ASTEC VDE

Gutachten mit Fertigungsüberwachung der VDE-Prüfstelle

Nr. 1633 **ÜG** Blatt 1

2/4

VDE-Prüfstelle, Merianstraße 28, D-6050 Offenbach

Nur gültig mit umseitigen Bedingungen

Name und Sitz des Gutachteninhabers

ASTEC Europe Ltd., Eduard-Pfeiffer-Str. 73, 7000 Stuttgart 1

Fertigungsstätte

-AA- ASTEC International Ltd., Hung Hom Kowloon, HK Hong Kong

Zeichen des Antragstellers	Antragsdatum	Aktenzeichen	Ausstellungsdatum
Be/rop	Schr.v. 18.6.1984	11774-3336-1001/A1J Wö/b	27.6.1984

Überwachungs-Kennzeichen:

Statistik: 1

„VDE-Reg.-Nr. 1633" oder 1633

Beschreibung:

Jahres-gebühren-Einheiten

Geprüft nach DIN IEC 380/VDE 0806/8.81

Einbauschaltnetzteil

Bezeichnung:	Modell AC 8151-01 V	25
Nennspannung:	115/230 V~, 50/60 Hz	
Nennstrom:	0,5 A (für 230 V~) bzw. 1 A (für 115 V~)	
Ausgangsspannungen und -ströme:	Siehe Datenblatt-Anlage Nr. 1	
Zulässige Umgebungstemperatur:	max. 50 °C	

Beim Einbau des Schaltnetzteiles, der entsprechend der zugehörigen Installationsanleitung zu erfolgen hat, muß sichergestellt werden, daß alle Anforderungen gemäß den Bestimmungen DIN IEC 380/VDE 0806/8.81 eingehalten sind.

Weitere Angaben vergleiche Anlage Nr. 1 - 4.

25,C

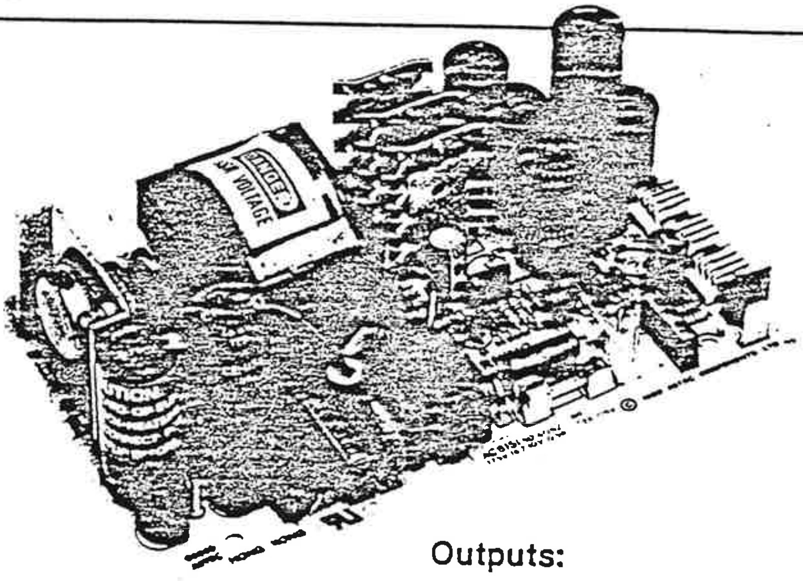
======

VDE-Prüfstelle
Abt. TD

i.A. Jakubski-Weihl

ASTEC

AC 8151-01
38 Watt Switching Power Supply



DESCRIPTION

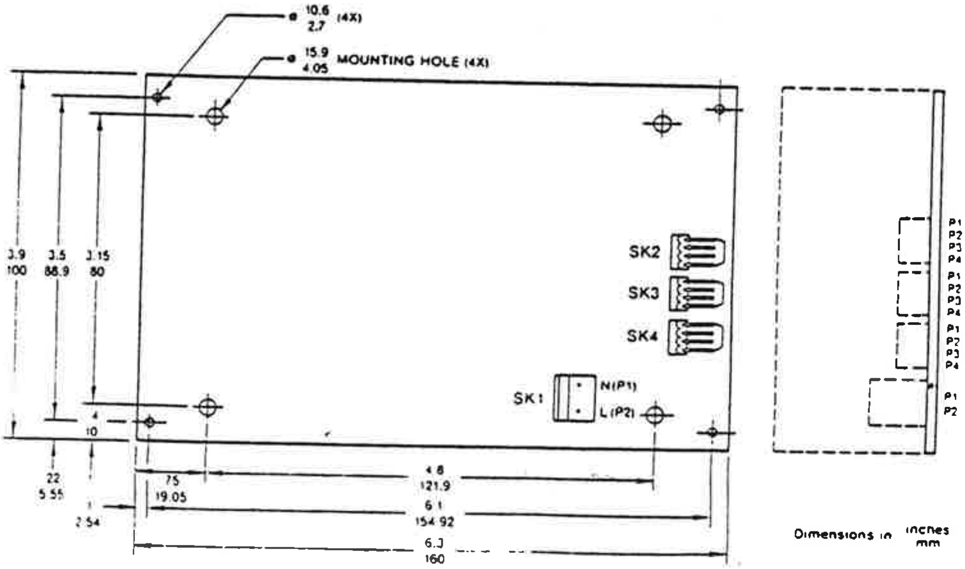
The AC 8151 is a 38 Watt, multiple-output switching power supply, produced on an open printed circuit board. It is ideally suited for use in small microprocessor based systems, disc drive systems, terminals, and other mixed logic applications. Input voltage is jumper selectable for either 115 VAC or 230 VAC. The AC 8151 is UL, CSA, and VDE approved.

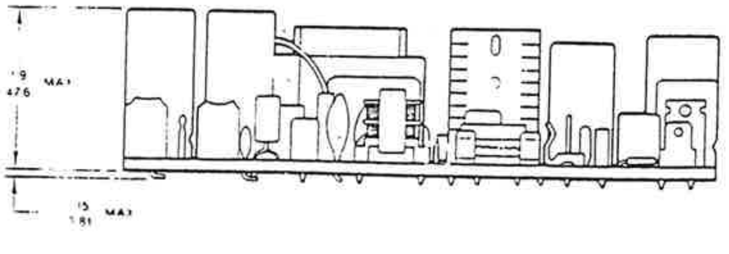
Outputs:

+5V DC @ 2.5A
+12V DC @ 2.0A
−12V DC @ 0.1A

FEATURES

- ☐ High Efficiency
- ☐ Built-in EMI filter
- ☐ UL/CSA/VDE approved
- ☐ Meets European safety standards
- ☐ 100% thermal cycle and burn-in
- ☐ Vacuum impregnated transformers
- ☐ Dual input voltage 115/230 VAC
- ☐ Convection cooling
- ☐ Over voltage protection
- ☐ Short circuit protection
- ☐ Open PCB construction





PIN ASSIGNMENT

SK1 P1 Neutral
P2 Live

SK2/3/4
P1 −12V
P2 +12V
P3 Common
P4 +5V

AC 8151-01 ELECTRICAL CHARACTERISTICS

Output Characteristics

Output current capacity:

Output Voltage	Load		Tolerance[1]	Output Ripple[2]
	min	max		
+ 5 VDC	0.45A	2.5A	± 4%	50mV p-p
+ 12 VDC	0.3A	2.02A	± 5%	150mV p-p
− 12 VDC	0.04A	0.1A	+ 25%/ − 8.3%	150mV p-p

Input Characteristics

AC Input voltage:	95 to 135 VAC, or 190 to 270 VAC (Selectable by jumper on PCB)
AC Input frequency:	47 to 63Hz
AC Input current:	0.85A (rms)

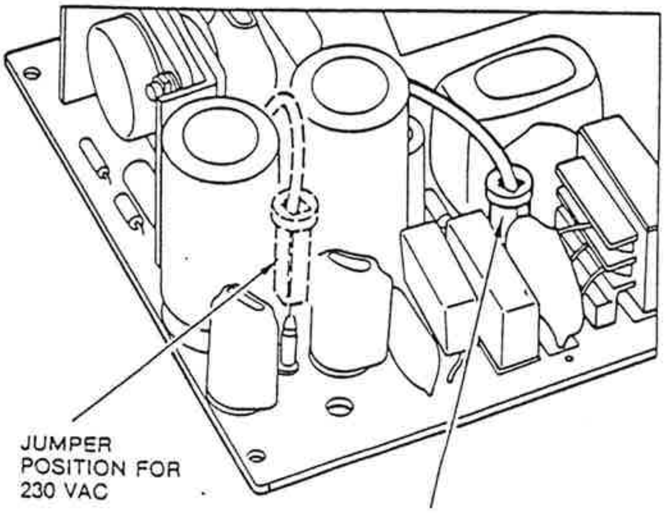
NOTES:

1. *The output tolerance refers to the nominal voltage and includes line regulation, load regulation, temperature drift and set-up tolerance.*
2. *The specified ripple is at the rated line voltage and load range.*

Individual maximum ratings:

+ 5V DC:	5 amps if no load on + 12V output, or 2.5 amps if load draws 2.5 amps on + 12V output.
+ 12V DC:	2.5 amps if + 5V output load draws 2.5 amps or less.
− 12V DC:	0.5 amps

NOTE: The maximum continuous output power shall not exceed 38 Watts. Specified voltage tolerances do not apply to the above individual maximum currents.



DUAL AC INPUT VOLTAGES
(JUMPER SELECTABLE)

General Characteristics

Operating temperature:	0 to 50 °C
Efficiency:	65% (min) at full load
Input line regulation:	± 0.2% max.
Overvoltage protection:	5.8V min. to 6.8V max. (+ 5V line only)
Hold-up time:	15msec.
Line regulation:	± 0.5% high line at full load
Overcurrent protection:	All outputs protected to short circuit conditions
Line transient response:	Meets IEEE standards
Power line disturbances:	Output supply unaffected through half cycle absence of input power during full load and 95 VAC input.
EMI Requirements:	Meets conduction limits of: a) FCC Class "B" rules b) VDE 0871 Class "B" rules
Safety requirements:	Meets or exceeds: a) UL 1012 (File #E69016) b) CSA 22.2 (File #LR42983-22) c) IEC 435 d) CEE: Part 1 and Part 2P
Inrush current:	30 amps @ 115 VAC, or 60 amps @ 230 VAC at 25 °C ambient cold start.

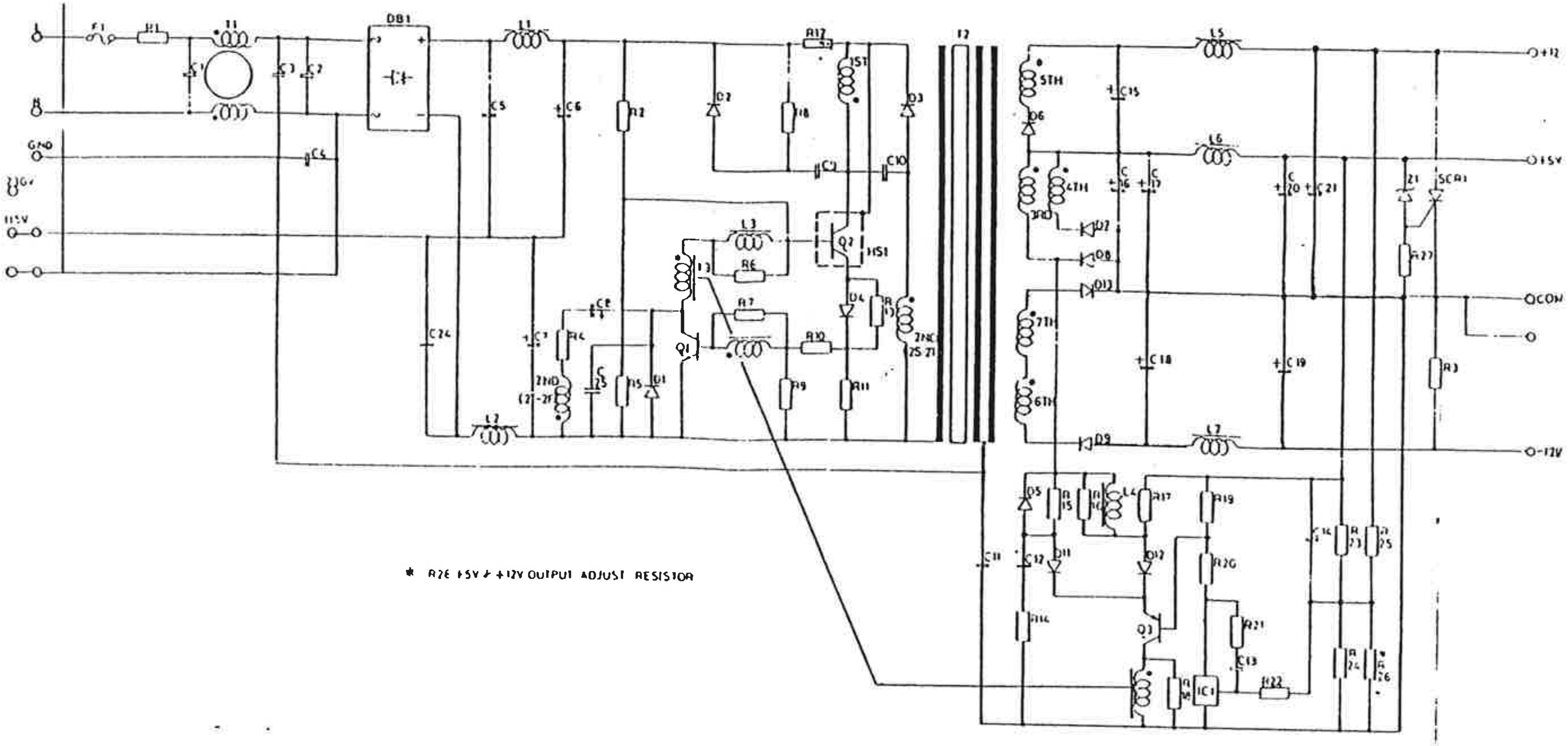
Mechanical Characteristics

Size (Overall):	length: 6.3 in. (160 mm) width: 3.9 in. (100 mm) height: 2.0 in. max (51.4 mm)
Weight:	15.17 oz (0.43 kg)
Mating connectors:	
AC input:	Molex P/N 09-50-3031
DC Output:	Molex P/N 22-01-1043

