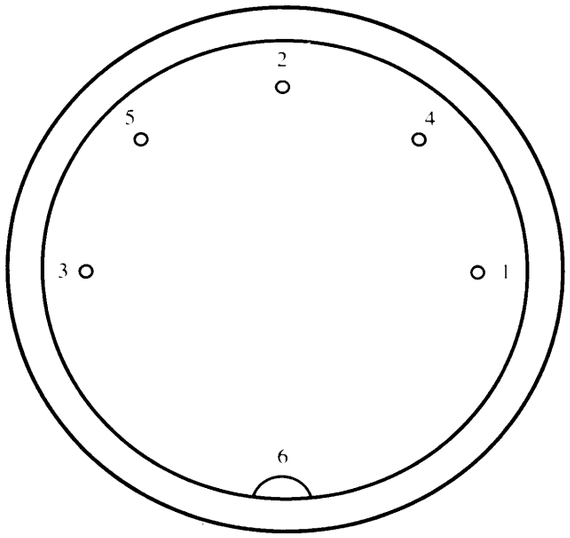
REVISIONS				
LTR	ZONE	DESCRIPTION	DATE	APPROVED



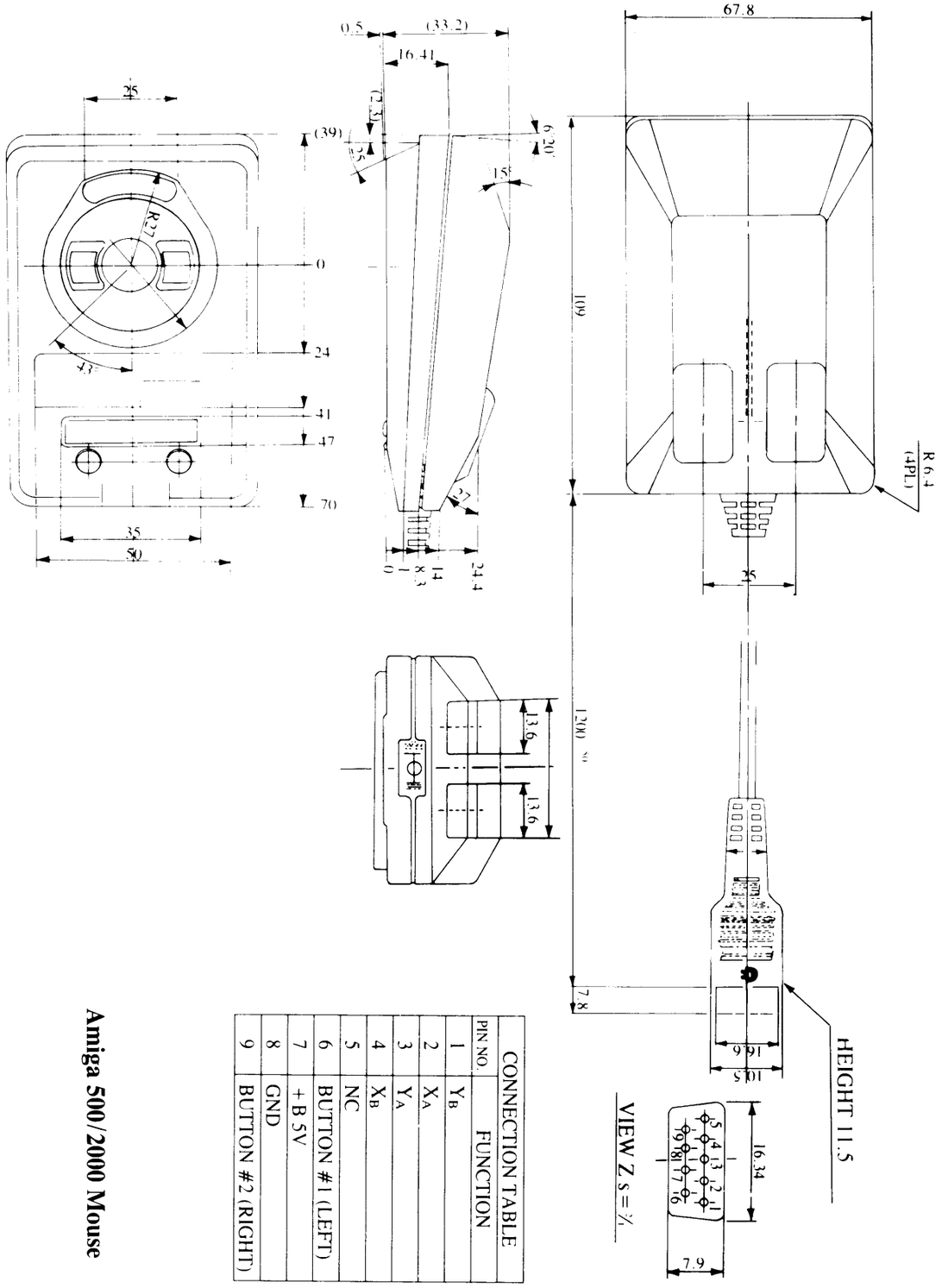
UNLESS OTHERWISE SPECIFIED TOLERANCES ON DECIMALS	DRAWN BY	DATE	commodore
	CHKD	05-27-86	SCHEMATIC AMIGA 2000 3-49
	ENGR.	05-27-86	
	APPR.		
MATERIAL	USED ON A2000	NEXT ASSY	SIZE C 380710
FINISH			SCALE NONE SHEET 2 OF 11