ASTEC AGENCIES LTD
Taipei Liaison office
3F 1 45 Sec 2 Fu-Hsing S Rd
Taipei Taiwan R O C

ASTEC
ASTEC (USA), LTD.
2880 San Tomas Expressway, Suite 200
Santa Clara, CA 95051 U.S.A.
TEL (408) 748-1200

R	R1 R4 R2 R5 R6 R7 R8 R9 R10 R11 R12 R13 R14 R15 R16 R17 R18 R19 R20 R21 R22, R23 R24, R25 R26 R27 R3
C	C1 C3 C2 C4 C24 C5 C6 C7 C25 C8 C9 C10 C11 C12 C15 C16 C17 18 19 C13 C20, 19 21 14
L/T	T1 L1 L2 T3 T2 L7 L5 L6
Q/D	D1 Q1 D2 Q2 D4 D3 D6 D5 D11 D7 8 9 13 12 Q3
SC	F1 DB1 HS1 IC1 Z1 SCR1



* R26 +5V & +12V OUTPUT ADJUST RESISTOR

	CHK	DATE	REVISIONS	ZONE	ISSUE	ECN
[illegible]	[illegible]	[illegible]	REVISED DRAWING		6	[illegible]
[illegible]	[illegible]	[illegible]	REVISED DRAWING		5	[illegible]
[illegible]	[illegible]	[illegible]	REVISED DRAWING			[illegible]
[illegible]	[illegible]	[illegible]	REVISED DRAWING		3	[illegible]
[illegible]	[illegible]	1.9.80	RELEASE DRAWING		2	
[illegible]	[illegible]	8.8.80	PRELIMINARY		1	

UNLESS OTHERWISE SPECIFIED ALL DIMENSIONS ARE IN MM TOLERANCES WHOLE NO ± ANGLE ± DECIMAL .X ± .XX ± ✓	DO NOT SCALE DRAWING DRN Y. F. CHAN DATE 8.8.80 CHK [illegible] DATE [illegible] APPD [illegible] DATE [illegible]	TITLE CIRCUIT DIAGRAM	
		FIRST USED ON AC8151	PART NO
		MATERIAL	FINISH
ASTEC COMPONENT LTD.		HONG KONG	SH 1 OF 1

C1	Cap-MPR .01U +-20% 250 VAC
C2	Cap-MPR .1U +-20% 250 VAC
C3	Cap-C 4700p +-20% 400 VAC
C4	Cap-C 4700p +-20% 400 VAC
C5	Cap-MP 0.22u +-20% 250V
C6	Cap-E 100U +-20% 250V
C7	Cap-E 100U +-20% 250V
C8	Cap-E 220U +50 -10% 10V
C9	Cap-C 470p +-20% 3KV
C10	Cap-C .01U +-20% 1KV
C11	Cap-MPR .01U +-20% 250VAC
C12	Cap-P 0.22u +-20% 100V
C13	Cap-P .022U +-20% 50V
C14	Cap-P .22U +-20% 100V
C15	Cap-E 1000U +50 -10% 25V
C16	Cap-E 1000U +50 -10% 25V
C17	Cap-E 1000U +50 -10% 25V
C18	Cap-E 330U +-20% 16V
C19	Cap-E 330U +-20% 16V
C20	Cap-E 470U +50 -10% 25V
C21	Cap-E 2200UF +-20% 16V
C24	Cap-MP 0.22u +-20% 250V
D1	Rect RGP 10A
D2	Rect RGP 10J
D3	Rect RGP 10M
D4	Rect 1N4001 GP

PROPRIETARY STATEMENT

THIS DOCUMENT CONTAINS DATA AND INFORMATION PROPRIETARY TO ASTEC COMPONENTS LTD. THIS DATA AND INFORMATION SHALL NOT BE DISCLOSED. DISSEMINATION OR REPRODUCED IN WHOLE OR IN PART WITHOUT THE EXPRESS PRIOR WRITTEN AUTHORIZATION OF ASTEC COMPONENTS LTD.

A	27JUN83	E	
B		F	
C		G	
D		H	

TITLE

COMPONENT CODE LIST

AC8151

ASTEC

HONG KONG

Sheet 1 of 4

A4

D5	Diode-SI 1N4606
D6	Rect RG3B
D7	Rect RG3B
D8	Rect RG3B
D9	Rect RGP10B
D11	Diode-SI 1N4606
D12	Diode-SI 1N4606
D13	Rect 1N4001GP
DB1	Bridge Rect KBP10
F1	Fuse F2A 250V
IC1	IC
L1	Toroid
L2	Toroid
L3	Base choke
L4	Choke
L5	Choke Coil
L6	Choke Coil
L7	Choke Coil
Q1	TRS-NPN 2SD467
Q2	TRS-NPN 2SD467
Q3	TRS-PNP 2SD561

A	27JUN83	E		TITLE COMPONENT CODE LIST	AC8151
B		F			
C		G		ASTEC HONG KONG	Sheet 2 of 4 A4
D		H			

R1	THMTR 4R +-20%
R2	RES-CF 330K +-5% 1/2W
R3	RES-MOF 180R +-5% 1W
R4	RES-MOF 33R +-5% 2W
R5	RES-CF 1K +-5% 1/4W
R6	RES-CF 27R +-5% 1/4W
R7	RES-CF 68R +-5% 1/4W
R8	RES-MOF 120R +-5% 1W
R9	RES-CF 10R +-5% 1/4W
R10	RES-CF 10R +-5% 1/4W
R11	RES-MF .75R +-5% 1W
R12	RES-MF 1R +-5% 1W
R13	RES-CF 5.6R +-5% 1/4W
R14	RES-CF 68R +-5% 1/4W
R15	RES-CF 270R +-5% 1/2W
R16	RES-CF 270R +-5% 1/2W
R17	RES-CF 8.2R +-5% 1/4W
R18	RES-CF 560R +-5% 1/4W
R19	RES-CF 56R +-5% 1/4W
R20	RES-CF 68R +-5% 1/4W
R21	RES-CF 12K +-5% 1/4W
R22	RES-CF 470R +-5% 1/4W
R23	RES-MF 4.7K +-2% 1/4W
R24	RES-MF 2.7K +-2% 1/4W
R25	RES-MF 22K +-2% 1/4W
R26	RES-CF 68K +-5% 1/4W
R27	RES-CF 12R +-5% 1/4W

A	27JUN83	E	
B		F	
C		G	
D		H	

TITLE

COMPONENT CODE LIST

AC8151

ASTEC

HONG KONG

Sheet 3 of 4

A4

AE 2022 D.P.

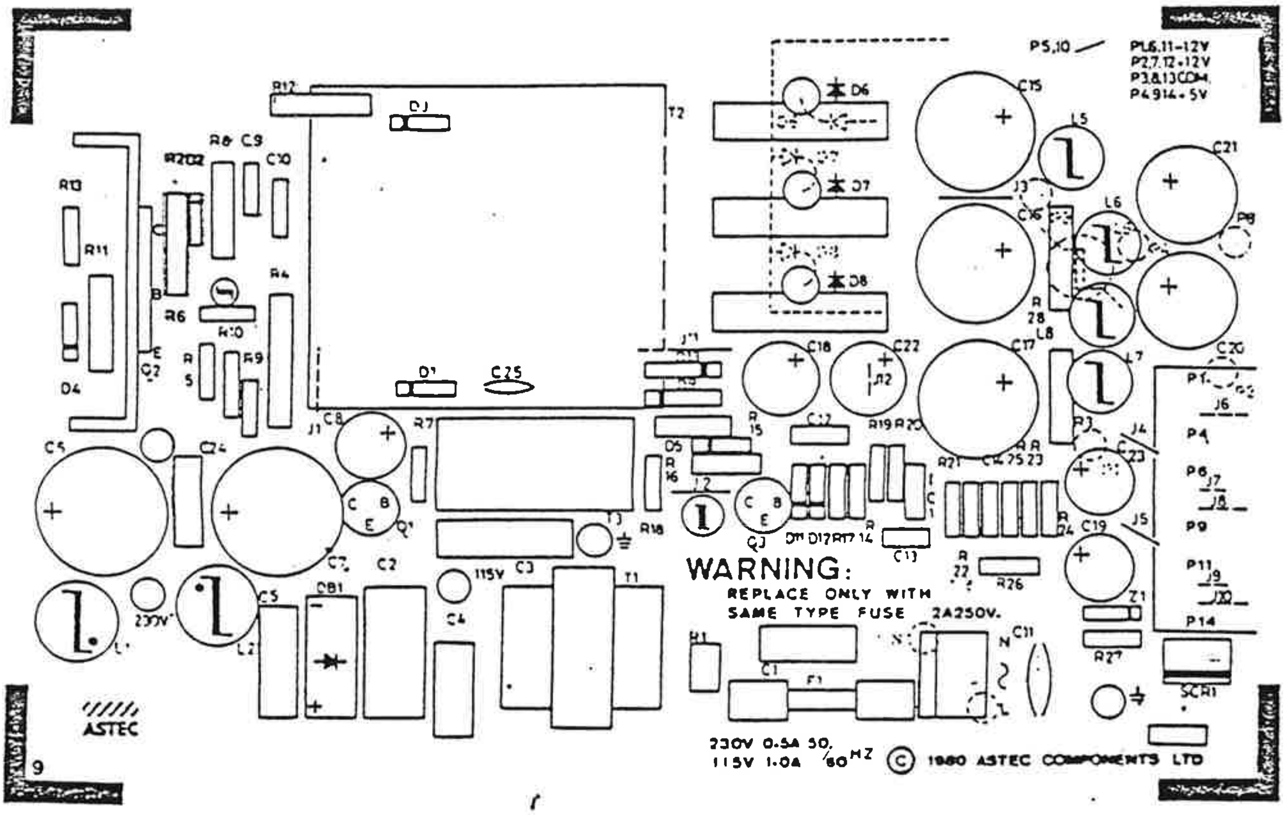
SCR 1	SCR C122U
T1	COM MODE TRF
T2	PWR TRF
T3	CONTROL TRF
Z1	DIODE-Z 5.6V +-5% 40MA

A	27JUN83	E		TITLE COMPONENT CODE LIST	AC8151
B		F			
C		G		ASTEC HONG KONG	Sheet 4 of 4 A4
D		H			

F 2022 D.P.

PROPRIETARY STATEMENT

THIS DOCUMENT CONTAINS DATA AND INFORMATION PROPRIETARY TO ASTEC COMPONENTS LTD. THIS DATA AND INFORMATION SHALL NOT BE DISCLOSED, DISSEMINATION OR REPRODUCED IN WHOLE OR IN PART WITHOUT THE EXPRESS PRIOR WRITTEN AUTHORIZATION OF ASTEC COMPONENTS LTD.



G	4/6/82	H	
A	29/8/80	D	6/1/81
B	9/10/80	E	11/5/81
C	10/11/80	F	9/7/81

TITLE

P. C. B. SPECIFICATION

REV 9

042- 020167-XX

ASTEC COMPONENTS LTD. HONG KONG

Sheet 4 of 7

A-4

BD1022

A User's Guide to FSMGDOS

Introduction:
This document describes the new features of FSMGDOS and FONTGDOS. We should begin with a quick review of what GDOS is and what it allows you to do. GDOS is actually an extention of your operating system. It allows the programs that you run to output text and graphics to printers and other devices besides the screen. GDOS also enables you to print text in many different font faces. Instead of typing characters in the standard system font, you now have access to Swiss, Times, Lucida and a multitude of other fonts. GDOS is run when you first turn on your machine. For now, don't worry about where to put it or what it needs to run. Our install program will set your system up so that everything will be placed in the correct location.

Basic Concepts:
FSMGDOS represents the latest and by far the most powerful release of Atari GDOS. Your first question is probably, "What's the difference between the GDOS that I have now and FSMGDOS"? The main difference between the two is that FSMGDOS allows programs to print in scalable outline fonts instead of bitmap fonts. Outline fonts offer high-quality text at all sizes, whereas bitmap fonts are limited to a small number of sizes that become unappealing when crudely scaled by the system. Furthermore, unlike bitmap fonts, which are found in GDOS, outline fonts allow you to use the same font file to print any size character on almost any output device. This is because the description of the characters and not the character data is stored in the font file. An "a" for example is built using the same set of rules whether it's built at 10 point or at 24 point. This means you no longer need a separate font for the different screen resolutions and, more importantly, you no longer need separate fonts for your printers and other devices. FSMGDOS can create all of these characters for all devices and all sizes using the same font description.

If you want to continue using some of the bitmap fonts that came with GDOS, that's fine. FSMGDOS will still support bitmap fonts. In fact, another new feature in FSMGDOS is "font caching". With old versions of GDOS, every bitmap font that you used took up memory, and your machine would quickly run out of space. With font caching, you tell FSMGDOS how much memory that you wish to devote to storing your bitmap fonts. This memory is called a cache. FSMGDOS will load in the font as it's used. When there is no longer any room for the next font, FSMDOS will get rid of one of the fonts to make room. As long as the memory that you allocate is bigger than the largest font, FSMGDOS will be able to use as many bitmap fonts as you wish in a limited amount of space. For those of you who are using STs without enough memory to support FSMGDOS, (an ST with less than a megabyte of ram) we have included a GDOS called FONTGDOS. FONTGDOS does not work with fsm fonts, but does have all of the font caching capabilities mentioned above.

Getting Started

Installing FSMGDOS or FONTGDOS onto your system is a simple task. Either you already have FSMGDOS pre-installed on your hard drive, or you have to install it from floppies. In the former case, you don't need to do anything. In the latter, put disk #1 into your floppy drive. Run install.prg to put a new GDOS onto your system. The install program will walk you through the steps needed to install your GDOS. When the installation is completed, you will have all of the necessary files on your system to begin.

After the installation is completed, you must reboot your machine to enable the new edition of GDOS. If you've installed it correctly, you'll get the new FSMGDOS or FONTGDOS sign-on message while you boot up. If all went well, GDOS was installed and you should be seeing a normal desktop. If GDOS gives you an error message while booting up, run the install program again paying close attention to any installation error messages which may come up. Note that the installation program will install brand new device drivers, and that old drivers will not work with either FSMGDOS or FONTGDOS.

Try running the demo program which was placed in your FSM directory. Unlike your old GDOS programs, this program was written especially for FSMGDOS and its scalable fonts. If you are running FSMGDOS, notice that you have access to any size of font as well as arbitrary rotation, arbitrary skew, mirroring, and many other features. Use the built in help function to assist you in using the program.

FSMGDOS should be compatible with whatever word processor and draw programs that you have now. Try running some of these programs to see how they work with FSMGDOS. Note that the installation program will set up your system so that you can use all of your new FSM fonts immediately. To change the set-up (e.g. you want to add more point sizes), read the next section on how to use the FSM Font Manager accessory.

In addition to the Font Manager, two other accessories/CPX's have been provided. The first one, the FSM Printer Init accessory, is used to change your printer driver configuration; you can change the page size, number of colors, and etc. The other one, the GDOS manager, is used to manage the older style bitmap fonts and change drivers. You should not need to use these accessories very often, since the install program should set up your system to be ready to use. For more information on these two accessories, please refer to **Appendix II**. Both of these accessories are compatible with FONTGDOS. Since FONTGDOS is identical to your old GDOS from a user's point of view, most of the rest of this document will be devoted to the discussion of FSMGDOS.

The FSM Font Manager

The FSM Font Manager allows you to configure FSMGDOS to your particular system. Using the accessory, you can tell FSMGDOS which fonts you want to use and what point sizes you want available for each font. After bringing up the FSM Font Manager, you will notice a list of installed fonts. These are the fonts currently active, which means that you will be able to use them in your applications. When you click on one of the font names (and it becomes high-lighted), two buttons will appear that will let you delete the font or change the available point sizes. You can also double-click on a font name to change its point sizes. The point sizes that you choose are the ones that you can use with applications, although newer programs allow you to choose any point size without specifying them through the accessory. In order to add different fonts, you can use the "Options Menu" and choose the "Show Deactivated Fonts" option. Now, you can choose from the list of unused fonts, and activate them. Select some deactivated fonts and click on the "Activate Font(s)" button. The fonts will be activated, and your applications will be able to access them. You can see the new fonts on the other list by going to the "Options Menu" and choosing the "Show Active Fonts" option. At any time, you can click on the save box to save your changes.

The FSM menu bar enables you to access other parts of the accessory. Clicking on the "FSM cache Options" menu item brings up a dialog box which allows you to manipulate the FSM cache. Remember that FSM must build its characters before it displays them. You may notice a slight delay the first time that you type a character. Because FSM stores (caches) away the generated character data, the next time you display those characters they will be printed out much quicker.

Using the FSM cache options dialog box, you are able to load, save, or flush your fsm character cache. At first glance, you can see how useful these functions can be. As an example, say you had a document that contained only Lucida Roman in 10, 14, and 18 point characters. Since the characters are being displayed on the screen, you know that they've been cached by FSM. Save the cache, using a name that you can easily associate with your document file. The next time you're ready to load this file, flush the cache, then read in the cache which was saved out with your file. After loading the document notice how fast the screen is able to redraw. This is because FSM does not need to generate any of the character data for the document. An append function allows you to add character data to your cache instead of replacing the already existing data. It should be noted here that your cache is not lost when you exit your program. If you leave your program then enter it again without a reboot, your cache will be preserved. If the FSM accessory detects a file named default.fsm in your FSM directory when your machine boots up, it will automatically be loaded in.

Click on the "Make Width Tables" menu option to save out a set of widthtables. You don't really need to know what widthtables are, but they can greatly improve the speed of an application. Call the software company that produced the application you are using to see if they recommend using widthtables. Otherwise, a test to see if using widthtables is worthwhile is to build them and make sure the widthtable option is set to "Yes"; then try running an application to see if it comes up faster than when the widthtable option is set to "No". If it is faster, then continue using widthtables. Just remember to build them whenever you change something in the accessory and it warns you to build them. Note that you may need to build widthtables again if you change resolutions.

The dialog box which is brought up when you click on the "Outline Font Setup" menu item allows you to set directory paths, cache sizes, widthtable option, and symbol/hebrew files. The items in this dialog box will probably not need to be changed that often. Refer to **Appendix I** to find out what these settings represent. Notice a button on the lower left hand corner of the dialog box named "Defaults". Clicking on this button will bring you into a dialog box which will let you set default point sizes for the fonts that have been, or will be installed. This saves you the trouble of setting the point size of every font if you want them to share the same point sizes. Use the "Set All Fonts" button to set point sizes for fonts which have already been installed. Otherwise, only fonts which are installed later are affected.

Conclusion

You now know enough about FSMGDOS to be able to use it to print scalable fonts using both new and existing programs. At the end of this document is a quick reference guide; the guide should assist you to use

FSMGDOS and its accessories. If you want more details about how FSMGDOS works, please go on to the following appendices

Appendix I
A Detailed Look at FSMGDOS

The EXTEND.SYS file provides a mode of communication between you and FSMGDOS. This works in very much the same manner as the way that GDOS is given information through its ASSIGN.SYS file. Your fsm accessory handles the details of what is contained in this file.

As mentioned before, FSMGDOS will also handle the caching of bitmap fonts. It is a good idea to make the bitmap cache larger than the biggest bitmap font that you have in your ASSIGN.SYS file. If the cache is too small for a font, the font will not be loaded into the system. If the cache is too small to hold any bitmap fonts, fsm will ignore them altogether. If you don't plan to use bitmap fonts, you can set the cache size to be 0.

FSMGDOS maintains two other caches to keep track of FSM font information. One of the caches holds the actual character data. When a character is requested, fsmgdos builds the bitmap using the instructions which it gets from the font file. This takes a fair amount of time. Instead of building the character each time it is requested, FSMGDOS saves the bitmap of the character into the cache. The next time FSMGDOS needs the character, it simply retrieves it from the cache taking a fraction of the time that it would have taken to build it. Obviously, the performance of your machine will improve as you devote more memory to this cache. The second cache used by FSMGDOS is used for internal buffers and data structures. The size needed for this cache varies depending on how many fonts and point sizes are included in your EXTEND.SYS. The cache size needed also depends on the size of the characters that must be generated. With all of the variables involved, it is hard to recommend an optimal amount of memory to devote to this cache. To make matters worse, if GDOS does happen to fill up this cache, system limitations prevent it from safely grabbing another chunk of memory to handle the overflow. The result is a "Not enough FSM Cache memory" message. When you see this message, save your document as soon as possible and increase the amount of cache. You must reboot your machine to incorporate the new cache size. When you're adding fonts or modifying the cache size in the FSM accessory, the accessory will warn you if it determines that your temporary cache is too small. Again, experiment with these values to tune it for your particular system. If you've got 4 megabytes on your system, you are able to allocate a lot more cache. If you're working with 1 megabyte, you may need to decrease these numbers. For normal usage (i.e. no 200 point characters), 100000 bytes or so is probably the most that you will need for the second cache.

To set the sizes for all available caches, go to the "Options Menu" of the FSM Font Manager accessory and click on the "Outline Font Setup". From there, click on "Set Font Cache." The top cache, labelled "Character", is the cache where the individual outline font characters are stored. This should be set to a minimum of 50000 bytes, although if your system has only a megabyte of memory, 20000 bytes would be reasonable. The second cache, "Miscellaneous", should be set to at least 50000 bytes, or 20000 bytes if you are short of memory. Remember that if you do have a limited amount of RAM, you should correspondingly limit the number and size of fonts that you choose. You can use FSM fonts effectively on small systems with very small caches, as long as you aren't using too many fonts and you use normal point sizes (e.g. 10, 14, 24 pts.). Finally, the last cache is the "bitmap" cache where bitmap fonts are stored. To set this cache, find out which bitmap fonts you use and what their file sizes are. To set the very minimal cache size, just set the cache to be as big as the biggest bitmap font file.

Programs need to obtain the widths of characters to determine where on the screen or page to place them. When the width of a character is requested, FSMGDOS will either get the width from the widthtables (if they're turned on), or, get the width from the font generator (if widthtables are turned off). If width tables are turned on, the user may notice long unexpected delays while the application is booting up. This delay is caused by the time it takes FSMGDOS to generate the tables. Obviously, the more fonts and point sizes you have, the longer the delay will be. The FSM accessory may be used to pre-build widthtables and store them into your FSM directory. When the application is being loaded, FSMGDOS sees if the widthtables are turned on, and then checks this directory to see if the width tables are there. If they are, FSM will load them in and use them, saving considerable time booting up the program. The widthtable must have been built in the same resolution that you're currently running the application in order for FSMGDOS to recognize it. Also, widthtables are only used for the screen, not for printers. Finally, it is important to note that certain programs (e.g. Microsoft Write), for reasons too complicated to discuss here, need widthtables turned on in order for them to run correctly.

FSMGDOS generates its characters by looking in QFM files. The QFM files are separated into three different categories. The first kind of file is the main QFM file: this file contains information to create the most commonly used characters of a particular font style. The second and third category are the symbol and hebrew QFMs; these are the files that generate characters common to all font styles (e.g. all fonts share the same trademark symbol, pi, or greek characters). Therefore, in order to generate a full set of characters, FSMGDOS must have access to all three QFM files. If however, you only need part of the character set, you may exclude the symbol or hebrew QFMs using your accessory. Note that all QFM files end with the extender: ".qfm".

To install the symbol or hebrew files, go to the "Options Menu" and click on the "Outline Font Set-Up". For either symbol or hebrew, click on the filename or the box next to the "SYMBOL" or "HEBREW" (if the filename says "NONE"). You will see a file selector and you can choose a QFM file. FSMGDOS comes with one symbol and one hebrew file: LUCSYM.QFM (a symbol file) and LHEBRW.QFM (a hebrew file). You should never install a non-hebrew or non-symbol file to the HEBREW or SYMBOL slot, respectively.