A2000-1

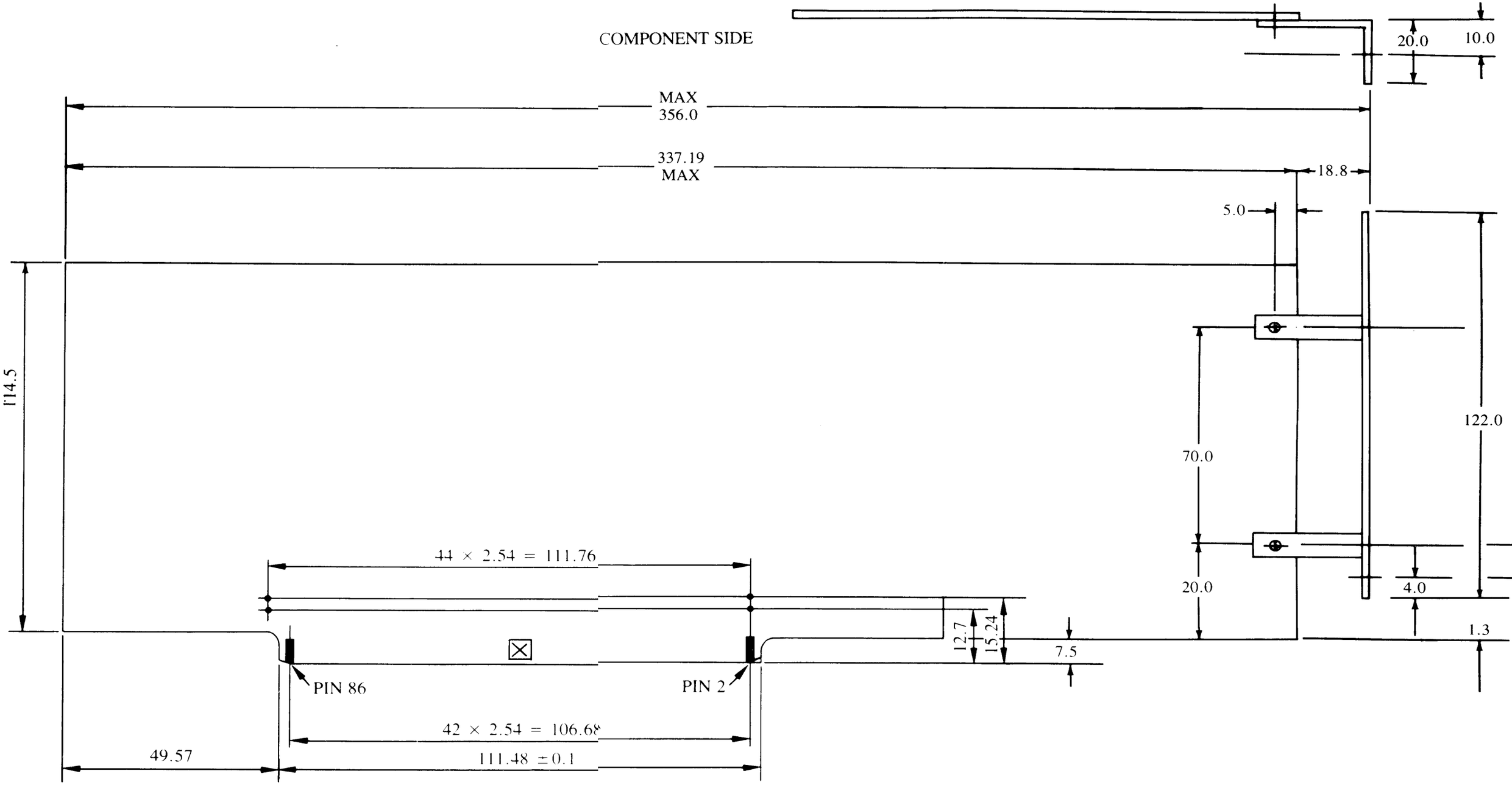
A2000
Keyboard Connector



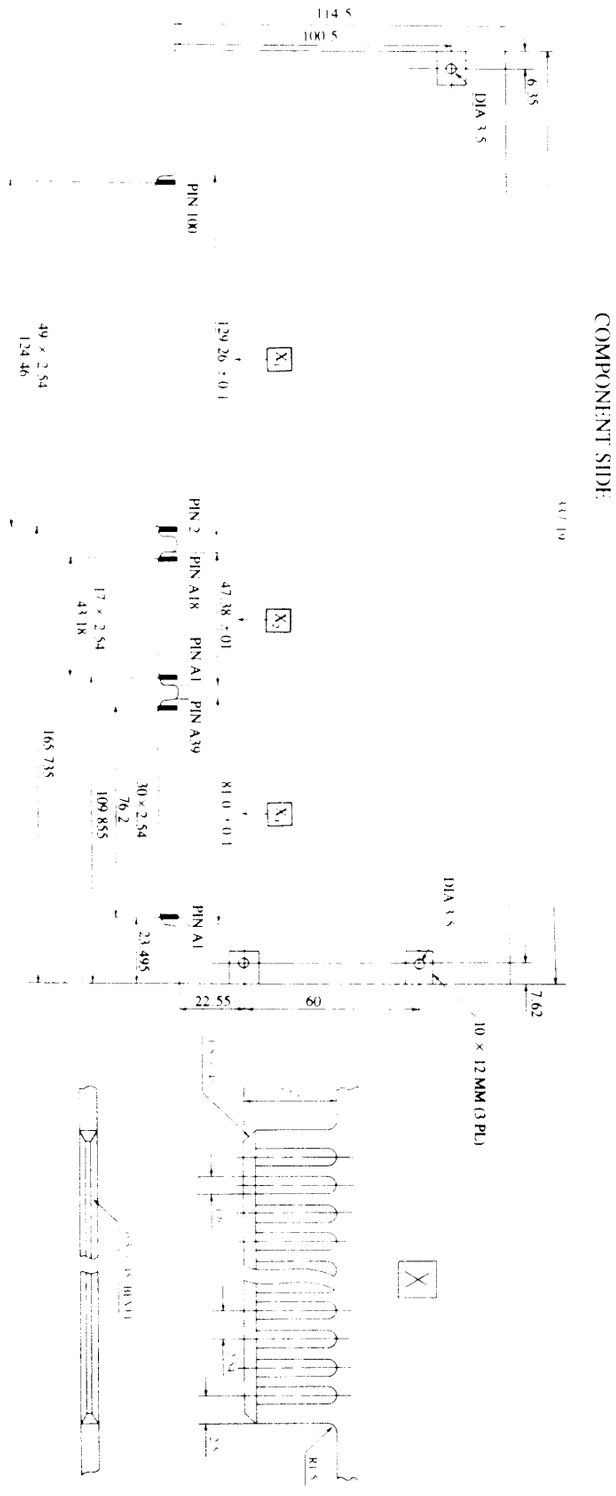
Pin	Name	Description
1	KCLK	Keyboard clock
2	KDAT	Keyboard data
3	NC	Not connected
4	GND	Ground