An important difference between bitmap fonts and outline fonts is that a font and its set of point sizes are made available to all devices, automatically. With bitmap fonts, each device requires a different set of font files, whereas with outline fonts, FSMGDOS can scale fonts for any device. At least one point size must exist in order for fsmgdos to recognize the font. Fonts and point-sizes may be added/deleted/changed freely, but FSMGDOS will only recognize changes when the next application is launched.

As a side note, the QFM files are not the only files that must be in the FSM directory; OTL files must be present for FSM to work correctly. Each QFM file should have a corresponding OTL file (e.g. LUCB.QFM and LUCB.OTL). Also, some fonts require more than one OTL file (possibly with an unrelated filename). Thus, it is advised that you keep all of your FSM-related files in the FSM directory. Should an OTL file be missing, FSMGDOS will issue a warning, and any on-going application should be abandoned

Appendix II
Using the Printer Selector and Printer Config Accessories

The main function of the Printer Selector is to allow you to choose which printer driver to use as your main driver. This is particularly useful when you physically want to change printers (e.g. You switch from a dot-matrix printer for proofing to a laser printer for final output.). You can also select final or draft mode from this accessory. Selecting draft mode reduces the quality of output while speeding up printing time (Note that some printers do not support draft mode.). In addition, the Printer Selector is an accessory that manipulates the configuration file called the ASSIGN.SYS. This file contains the information on where GDOS locates its drivers and bitmap fonts and which of those GDOS should use. The installation program should have set up your ASSIGN.SYS file so that you don't really need to change anything. The accessory will be most useful to you when you need to change printer drivers or add new bitmap fonts. Novice users should limit the changes made with this accessory although users familiar with the ASSIGN.SYS format will find it easy to configure their systems.

When you run the accessory, it should come up displaying the current printer driver. If you wish to use a different driver, just click on the driver name, and select from the list that pops down. Notice the two buttons labelled "Quality:" which allow you to choose final or draft mode. To change other drivers, as well as the bitmap fonts and the pathname used to find them, click on the "Options" button to enter a box containing three buttons. Clicking on "Set Font/Driver Path" will come up with the standard file selector and allow you to change where the fonts and drivers are found. If you click on "Driver Installation", the top half should display what the active devices are (i.e. the drivers that applications can use). Use the up and down arrows to look at the different device numbers and their corresponding drivers. Notice that you cannot delete the "screen.sys" drivers found in devices 1 to 10 (They should always use "screen.sys"), but you can change the others. Notice that device 21 corresponds to the "Current Printer" of the previous menu. You can change the driver from either menu, although the current menu allows you to change more than just the current printing device. The bottom half of this menu is used to add more devices to the active list. Novice users should not worry about this section. If you click on the "Font Installation" button, you can add or delete fonts for different devices. As you walk through the different devices, notice how each device has a set of fonts listed. These are the available bitmap fonts if an application uses that device. Note that devices 1 through 10 represent the different screen resolutions, and that device 21 is the printer device, the one that your applications are most likely to use. If you wish to add fonts for a particular application, add them to the devices that are listed in your application's manual. To actually add fonts, click on the drop-down menu called "Active Fonts" and click on "Show Inactive Fonts". Click on the fonts that you want (hold the shift key to select multiple fonts) and click on "Transfer".

The Printer Config accessory is used to change settings in the drivers for items such as the page size or number of colors. Configuring drivers should be done before running an application. If you click on "Modify Drivers", select the driver that you want to change. You should then see a display of the driver specifications. Note that if a box is shadowed, you may change this value by clicking on it. If the particular printer driver you have chosen supports a particular feature (e.g. different page sizes or color pallettes), use the pop-up menu to look at the available selections. All you need to do is click on the appropriate selection. Note that some do not have any configurable features, and that the CPX/accessory will modify the actual driver on your floppy or hard disk when a modification is made.

QUICK REFERENCE GUIDE

To Add FSM Font(s):

Run FSM Font Manager.
Use Options Menu to select "Show Inactive Font(s)".
Click on font names (shift-click to grab more than one).
Double click on font or click on "Activate Font" button.
Click on "SAVE".

To Delete FSM Font(s):

Run FSM Font Manager.
Click on desired font names (shift-click to grab more than one).
Click on "Deactivate" button.
Click on "SAVE".

To Set Cache Sizes:

Run FSM Font Manager.
Use Options Menu to select "Outline Font SetUp" and select "Set FSM Cache".
If you notice frequent regeneration of the screen characters or slow printer output, increase "Character" cache.
If "Out of memory" error messages appear, increase "Miscellaneous" cache.
If bitmap fonts that you've made active do not appear, increase the "Bitmap" cache.
Save and REBOOT.

To Set Point Sizes:

Run FSM Font Manager.
Double-click on a font or click on font and use "Activate" button.
Use buttons with numbers to select point sizes or click on "Add Point Size".
Click on point sizes and use the "Delete Point Size" button to remove sizes.

To Set Default Point Sizes:

Run FSM Font Manager.
Use Options Menu to select "Outline Font Setup".
Click on "Defaults" button.
Add/delete fonts.
To set all active fonts to the chosen point sizes, click on "Set All Fonts".

To Install Symbol/Hebrew Characters:

Run FSM Font Manager.
Use Options Menu to select "Outline Font Setup".
Click on either SYMBOL or HEBREW.
Use file selector to choose an appropriate QFM file (Ask your font supplier).

To Change Printer Drivers:

Run Printer Selector.
Click on pop-up below"Current Printer".
Select new driver and save.

To Change Bitmap Fonts:

Run Printer Selector.
Click on "Options" then on "Font Installation".
Select device number with up/down arrows (1-10 are the different screen resolutions, 21 is the printer).
To add fonts, click on "Active Fonts" menu, and choose the "Show Inactive Fonts" option. Click on desired fonts and use "Transfer" button to activate.

XControl - Extensible Control Panel for ST/MEGA/STe/TT Computers

SPECIFICATIONS SUBJECT TO CHANGE WITHOUT NOTICE
XControl Version: 1.0

OVERVIEW

This document describes a new Control Panel (XControl) for ST/MEGA/STe/TT computers, which features loadable Control Panel extensions (CPXs) that performs various sytem configuration functions. XControl, with its loadable modules, gives the Control Panel the advantages of any software with modular design: ease of maintenance and expandability. Improvements to any part of XControl can be distributed individually, by distributing CPX updates, without the need for updating all parts of XControl. This scheme is more flexible for users, since XControl will only load the CPXs which a user needs. Software vendors can create and distribute their own CPXs to extend the functionality of XControl beyond what Atari provides, or to provide graphical front ends for their TSR utilities.

HOW XCONTROL WORKS

XControl is the combination of a master desk accessory, which loads the various CPXs and manages user selection of CPXs, and the extensions themselves, which perform the various system configuration tasks.

When XControl is loaded, at boot time, it looks for a a file called CONTROL.INF in the root of the boot device. CONTROL.INF contains the default settings for XControl.

One of the settings is the CPX DIRECTORY PATH. This is the directory where XControl looks for CPXs. If XControl finds the folder, it reads the header of each .CPX file that it finds there. The header's id number and version number are compared to those already loaded. The end result is that only the latest version of a CPX will be retained.

If the header indicates that a .CPX needs to be run at boot time, XControl loads the CPX and calls its initialization hook. If the CPX header also indicates that the user prefers it to be memory resident, XControl keeps the CPX in memory.

After checking CPXs and initializing those that need it, XControl adds an additional set of CPX slots. The number of slots available is based upon either the default minimum number of slots set by the user, or 1-1/2 times the number of CPXs loaded, whichever is greater. Additional CPXs may be loaded later during a RELOAD command. All CPXs are initialized as if they were new, except that existing 'resident' CPXs are retained. New CPXs marked 'resident' are treated as non-resident CPXS after a reload. During a reload, a new CPX will only be loaded if there is a vacant slot available.

After adding additional vacant slots, XControl waits like any other desk accessory for an AC_OPEN message.

When selected from the Desk menu, XControl opens a window and displays a menu of active CPXs. When a CPX is chosen from the menu, XControl attempts to invoke it via the XControl <-> CPX software interface described below. Resident CPXs are invoked immediately; non-resident CPXs are loaded from the CPX storage directory. If the CPX is not found by name, XControl looks at all other CPXs in that directory, comparing id numbers and version numbers. If an exact match is found, that CPX is invoked instead. Otherwise, a file-not-found alert is displayed.

When invoked, the CPX assumes control of the work area of the XControl window, and can present its own interface there. XControl dispatches user events through the CPX, but handles most of the window related events itself. XControl also provides a number of utility routines to CPXs, including an extended form_do call which CPXs can use to simply handle dialog-style interfaces within the XControl window.

It's expected that most CPXs will use this extended form_do() software interface so that the user can move or close the XControl window at will. Each CPX should provide at least an OK and Cancel button so that the user can return to the XControl master from the CPX. Each CPX must also be able to respond to an Abort signal from XControl, so that the user can dismiss XControl with the close box, and so that XControl can clean up if it is active while the main application is terminated. When the user exits a non-resident CPX, it is unloaded, and the memory that it took up is recovered by the system.

CONTROL PANEL EXTENSIONS

The concept of what constitutes a CPX is very important to the implementation of the extensible Control Panel. A CPX is effectively a subroutine call. It is neither an application nor a desk accessory, but only a means for setting system parameters. Examples of CPX functions include:

- Color Selection
- Keyboard/Mouse configuration (repeat rate, audible keyclick, etc.)
- RS232 port configuration
- Printer configuration

Note the key word "configuration" in most of the above functions. A printer driver does not belong in a CPX, but the ability to configure a TSR printer driver would be a good thing to have in a CPX. The key concept to keep in mind here is that of the "Control Panel" - the one place where a user goes to toggle switches, press buttons, or whatever, to "control" the functions of the computer. Obviously, it is silly to have a CPX which controls the operation of a desk accessory, Instead, CPXs should primarily be used as graphical front ends for TSR utilities.

XCONTROL <-> CPX SOFTWARE INTERFACE

When XControl first starts up, it loads the headers of all the CPXs that it can find. At boot time, it initializes each CPX which has the bootinit flag set in its header by jsr'ing to the start of the CPX's text segment. This in turn will jmp to the cpx_init() function. This function returns a pointer to a structure containing information about the CPX, including pointers to routines which XControl uses to invoke CPX functions.

"Set-only" CPXs may be implemented. They should set whatever is needed during the cpx_init() call and return NULL. If a CPX is set for bootinit, XControl also checks the 'Set-Only' flag in the header to determine if the CPX is Set-only. XControl will only execute Set-only CPXs at boot time and at reloads. They will not appear on the XControl main menu, and thus will never again be called after the cpx_init() call.

XControl uses an event_multi() for its user interaction. When a CPX is chosen by the user, XControl loads the CPX into memory and calls cpx_init() again, this time, with the 'booting' parameter set to FALSE. XControl then invokes the cpx_call() routine to begin CPX interaction. The cpx_call() routine should first initialize the CPX interface. It may then handle the user interface via the extended form_do call and return FALSE to exit the CPX (See Form CPXs), or to return TRUE and allow XControl to manage the user interace by dispatching evnt_multi() events through the CPX event handling routines. (See Event CPXs).

FORM CPXs versus EVENT CPXs

FORM CPXs are those which use Xform_do() to handle user interaction with a standard AES form. XControl handles window movements and redraws. To the CPX, it looks just like the old familiar form_do() with a few extensions:

- Keys other than those which work in editable text fields can be handled by the CPX.
- Special redraws may be done by the CPX.
- If the user closes the XCONTROL window,k or quits the parent application, the CPX is informed so that it may clean up. AC_CLOSE should be treated as "Cancel", and WM_CLOSE as "OK".

To give you an idea of the flexibitlity FORM CPXs may have, all of the CPX's released with XControl 1.0 are FORM CPXs.

EVENT CPXs are those which use XControl to dispatch AES events directly, for maximum flexibility. These CPXs give XControl a list of event handlers in the CPXINFO struct returned by cpx_init(). When an Event CPX is called, it tells XControl which events it cares about via the set_event_mask() function, then it returns to XControl and waits for tis event handlers to be called. Event CPXs exit by setting a flag passed to the event handlers.

Because of the flexibility offered by Xform_do(), and the more complex nature of the event handlers, it's generally much easier to write a Form CPX than an Event one. The main reason you might want an Event CPX is to handle timer events which are not supported by XForm_do().

XCONTROL ROUTINES

At boot-time or invocation time, XControl jsr's to the text segment of the CPX. XControl passes on the stack a pointer to an XControl Parameter Block, with information of interest to the CPX. The XCPB struct looks like:

```
typedef struct {
short      handle;              From XControl's Graf_Handle() Call.
                                See the Notes on workstations below.

short      booting;             Non-zero if this cpx_init() call is
                                part of XControl's initialization.

short      reserved;
short      SkipRsh Fix;         The cpx must call the resource fixup routine
                                only once.  Non-zero means skip the fixup.
void       *reserve1;
void       *reserve2;

void       (*rsh_fix)(  int num_objs, int num_frstr, int num_frimg,
                        int num_tree, OBJECT *rs_object,
                        TEDINFO *rs_tedinfo, BYTE *rs_strings[],
                        ICONBLK *rs_iconblk, BITBLK *rs_bitblk,
                        long *rs_frstr, long *rs_frimg, long *rs_trindex,
                        struct foobar *rs_imdope );

void       (*rsh_obfix)( OBJECT *tree, int curob );

short      (*Popup)( char *items[], int num_items, int default_item,
                    int font_size, GRECT *button, GRECT *world );

void       (*Sl_size)( OBJECT *tree, int base, int slider, int num_items,
                       int visible, int direction, int min_size );

void       ( *Sl_x) (  OBJECT *tree, int base, int slider, int value,
                       int num_min, int num_max, void (*foo)() );

void       (*Sl_y)( OBJECT *tree, int base, int slider,  int value,
                    int num_min, int num_max, void (*foo)() );

void       (*Sl_arrow)( OBJECT *tree, int base, int slider, int obj,
                        int inc, int min, int max, int *numvar,
                        int direction, void (*foo)() );

void       (*Sl_dragx)( OBJECT *tree, int base, int slider,
                        int min, int max, int *numvar, void (*foo)() );

void       (*Sl_dragy)( OBJECT *tree, int base, int slider,
                        int min, int max, int *numvar, void (*foo)() );

WORD       (*Xform_do)( Object *tree, WORD start_field, WORD puntmsg[] );
GRECT     *(*GetFirstRect)( GRECT *prect );

GRECT     *(*GetNextRext)( void );

void      (*Set_Evnt_Mask)( int mask, MOBLK *m1, MOBLK *m2, long time );

BOOLEAN  (*XGen_Alert)( int id );

BOOLEAN  (*CPX_Save)( void *ptr, long num );

void      *(*Get_Buffer)( void );

int       (*getcookie)( long cookie, long *p_value );

int       Country_Code;    Contains the Country Code that the Control Panel was
                           compiled for. For a list of the Current Country Codes,
                           please see the Rainbow TOS Release Notes -
                           BIOS/XBIOS Supplemental Documentation, page 63.

void      MFsave(  BOOLEAN saveit, MFORM *mf );
} XCPB;
```

RESOURCE MANAGEMENT:

Resource Object Tree Fixup Function:

Rsh_fix() fixes up the CPX object tree based upon 8x16 pixel characters. This ensures that the CPX will be the same size in all resolutions. The CPX object tree should be a 'Panel', not a 'Dialog'. It should be created in ST HIGH resolution. In comparison, the 'rsrc_load()' function would fixup the tree based upon the current character width and height for that resolution. This is why panels can appear stretched or scrunched in different resolutions when using 'rsrc_load()'.

The CPX should only call rsh_fix() when the XControl Parameter Block 'SkipRshFix' flag is ZERO. The reason for this is because a resource should only be converted to pixels ONCE.

```
void        (*rsh_fix)( int num_objs, int num_frstr, int num_frimg,
                        int num_tree, OBJECT *rs_object,
                        TEDINFO *rs_tedinfo, BYTE *rs_strings[],
                        ICONBLK *rs_iconblk, BITBLK *rs_bitblk,
                        long *rs_frstr, long *rs_frimg, long *rs_trindex,
                        struct foobar *rs_imdope );
```

IN: All of the input variables can be found in the CPX RSH file.
OUT: None

Resource Object Fixup Function:

Call this function ONLY when you want to convert a specific object to pixel format AND when the object is still CHARACTER based. The only reason you would need to call this function would be if you were doing your own resource fixup for a resource that was not created by the Atari RCS.

```
void        (*rsh_obfix)( OBJECT *tree, int curob );
```

IN:	OBJECT *tree	The object tree of the CPX
	int curob	The resource object to convert

OUT: None

POP UP MANAGEMENT:

Call this function to have the CPX display a popup box:

```
short       (*Popup)( char *items[], int num_items, int default_item,
                      int font_size, GRECT *button, GRECT *world );
```

IN:	char	*items[];	Pointer to an array of strings.
	int	num_items;	Number of items (1 based)
	int	default_item;	The default item (zero based)
	int	font_size;	8x16 (Large) or 8x8 (small) Font
	GRECT	*button;	GRECT of button pressed to invoke popup.
	GRECT	*world;	GRECT of your tree.
OUT:	short;	Returns the item selected (zero based) or -1	

The string array passed to the popup routine must be properly padded by the calling cpx. There must be at least 2 spaces in front of each string, and each string must be padded with spaces up to the length of the largest string plus 1.

The number of items listed versus the number of strings available is not checked. If they do not correspond, errors may occur. In addition, if there are more than 5 items, only 3 will be displayed at any one time. The first position will display an up arrow, and the 5th position will display a down arrow. Scrolling thru the popup will display the remaining menu items, with a check mark indicating the default item.

Sometimes a default item is not necessary. Setting the default_item = -1 will prevent a check mark from appearing.

There are 2 font sizes available for AES objects the large and the small font. Currently, the large font is always used and the height is assumed to be 16 pixels.

The GRECT of the button that activated the popup menu is required so that the menu is at least as wide as the button.

The GRECT of the world is required so that if the popup menu exceeds the right edge, it pops left instead. If the popup menu exceeds the bottom, it pops upwards. In most cases, the "world" is the dimensions of your CPX's main form .

While the popup menu is displayed no other action other than popup menu manipulation is allowed.

SLIDER MANIPULATION:

(Need Overall Description)

Slider Size Adjustment:

This function is used to adjust the slider size within its base, so that the size of the slider represents the amount of data displayed, relative to the total amount of data. In certain cases, it is best that the slider not be sized. An example of this is when the slider also contains a text string. It is possible, that if sized, the slider can no longer display the text string properly by either being too small or too large.

```
void    (*Sl_size)( OBJECT *tree, int base, int slider, int num_items,
                    int visible, int direction, int min_size );
```

IN:

OBJECT *tree;	The object tree: (OBJECT *)rs_trindex[TREENAME].
int base;	The base is the object of the sliders limits.
int slider;	The object that moves within the limits defined by base. The slider must be the child of the base.
int num_items;	The number of items (range)
int visible;	The number of items visible
int direction;	Horizontal or Vertical
int min_size:	The minimum pixel size of the slider

OUT: none

Slider X/Y Functions:

Sl_x() and Sl_y() are used to update the position of the slider within its base.

```
void    (*Sl_x)( OBJECT *tree, int base, int slider, int value,
                 int num_min, int num_max, void (*foo)() );

void    (*Sl_y)( OBJECT *tree, int base, int slider, int value,
                 int num_min, int num_max, void (*foo)() );
```

IN:

OBJECT *tree;	(OBJECT *)rs_trindex[TREENAME];
int base;	Base of the slider (slider limits)
int slider;	The object that will be moved around. This must be the child of the base.
int value;	The NEW value related to the slider range. (Range: 0 - 1000)
int num_min;	The minimum number value can equal to.
int num_man;	The maximum number value can equal to.
(*foo)();	Pointer to a CPX defined function to update its items. Set to NULLFUNC if there is no routine.

OUT: none

Slider Arrow Functions:

Call this when the user selects the arrows of a slider. Direction is either Horizontal or Vertical. Note that this is an ACTIVE slider where objects are updated immediately, unlike the AES graf_slidebox where objects are updated only after the user lets go of button one.

```
void        (*Sl_arrow)( OBJECT *tree, int base, int slider, int obj,
                         int inc, int min, int max, int *numvar,
                         int direction, void (*foo)() );
```

IN:	OBJECT *tree:	The resource tree
	int base:	The base of the slider (slider limits)
	int slider:	The object that can be moved around
	int obj:	The arrow button object clicked on
	int inc:	The increment amount (+/- #)
	int min:	The minimum value possible
	int max:	The maximum value possible
	int *numvar:	The current value
	int direction:	Horizontal or Vertical
	void (*foo)():	Pointer to a CPX defined function to update its items. Set to NULLFUNC if there is no routine.

OUT: none

Slider Paging Functions:

Paging is implemented by calling sl_arrow() with an increment/decrement value representing a "page" worth of data. Paging is done when the user clicks on the base. To implement paging the CPX can do this:

```
MRETS mk;
int  inc, ox, oy;

Graf_mkstate( &mk );
objc_offset( tree, slider, &ox, &oy );
inc  = ( ( mk.y < oy ) ? ( -1 ) : ( 1 ) );
sl_arrow( fill in variables here );
```

This example is for vertical sliders and the increments were set to +/- 1. Paging usually increments or decrements by the visible amount. To do horizontal pages, use the 'ox' and 'mk.x' variables instead and don't forget to set the horizontal or vertical flag as necessary in sl_arrow().

Slider Drag Functions:

Called when the user 'drags' the slider around. Again, this is an ACTIVE slider and will call Sl_x() or Sl_y() appropriately.

void (*Sl_dragx)(OBJECT *tree, int base, int slider, int min,
int max, int *numvar, void (*foo)());

void (*Sl_dragy)(OBJECT *tree, int base, int slider, int min,
int max, int *numvar, void (*foo)());

IN:	OBJECT *tree:	The resource tree
	int base:	The base of the slider (slider limits)
	int slider:	The object that can be moved around
	int min:	The minimum value possible
	int max:	The maximum value possible
	int *numvar:	The current value
	void (*foo)():	Pointer to a CPX defined function to update its items. Set to NULLFUNC if there is no routine.

OUT: none

User Supplied Slider Update Function:

The User Supplied CPX function may be required so that the CPX can perform operations specific to the active slider. In most cases, this will simply be updating the text string and then performing a redraw. In a more complicated setting, this can be anything from changing colors to performing a blit. If no function is required, pass NULLFUNC instead.

The value that you can use to update the text string is contained in the variable that you passed by reference into the calling slider function. Whenever you call sl_arrow(), or sl_draw(), XControl updates that variable just before calling sl_x() and sl_y(). These in turn will call your foo() function.

The prototype for foo() is: void (*foo)(void);

XFORM_DO FUNCTION:

XControl makes a custom form handler available to CPXs so that they may use the standard AES forms interface in a window, without worrying about handling window messages. The object tree should fit within the standard control panel window (256x176 pixels work area). This restriction may be lifted in a future version. The name of the routine is Xform_do and it functions like the built-in AES form_do routine with a few exceptions. One additional parameter is used, and a return value of -1 has a special meaning.

Timer Events are not supported under XForm_do(). If timer events are necessary, the CPX should be designed as an Event CPX.

If the CPX is looking for double clicks, the return value should be checked for -1 BEFORE checking for a double click.

```
WORD    (*Xform_do)( Object *tree, WORD start_field, WORD puntmsg[] );
```

IN:

OBJECT *tree;	Same as form_do;
WORD start_field;	Same as form_do;
WORD puntmsg[];	Defined as WORD msg[8];

OUT: Same as form_do(): Returns the object number with the high bit set if a touch-exit was double clicked on.

-However-

if return is -1, this means that the CPX should look at the puntmsg[] array and treat it like the message array from an event multi. The three messages to look for are:

WM_REDRAW: Sometimes the CPX needs to redraw items that are not part of the tree. This is the time to do so. XControl makes available GetFirstRect() and GetNextRect() so that the CPX can get the rectangle list and redraw accordingly.

AC_CLOSE:
WM_CLOSE: When these messages are received, the CPX should immediately FREE any memory that it malloc'ed and return to XCONTROL by exiting cpx_call(). Do NOT leave any memory allocated, else fragmentation will occur. We strongly recommend that CPXs **Do Not Malloc** any memory.

CT_KEY: A key was pressed. puntmsg[3] contains the keycode of the key pressed as returned from an 'evnt_keybd()'. Note that we return non-printable keys only, such as F1-F10, Help and Undo. However, the 'Arrow' keys are not

supported, because they are handled by Xform_do() for editable text fields.
Note: CT_KEY == 53.

IMPORTANT: Always treat AC_CLOSE() as "Cancel" and treat WM_CLOSE() as an "OK".

GET FIRST/NEXT RECTANGLE LIST FUNCTIONS:

When redrawing the CPX due to a WM_REDRAW message, the CPX should use these routines to go down the rectangle list. Since the Xform_do() routine will handle resource object redraws, the CPX must handle non_resource objects.

```
GRECT        *(*GetFirstRect)( GRECT *prect );

GRECT        *(*GetNextRext)( void );
```

IN: GRECT *prect: The GRECT of the dirtied area.