5	+5V	+5 Volts power (100 mA)
6	SHIELD	Shield



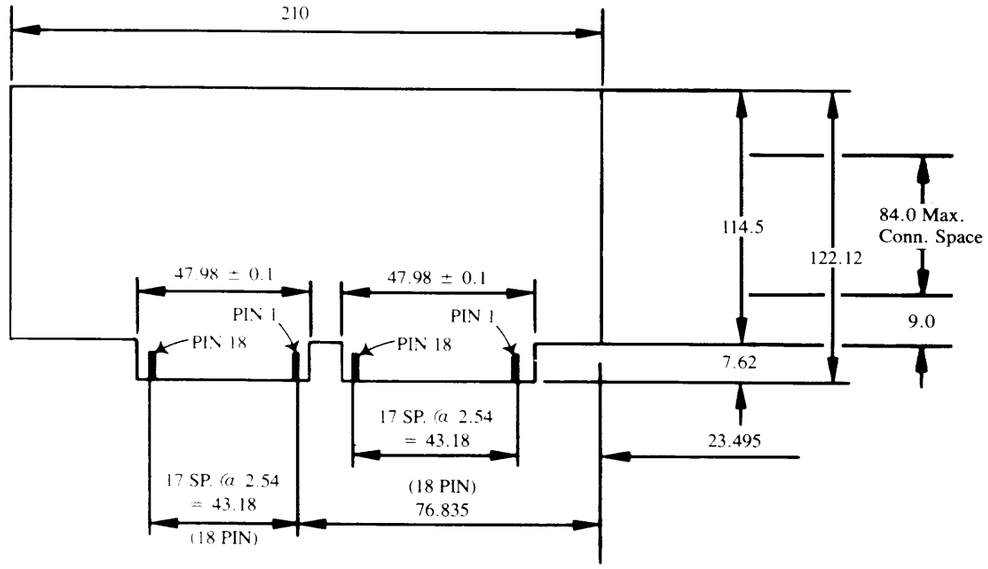
CONNECTION TABLE	
PIN NO.	FUNCTION
1	Y_B
2	X_A
3	Y_A
4	X_B
5	NC
6	BUTTON #1 (LEFT)
7	+B 5V
8	GND
9	BUTTON #2 (RIGHT)