OUT: The intersecting GRECT that you should redraw or NULL if there are no more rectangles.

SET EVENT MASK
(Use only with Event CPXs)

Used to set XControl's Evnt_multi() function. Messages will be dispatched to the CPX thru procedure variables passed in.

```
void    (*Set_Evnt_Mask)( int mask, MOBLK *m1, MOBLK *m2, long time );
```

IN:	int mask:	Events to receive (ie: MU_MESAG | MU_KEYBD)
	MOBLK *m1:	Mouse rect and direction number one.
	MOBLK *m2:	Mouse rect and direction number two.
	long time:	Time to wait for a timer event (1000 = 1 sec) Note that you must set the mask with MU_TIMER.

OUT: none

MOBLK is defined as:

```
typedef struct {
    int m_out;  Direction for evnt_multi() to look for.
    int m_x;    The x,y,w,h of the bounding rectangle.
    int m_y;
    int m_w;
    int m_h;
} MOBLK;
```

XCONTROL ALERT BOX:

Use this function to display an XControl Alert Box. The dialog box will be centered within the work area of the XControl window. The Alerts available are:

SAVE_DEFAULTS	0	Save Defaults?
MEM_ERR	1	Memory Allocation Error
FILE_ERR	2	File I/O Error
FILE_NOT_FOUND	3	File Not Found Error

```
BOOLEAN (*XGen_Alert)( int id );
```

IN:	int id:	The alert id number
OUT:	BOOLEAN:	TRUE - OK FALSE - Cancel Alerts with only one button always return TRUE.

CPX SAVE DEFAULTS:

XControl allows a CPX to write configuration data directly into its file. XControl will write the number of bytes specified from *ptr to the data segment of the CPX. If the CPX isn't found by name, XControl will search the CPX directory for another file with the same id number and version number. If found, that CPX will become the active cpx. If still not found, a file not found alert will be generated. The standard GEMDOS error will also occur if the disk is write-protected. The start of the DATA segment begins at the variable SAVE_VARS which is declared in the CPXSTART.S file. The CPX designer must allocate the appropriate amount of DATA segment storage by editing CPXSTART.S.

During boot_time initialization, the CPX should read the defaults from the data segment and act accordingly.

Lastly, the CPX should treat a "SAVE" action as an 'OK', but do not exit the CPX.

```
BOOLEAN (*CPX_Save)( void *ptr, long num );
```

IN:	void *ptr long num	Pointer to the data that needs to be stored. Number of bytes to write to data segment of CPX.
OUT:	BOOLEAN:	TRUE - OK FALSE - Error occurred

XCONTROL GET_BUFFER FUNCTION:

This call returns a pointer to the 64 byte buffer in each header which can be used by the CPX. The buffer should be used by CPXs that rely upon write-only registers. For example, the baud rate and flow control data cannot be read from the Rsconf() call. (In TOS 1.4 and greater, the baud rate CAN be inquired.) Since a CPX cannot be guaranteed to be in memory, a non-volatile storage location must be set aside to accomplish this. The CPX can set the register, store the value in the buffer and when cpx_init() is called again, the CPX can restore the data into its internal variables.

void *(*Get_Buffer)(void);

IN: none

OUT: (void *) Returns a pointer to the CPX. The CPX should cast the pointer it's required format.

CPX GET COOKIE FUNCTION:

Use this routine to look for a cookie. Please see the Cookie Jar specifications for more details. The parameters are exactly the same.

int (*get_cookie)(long cookie, long *p_value);

IN:
long cookie: Cookie that we are looking for.
long *p_value: Value of cookie goes here if the cookie is valid.

OUT:
Zero if the cookie is not found
Non-Zero if the 'cookie' is found and places its value in the longword pointed to by p_value. If p_value is NULL, it doesn't put the value anywhere, but still returns the error code.

A cookie can be a convenient marker for a TSR to indicate where a CPX can find the configuration data used by the TSR. That's one of the reasons the cookie jar exists! Use it!

CPX SAVE/RESTORE MOUSE FORM

Use this routine to save/restore a mouse image to/from an MFORM structure. This is useful when one needs to use a FLAT_HAND for example, and then must restore the mouse to its original shape. This is required so that a CPX doesn't wipe out a custom mouse form being used by an application when the CPX is invoked.

void MFsave(BOOLEAN saveit, MFORM *mf);

IN:	BOOLEAN saveit	MFSAVE - Save Mouse Form MFRESTORE - Restore Mouse Form
	MFORM *mf	Mouse Form to store image in
OUT:	none	

CPX INFORMATION ROUTINES

INITIALIZATION:

This routine is called at boot time and also whenever the CPX is executed and should be used by the CPX to initialize global variables, etc.. XControl passses on the stack a pointer to the XControl Parameter Block, which was defined earlier. Cpx_init() should return a POINTER to the following structure, or NULL if it is a "set_only" CPX:

```
CPXINFO *cpx_init( XCPB *xcpb );
```

```
typedef struct {
                BOOLEAN     (*cpx_call)( GRECT *work );
                void        (*cpx_draw)( GRECT *clip );
                void        (*cpx_wmove)( GRECT *work );
                void        (*cpx_timer)( int *event );
                void        (*cpx_key)( int kstate, int key, int *event );
                void        (*cpx_button)( MRETS *mrets, int nclicks, int *event );
                void        (*cpx_m1)( MRETS *mrets, int *event );
                void        (*cpx_m2)( MRETS *mrets, int *event );
                BOOLEAN     (*cpx_hook)( int event, int *msg, MRETS *mrets,
                            int *key,   int *nclicks);

                void        (*cpx_close)(  BOOLEAN flag );
}CPXINFO;
```

Most of these calls are not used when the CPX is an Xform_do type. Those routines not used should be set to NULL in cpx_init();

INVOCATION:

Called when a CPX is invoked AFTER the cpx_init() call has been completed. The function is passed a rectangle describing the current work area of the XControl window. This allows a CPX to set up for user interaction and optionally call the custom xform_do() routine to handle its user interface.

```
BOOLEAN   (*cpx_call)( GRECT *work );
```

IN:	GRECT *work:	GRECT of XControl work window.
OUT:	FALSE	Return FALSE if the CPX is done.
	TRUE	Return TRUE to tell XControl to continue to dispatch events via the XControl CPXINFO routines.

EVENT HANDLING FUNCTIONS:

While an Event CPX is active, these are called in response to the appropriate events. The events returned by XControl are defined by the Set_Evnt_Mask() call in the XCPB. The event mask may be changed at any time while a CPX is active, and the new mask will be used for the next evnt multi. Note that the routines are listed in the same order they will be called for multiple event returns from evnt_multi(). These routines should set the word pointed to by 'event' to TRUE(1) to return control to XCONTROL and its main menu, or leave that word alone to continue with CPX interaction. The *event variable is the event mask and should be ignored otherwise.

Message events are handled by XControl, unless intercepted by cpx_evhook() as described below.

WINDOW MANAGEMENT:

CPXINFO Redraw Event:

Called when a CPX is active and the XControl window needs to be redrawn. This call is required by a CPX that uses XControl to dispatch events (an Event CPX). The CPX should pass the dirty area to GetFirstRect() and GetNextRect() in order to redraw using the rectangle list.

void	(*cpx_draw)(GRECT *clip);	
IN:	GRECT *clip:	GRECT of the dirtied area.
OUT:	none	

CPXINFO Window Move Event:

Called when the user moves the XControl window, so that the CPX may fix up its object tree as necessary. GRECT contains the work window's new coordinates. This call is required by a CPX that uses XControl to dispatch events (ie: an Event CPX).

void	(*cpx_wmove)(GRECT *work);	
IN:	GRECT *work:	GRECT of the new window coordinates
OUT:	none	

TIMER EVENTS:

Called when a timer event occurs. This call is required by a CPX that uses XControl to dispatch events(ie: an Event CPX). The '*event' variable is used to tell XControl that this event has terminated the CPX. Set to '1' to terminate the CPX, else IGNORE it. Note that timer events for Form CPXs are not supported.

```
void        (*cpx_timer)( int *event );
```

IN:	int *event:	Set to '1' if this event terminates the CPX. else ignore this variable.
OUT:	none	

KEYBOARD EVENTS:

Called when a keyboard event occurs. This call is required by a CPX that uses XControl to dispatch events. The '*event' variable should be set to '1' if this event has terminated the CPX, otherwise, ignore it.

```
void        (*cpx_key)( int kstate, int key, int *event );
```

IN:	int kstate:	The state of the Control, Alt and Shift Keys.
	int key:	The high byte contains the scan code of the key pressed, and the low byte contains the ASCII code, if any.
	int *event:	Set to '1' if this event terminates the CPX. Ignore this variable otherwise.
OUT:	none	

MOUSE BUTTON EVENTS:

Called when a mouse button event occurs. This call is required by a CPX that uses XControl to dispatch events. Set the '*event' variable to '1' if this event terminates the CPX, otherwise, ignore it.

```
void        (*cpx_button)( MRETS *mrets, int nclicks, int *event );
```

IN:	MRETS *mrets:	The mouse parameters returned by the event.
	int nclicks:	The number of button clicks for this event
	int *event:	Set to '1' if this event terminates the CPX. Otherwise, ignore it.
OUT:	none	

MRETS is defined as:

```
typedef struct {
      WORD        x;
      WORD        y;
      WORD        buttons;
      WORD        kstate;
}MRETS;
```

MOUSE RECTANGLE EVENTS:

Called when a mouse event occurs. This call is required by a CPX that uses XControl to dispatch events (ie: an Event CPX). Set the '*event' variable to '1' if this event terminates the CPX, otherwise ignore it.

```
void          (*cpx_m1)( MRETS *mrets, int *event );

void          (*cpx_m2)( MRETS *mrets, int *event );
```

IN: MRETS *mrets: Mouse parameters returned by this event.
int *event: Set to '1' if this event terminates the CPX. Otherwise, ignore it.

OUT: none

MRETS is defined as:

```
typedef struct {
      WORD        x;
      WORD        y;
      WORD        buttons;
      WORD        kstate;
}MRETS;
```

CPX EVENT PREEMPTION HOOK:

Cpx_hook() is called immediately after evnt_multi returns BEFORE the event is handled by XControl. This routine should not normally be required by a CPX, but is included for flexibility.

BOOLEAN (*cpx_hook)(int event, int *msg, MRETS *mrets,
int *key, int *nclicks);

IN:
int event: The event mask.
int *msg: The AES event message buffer.
MRETS *mrets: mouse parameters for this event.
int *key: Key returned.
int *nclicks: Number of button clicks for this event.

OUT:
TRUE Return (non-zero) to override default event handling.
FALSE Return (zero) to continue with event handling.

CPX TERMINATION FUNCTION:

This routine is called whenever an ACC_CLOSE or WM_CLOSE message is generated. The CPX should immediately free up any allocated memory and return to XControl. This routine is required for all CPXs that use XControl to generate events. Failure to free allocated memory will result in a fragmented system. Note that this is for an Event CPX only and is not necessary for Form CPXs. **IMPORTANT**: Always treat ACC_CLOSE messages as 'Cancel' and WM_CLOSE messages as 'OK'. In addition, CPXs should not malloc memory if at all possible.

void (*cpx_close)(BOOLEAN flag);

IN:
TRUE - WM_CLOSE Message
FALSE - ACC_CLOSE Message

OUT: none

CPX FILE FORMAT:

A CPX file header looks like: (512 bytes - 0x200 hex)

```
typedef struct _cpxhead {
unsigned short magic;            Magic Number == 100
struct {
        unsigned reserved  :     13;     Reserved for Expansion
        unsigned resident  :     1;      RAM Resident Flag
        unsigned bootinit  :     1;      Boot Initialization Flag
        unsigned setonly   :     1;      Set Only CPX Flag
} flags;

long            cpx_id;          CPX ID value
unsigned short  cpx_version;     CPX Version number
char            i_text[14];      Icon Text
unsigned short  sm_icon[48];     Icon bitmap - 32x24 pixels
unsigned short  i_color;         Icon Color

char            title_txt[18];   Title for CPX entry.
                                 Note: Only use 16 Characters! The remaining 2
                                       positions are Nulls.
unsigned short  t_color;         Tedinfo field for color
char            buffer[64];      Buffer for RAM storage

char            reserved[306];   Reserved for Expansion

} CPXHEAD;
```

The first file in the link must be CPXSTART.S which jmp's to cpx_init(). In addition, it also contains the default variable storage in the DATA segment.

The user will be able to set the Resident Flag, Title Text, Title Color, Icon Text and Icon Color with a CPX.

The rest of the CPX file has the same format as a GEMDOS executable file. PREFIX.PRG should be used to design and prepend the header to the CPX executable. The executable part does not need to be completely relocatable, as XControl will perform whatever relocation is necessary when it loads the CPX. The resource for the CPX must be built into the file and should be fixed up in place using the rsh_fix() facility of XControl.

FILE:	Header	512 bytes
	GEMDOS Header	28 bytes
	Text Segment	...
	Data Segment	...

TERMINOLOGY:

*.CPX:	A standard CPX file with header ready for use.
*.CP:	A standard CPX file without a header.
*_R.CPX:	A resident CPX
*_S.CPX:	A Set-only CPX
SLOT:	A slot is where a CPX is stored in memory. There are both active and non-active slots. The number of slots available is decided at boot-time. XControl will create the minimum number of slots specified by the DEFAULT or 1-1/2 times the number of active slots, whichever is greater. This is the ONLY time slots are allocated.
*.HDR:	A CPX header created by PREFIX.PRG
Event CPX	A CPX that handles the event messages explicitly.
Form CPX	A CPX that uses XForm_do to handle event messages.
*.CPZ	An Inactive CPX

DO's:

1) DO remember to deallocate memory whenever appropriate.
2) DO use the XControl functions whenever possible.
That's why we put them there.
3) DO take the time to design an appealing user interface.
4) DO use graphics whenever possible instead of menu commands.
5) DO have OK and Cancel buttons available for each CPX.
Please note that it is 'Cancel' and NOT 'CANCEL'.
6) DO SHADOW Popup boxes. We want the user to know that 'shadowed' boxes are Popup boxes.
7) Treat AC_CLOSE as 'Cancel'
8) Treat WM_CLOSE as 'OK'
9) Treat a "SAVE" action as an 'OK' by updating the 'Cancel' variables.

DON'T's:

1) DON'T have the CPX object tree exceed the work area (256x176 pixels).
2) DON'T have CPXs stealing interrupt vectors.
3) DON'T mix XForm_do() calls with Call-CPX type functions.
4) DON'T forget to deallocate memory whenever appropriate. Not doing so will fragment the system memory.
5) DON'T use existing ID#s for existing CPXs that were not written by you.
6) DON'T forget to close a file if your CPX opens one.
7) DON'T forget to open and close a VDI workstation when needed. DO NOT open a VDI workstation and leave it open.

WORKSTATION NOTES and BRIEFS

When a CPX wishes to perform VDI functions, the CPX must open the workstation, perform its duties and then close the workstation immediately. The CPX must not leave any workstations open when it returns to accept more events. The handle passed to the CPX is the Physical Handle of the Control Panel returned by a graf_handle() call. The proper procedure of opening a workstation is:

```
work_in[0] = Getrez()+2;
for( i = 1; i < 10; work_in[i++] = 1 );
work_in[10] = 2;
vhandle = xcpb->handle;
v_opnvwk( work_in, &vhandle, work_out );
```

MEMORY ALLOCATION NOTES and BRIEFS

CPXs should not allocate any memory unnecessarily. If the CPX must allocate memory, the CPX should perform its operation and deallocate the memory immediately. An example of this is when calculating the amount of free memory. The reason CPXs should not allocate memory is because the allocated memory may be invalidated at any time by the OS. This can occur during a resolution change or when a process exits and returns to the desktop. On the TT, ALL memory is freed up during a resolution change, so memory fragmentation isn't a problem there.

INTERNAL USE ONLY

The following routines are for **INTERNAL USE ONLY**.

CPX GET HEADER NODE FUNCTION:

The call returns a pointer to the parent node of the CPX linked list structure.

Declared: void *reserved1

Actual: CPXNODE *Get_Head_Node(void);

CPXNODE is defined as:

```
typedef struct cpxnode
{
        char            fname[14];      CPX filename
        int             vacant;         1 = not vacant
        int             SkipRshFix;     Non-Zero, skip rsh_fix()
        long            *baseptr;       Pointer to cpx load area
        struct cpxnode  *next;          Pointer to next cpxnode
        CPXHEAD         cpxhead;        CPX head structure
        Prghead         prghead;        program header of CPX
}CPXNODE;
```

Prghead is defined as:

```
typedef struct _Prghead
{
        int     magic;
        long    tsize,
                dsize,
                bsize,
                ssize;
        int     fill[5];
} Prghead;
```

CPX SAVE HEADER FUNCTION:

The current header found in the node is written out to the node's file. If the file isn't found, XControl will search the remaining CPXs for a matching id number and version number. If found, that file will be the active CPX. If not found, a file not found alert error will be generated. Please see the CPXHEAD.H file for a description of the CPX HEADER.

Declared: void *reserved2

Actual: BOOLEAN Save_Header(CPXNODE *ptr);

IN: CPXNODE *ptr: Pointer to the cpx node

OUT: BOOLEAN: TRUE - AOK
FALSE - an Error has occurred.

STBook/STylus Expansion Bus
Electrical Specification

August 6, 1991

Power Available

External devices must not draw more than 400mA total from VCC on the connector. If the device is to operate on batteries alone, it should not draw more than 100 mA. It is good design practice to use CMOS wherever possible and to shut down power to any circuitry not in use.

Loading

External devices must not present more than a total of 1 (one) LS-TTL load per line onto the signals. This expansion bus is completely unbuffered, therefore, loading in excess of the recommended amount may cause the system to fail. Open-collector drivers should be prepared to sink 20mA, on those lines which require it, such as /EXPANSION_WAKE.

Signal Descriptions

The Atari STBook and STylus can be expanded externally using a 120-pin expansion bus, which is new to these machines. It essentially allows direct access to the 68HC000 address and data buses, and bus control signals to allow appropriate response. There are also the signals to allow for conversion to the previous ROM Cartridge format without the need for active electronics (i.e. a 120-pin expansion to 40-pin ROM cartridge convertor would consist of two connectors and a PCB).

The following signals are all direct from the 68HC000, and need no special description:

o A1-A23	Address Lines
o D0-D15	Data Lines
o /AS	Address Strobe
o /LDS,/UDS	Lower/Upper Data Strobes
o R/W	Read/Write Control
o FC0-FC2	Function Code 0-2
o /VPA	Valid Peripheral Address
o /VMA	
o E	E clock
o /RESET	Reset signal
o /HALT	Halt signal

Two signals are also direct from the 68HC000, but require a bit more operational detail:

o /DTACK	Data Transfer Acknowledge
o /BERR	Bus Error

The Combo chip uses /DTACK to acknowledge memory spaces it controls; it Bus Errors on other spaces (or illegal access to valid spaces) by not generating /DTACK. Other circuitry in the Combo chip times the length of the /AS signal; if it is longer than 16uS, than a /BERR is generated. What this means is that a device on the 120-pin expansion bus can be logically located in address spaces that the Combo chip considers illegal; all that is neccesary is to generate a /DTACK early enough such that /AS does not extend to 16uS.

- /ROM3
- /ROM4

These two signals are simply the outputs generated by the Combo chip for particular memory spaces, specifically those for the ROM cartridge space. Because the Combo assumes these are ROMs, only reads of this space are acknowledged or selected by the Combo chip. A third signal, /DEV, simply indicates when a peripheral address has been selected in supervisor mode; /DTACK is not neccesarily asserted.

- /DEV
- /DMA

There is also /DMA, which indicates that a Floppy or ACSI DMA cycle is occuring. It is included because the Combo chip, while asserting /AS and /L/UDS, leaves the address bus in a high-impedance state. Because of the high value pull-up resistors used in the STBook and STylus, the address lines may rise quite slowly when the lines are left in high-impedance. Noise could couple in, and false addresses could be asserted. It is therefore recommended that any address decoding added to the STBook or STylus use /DMA as an additional (active HIGH) qualifier.

o /BR	Bus Request
o /BGACK	Bus Grant Acknowledge
o /MCUBG	Bus Grant, out from the Combo chip.
o /CPUBG	Bus Grant, from the CPU to Combo chip (this is for reference only)

Use of the Bus Grant system is possible, with some limitations. While the Bus Request and Bus Grant Acknowledge are direct connections to the 68HC000, the Bus Grant signal is an output from the Combo chip. This means that the Combo chip (which includes the Blitter and DMA control) has priority for the gaining control of the Bus; Bus Grant is passed through only if no request is pending internal to the Combo.

- /EINT3
- /MFPIEI
- /MFPIEO
- /MFPINT
- /IPL0, /IPL1, /IPL2
- /IACK

Some interrupt control is also possible, at two seperate priority levels. One is a level 3 interrupt, for which an input into the Combo chip priority encoder is provided. For this level, it is the responsibility of the external circuit to respond to the interrupt acknowledge cycle, and to provide a method to clear the interrupt request. Both Auto-Vector and Vectored interrupts are possible.

The external circuitry can also share the Level 6 interrupt with the 68HC901 MFP internal to the STylus and STBook. The external interrupt source can have either higher or (preferably) lower priority than the internal MFP. All of this is accomplished with three signals: /MFPINT, /MFPIEI, /MFPIEO. The first is a open-collector driven, wire-OR signal, indicating a level 6 interrupt. The next two establish the relative priority of the two interrupt sources. /MFPIEI (MFP Interrupt Enable In) signals the MFP that no higher priority device is requesting the interrupt service (active LOW, internal pull-down). /MFPIEO signals that the MFP has no pending interrupts, and that /MFPIEI is active; i.e. no higher priority interrupt is pending. Thus, a multilevel structure can be obtained. Because many internal functions depend on the level 6 interrupts of the MFP, we recommend that external devices install themselves at a lower level, but do not require it.

To help in synchronization of external circuits (particularly when the Refresh Machine described above is running), a small number of clock signals are provided. They are:

o CLK16	main 16MHz clock
o CLK8	8MHz CPU clock
o KHZ500	500 KHz Baud Rate Clock

Some power and power control signals are provided to allow external devices to draw some power from the VCC supply of the STylus or STBook. To help distribute the power evenly, and to help maintain clean logic levels, there are 10 VCC signals, and 30 GROUND signals. 10 of the GROUND signals are located at the ends of the connector, opposite the VCC signals; the other 20 are distributed as every 5th pair of signals accross the connector. This should aid in both maintaining a clean ground, and reducing EMI.

Power Control is possible to some degree using the signal /EXPANSION_WAKE. This signal expects to be driven by an open-collector driver; when pulled to ground, this powers on the STylus/STBook. It is equivalent to pressing the Power button on either machine.

Finally, there is a pin which allows a peripheral plugged into the STBook or STylus to determine which it is connected to. Pin 94 is defined to be a no-connect on an STBook, and grounded on a STylus. The peripheral could, conceivably, determine the type of host without the host being powered; this is the responsibility of the peripheral, if it needs to know it.

e Expansion connector has the following pin assignments:

```
---- Expansion Port Pin Assignments ----------------

 STBook         micro-D 120S
                 ----------------------------------
          1 |-- VCC                     GND --| 61
          2 |-- VCC                     GND --| 62
          3 |-- VCC                     GND --| 63
          4 |-- VCC                     GND --| 64
          5 |-- VCC                     GND --| 65
          6 |-- D0                       D1 --| 66
          7 |-- D2                       D3 --| 67
          8 |-- D4                       D5 --| 68
          9 |-- D6                       D7 --| 69
         10 |-- GND --------------      GND --| 70
         11 |-- D8                       D9 --| 71
         12 |-- D10                     D11 --| 72
         13 |-- D12                     D13 --| 73
         14 |-- D14                     D15 --| 74
         15 |-- GND --------------      GND --| 75
         16 |-- NC                       A1 --| 76
         17 |-- A2                       A3 --| 77
         18 |-- A4                       A5 --| 78
         19 |-- A6                       A7 --| 79
         20 |-- GND --------------      GND --| 80
         21 |-- A8                       A9 --| 81
         22 |-- A10                     A11 --| 82
         23 |-- A12                     A13 --| 83
         24 |-- A14                     A15 --| 84
         25 |-- GND --------------      GND --| 85
         26 |-- A16                     A17 --| 86
         27 |-- A18                     A19 --| 87
         28 |-- A20                     A21 --| 88
         29 |-- A22                     A23 --| 89
         30 |-- GND --------------      GND --| 90
         31 |-- /HALT               /STylus --| 91
         32 |-- /VMA                 /CPUBG --| 92
         33 |-- /BR                  /MCUBG --| 93
         34 |-- /BGACK                   NC --| 94
         35 |-- GND --------------      GND --| 95
         36 |-- FC0                     FC1 --| 96
         37 |-- FC2                     /AS --| 97
         38 |-- R/W                    /LDS --| 98
         39 |-- /UDS                 /DTACK --| 99
         40 |-- GND --------------      GND --| 100
         41 |-- /RESET                 /VPA --| 101
         42 |-- /IPL0                 /IPL1 --| 102
         43 |-- /IPL2                 /IACK --| 103
         44 |-- /EXPANSION_WAKE       /BERR --| 104
         45 |-- GND --------------      GND --| 105
         46 |-- /MFPINT             /MFPIEI --| 106
         47 |-- /EINT3              /MFPIEO --| 107
         48 |-- /DMA                   /DEV --| 108
         49 |-- /ROM3                 /ROM4 --| 109
         50 |-- GND --------------      GND --| 110
         51 |-- NC                       NC --| 111
         52 |-- NC                       NC --| 112
         53 |-- CLK16                  CLK8 --| 113
         54 |-- KHZ500                    E --| 114
         55 |-- GND --------------      GND --| 115
         56 |-- VCC                     GND --| 116
         57 |-- VCC                     GND --| 117
         58 |-- VCC                     GND --| 118
         59 |-- VCC                     GND --| 119
         60 |-- VCC                     GND --| 120
                 ----------------------------------
```

GND

VCC

```
        ****
        ****
        ****
       ******
      ** ** **
     **  **  **
   **    **    **
```

Atari Corporation
Personal Workstation Group
Atari Texas
Carrollton, Texas

WORKING DRAFT

Atari Mega STe Plus Product Specification

8 June 1990

1. INTRODUCTION

The MegaSTe is the newest enhancement in the series of Atari ST (tm) compatible 68000-based computers. It is upward compatible with the Atari STe, and includes a number of enhanced features, including 16K bytes of cache memory and the capability of running at 16MHz.