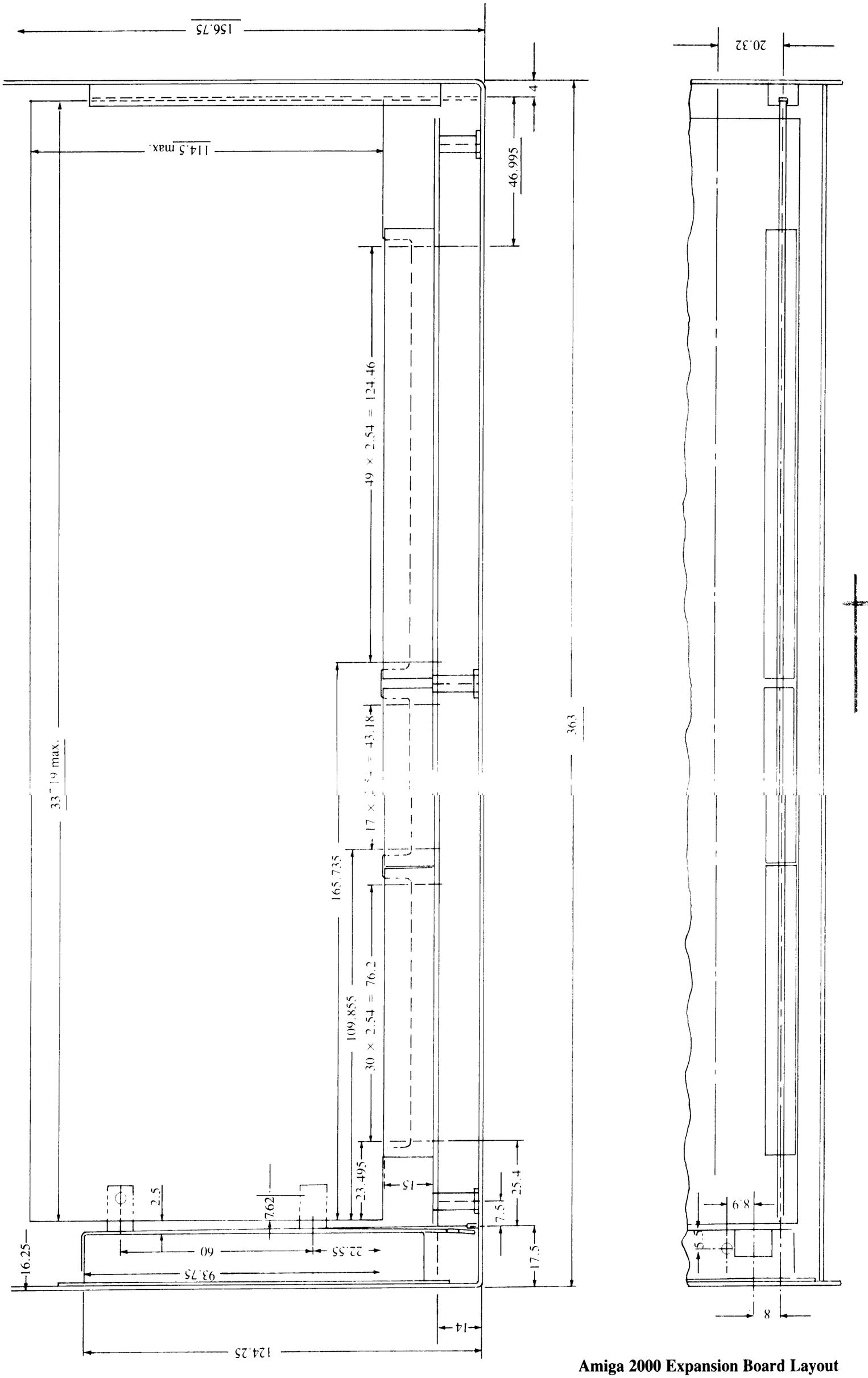
86-Pin Slot Expansion Board



Amiga 2000 Form Factor



Amiga 2000 Video Card



Amiga 2000 Expansion Board Layout