The hardware specifications of the MegaSTe computer are as follows:

- Motorola MC68000 at 8MHz or 16MHz, software selectable
- 16K Bytes (8K Words) of 16-bit cache, enabled under software control
- Motorola MC68881/68882 Floating Point Coprocessor (optional/socketed)
- RAM: 2 or 4 Mbyte of dual-purpose (video/system) RAM
- ROM: 2 socketed ROMs, providing 64K, 128K, or 256K of ROM space (strappable).
- internal video modes that are compatible with those in the Atari ST series-- Color: 320x200, 640x200. Monochrome: 640x400.
- an industry standard analog RGB/monochrome color monitor interface.
- parallel I/O port, implemented using the one of the parallel ports on the General Instruments AY-3-8910 / Yamaha YM-2149 sound chip
- 1 low-speed async serial I/O port (implemented using a 68901 MFP)
- 2 high-speed SDLC serial I/O ports (from a Zilog 8530 SCC), one port of which can be strapped to be a medium-speed LAN interface.
- ST/MEGA compatible intelligent keyboard, with mouse and joystick ports
- Atari ACSI DMA channel (for Atari Hard Disk, Laser Printer, CD-ROM, etc)
- floppy disk controller and interface sharing the ACSI DMA channel
- Musical Instrument Digital Interface (MIDI)

- Atari ST compatible cartridge port (128 Kbyte storage)
- VMEbus for expansion: 1 single Eurocard A24/D16 slave-only interface

2. MAIN SYSTEM

The MegaSTe is intended to be a compatible, high-performance extension of the Atari ST architecture. By including the VMEbus the system can be expanded for future needs.

2.1. Processor

The MegaSTe uses a Motorola 16-MHz 68000[1] 16/32-bit microprocessor.

Support circuitry built around the 68000 provides 16KBytes (8K words) of 16-bit wide cache memory. It also allows switching the processor clock between 16.02 MHz and 8.01 MHz. Both cache control and processor clock control are provided by control bits in a System Control Unit register (I/O address 8E21 Hex). The cache memory is only enabled when the processor is being clocked at the high clock speed, since it provides no saving at the lower clock speed.

2.2. Floating Point Coprocessor

The MegaSTe design has a socket for an optional Motorola MC68881 or the newer, higher-performance, MC68882. These two parts are hardware compatible. There is a slight software difference in the size of the exception stack frames, but it is possible to write software that will run transparently with either part.

The floating point operations are performed in accordance with IEEE Standard 754, with both 32-bit (single) and 64-bit (double) precision external access.

The floating point coprocessor is clocked at 16 MHz, regardless of the clock speed of the main processor. Since the MC68000 does not support the integrated co-processor interface, it is necessary to access the floating point processor as an I/O device, and read and write its internal registers by memory transfer instructions.

2.3. ROM

The system includes on-board sockets for a set of two 1Mbit ROMs, providing a total of 256KB ROM. Since system bus access is 16-bits wide, both ROMs must be present.

---

[1] MC68000, MC68881, and MC68882 are trademarks of Motorola, Inc.

Jumpers are provided to allow the use of 27256, 27512, 27010/27C1001, and 571001/27C1000 EPROMs, in addition to 531000 ROMs. The default jumper position allows the use of 27512 EPROMs (for a total of 256 Kb of ROM) as well as 571001/27C1000 EPROMs or 531000 ROMs (for a total of 512 Kb of ROM). 32 pin sockets are provided, although 27256, 27512, and 531000 only use the bottom 28 pins.

3 jumpers (W201, W202, and W203) are provided adjacent to the ROM sockets to allow selection of different types of ROM/EPROMs. The following table shows the appropriate connections to be made:

Part Type	W201	W202	W203
27256	Don't Care	1 and 2	N/C
27512	2 and 3	2 and 3	N/C
571001/ 27C1000/ 531000	2 and 3	2 and 3	2 and 3
27010/ 571000/ 27C1001	1 and 2	2 and 3	1 and 2

An image of the first 8 bytes of ROM resides in the first 8 bytes of the RAM memory. These first 8 bytes (0x00000000-0x000007) are accessible only in supervisor mode. Attempts to read from this area in user mode or any write results in a bus error. The full ROM resides at the memory location 0x00E00000 - 0x00EFFFFF.

Among the tasks this ROM perform are system initialization and boot code that can boot from a floppy, ACSI device, or network. The ROM is expected to contain a multi-lingual implementation of TOS. Moreover, if sufficient space is available, ROM-based service diagnostics will be provided.

2.4. RAM

The basic system includes 2 or 4 Mbytes of dual-purpose RAM which is used for both video and system memory. These are implemented using 8-bit wide SIMMs (Single Inline Memory Modules), which must be used in pairs.

The bus architecture is similar to the ST in that memory access cycles are interleaved between the MPU and the video controller in 250 nS RAM time slices, thus allowing video display memory to reside efficiently as part of main memory. During active display cycles the processor is prevented from accessing the memory but is allocated the next 250 nS time slice.

Additional memory can be installed in the system by plugging in VME memory cards. Up to 4 MBytes (actually, 4 Mbytes minus 64K) of A24/D16 memory can be addressed on the VME bus beginning at address A00000 hex. The VME RAM cards will run slightly slower than the system RAM as all VME accesses incur an extra wait state per bus cycle.

The MC68000 accesses to on-board RAM typically require 4 clock cycles.

There is no provision for parity or ECC protection on the system RAM. The reliability of current DRAM technology makes this unnecessary. However, such features could be included in add-on VME cards.

The first 0x800 bytes (2K) of RAM (0x00000008-0x000007FF) are accessible only in supervisor mode. Attempts to read or write to this area in user mode results in a bus error.

RAM memory cycles are cached using 8K words of 16-bit wide, 45 nS cache memory. This permits cache-hit memory cycles to run with no wait states, and without taking a cycle on the system bus, when the processor is running at 16-MHz.

2.5. System Control Unit

The System Control Unit (SCU) provides an additional level of interrupt control for the system. It also contains registers that allow the software generation of interrupts, and control of the processor clock speed and cache memory operation. All of the SCU registers are reset at power-on and by the reset pushbutton.

2.5.1. Interrupt Mask and Current Status

The SCU contains two mask registers that permit independent control over which interrupt levels will be seen by the processor. One register masks interrupts generated on the system board and the other masks VMEbus sources. These registers are cleared at power-up or reset, disabling all interrupts.

There are also interrupt request registers that show the current state of the seven interrupt request levels from each of the sources. This register shows the physical status of the interrupt lines before they are ANDed with the SCU's mask register.

2.5.2. System Control Registers

The SCU also contains two read/write registers that can be used for system configuration information.

2.5.3. Interrupt Generator

The system can write to an I/O address to generate a low priority (level 1) interrupt to the 68000. This I/O address contains a read/write status/control port, only the least significant bit of the least significant byte is defined. When set to 1, it generates an autovectored level 1 interrupt. When cleared, the interrupt request is taken away.

The SCU is hardwired so that:

- only interrupts 5 and 6 have external IACK pins and are capable of generating vectored interrupts on the motherboard.
- SCU generated IRQ1 and IRQ3 are hardwired to the corresponding priorities and are always autovectored.
- VMEbus ACFAIL generates a system (motherboard) IRQ7 to the MPU. The only other source of an IRQ7 is a VMEbus card.

2.5.4. Bus Timer

The SCU also implements a system bus timer. If nothing concludes a bus cycle within 16 microseconds, the SCU will signal a bus error.

2.5.5. Processor/Cache Control

The process/cache control register implements the low order two bits to control processor clock speed and enabling of the cache memory. Bit 0 enables the cache when it is set to a one. Bit 1 enables the high speed (16.02 MHz) processor clock when it is set to a one. Both bits are set to zero on power-on, or when the reset button is pushed. The cache tag registers are cleared when the cache is disabled, resulting in no cache-hits until there have been valid memory reads with the cache enabled.

2.6. Floppy/ACSI Interface

The ST compatible Floppy/ACSI subsystem interfaces between RAM and ACSI compatible peripherals, such as the SLM804 laser printer, SHxxx/Megafile hard disk drives, and Atari CD-ROM. This DMA channel is shared with the internal floppy disk controller.

2.7. Real Time Clock

The MegaSTe system includes a Ricoh RP5C15 Real Time Clock chip. This provides time of day (down to one second resolution) and date. The RTC is provided with a 32.768 kHz oscillator that is independent of all other system clocks.

The chip is accessed through 32 4-bit registers accessed in two banks. Bank 0 allows reading and setting each digit of the date and time, and also allows access to test and control registers. Bank 1 allows setting the digits of an alarm function, and controlling the mode of operation of the clock chip.

2.8. Configuration Switch Register

The MegaSTe implements an 8-bit configuration switch register to indicate the presence or absence of options. Depending on printed circuit board layout, the register may be implemented using an 8-bit DIP switch, solder pads, or double "row of stakes" jumpers. A bit will read as a "1" if the circuit is open As of this writing, the following bits have been assigned meanings:

Configuration Bits

Bit	Meaning
7	0 => No DMA sound hardware is installed. 1 => DMA Sound hardware is available.
6	0 => High speed (16 MHz) 1772 Floppy Disk controller is installed. 1 => Only low speed (8 MHz) 1772 Floppy Disk controller is installed.
5-0	Undefined, reserved.

3. Device Subsystems

The MegaSTe architecture supports the following device subsystems:

- ST compatible ACSI
- floppy disk interface sharing the ST "ACSI" DMA channel
- high-speed serial ports and a low speed network port through the SCC chip
- one additional serial port and an external interrupt port connected to MFP controller
- a Centronics parallel printer port driven by the Yamaha YM-2149 sound chip
- a ST/MEGA compatible intelligent keyboard, mouse, and joystick interface
- a port supporting application and diagnostic cartridges

3.1. ACSI

3.2. Floppy Disk

The MegaSTe series floppy disk subsystem is designed around the WD1772 Floppy Disk Controller supporting up to two daisy-chained floppy disk drives (drive 0 or 1). A higher speed version of the 1772 is planned to allow 1.44Mb (formatted) capacity drives. The MegaSTe is designed for one internal floppy disk drive and one external drive (such as the SF314).

The subsystem interfaces to the RAM through the ACSI DMA controller. Commands and arguments are sent to the FDC by first writing to the DMA Mode Control Register to select the desired FDC register and then writing the data bytes.

The standard floppy for the MegaSTe series is the 3.5 inch floppy disk with the capacity of 720 Kbyte (formatted). The 1.44Mb drives will be available as an option.

The internal drive cabling supports the DiskChangeLine signal from the floppy drive(s) to a bit on MFP-2. DiskChangeLine can be read when the drive is selected, and is asserted when (1) power is applied or (2) a diskette is removed from the drive. The signal is cleared by issuing a step command to the drive.

3.3. High Speed Serial Ports

The Zilog 85C30 SCC, a dual channel, multi-protocol data communications peripheral, is included in the MegaSTe design to provide two serial ports (ports A and B).

Port A can be used as either a network port or a standard low speed RS232C port. When bit 7 of the GI Sound Chip port A is a 0, LAN mode is selected. The input/output of Port A is routed to the appropriate connector: (1) if RS232C mode is selected, the port is connected to a 10-pin header (that can be connected via ribbon cable to a DB-9P) or (2) if the network port is selected, it is connected to an 8-pin mini-DIN connector. The output pins on the unselected port remain inactive.

The channel A Wait/Req line is connected to the DTACK circuitry in such a way that, when programmed for operation as a WAIT line, it will allow high speed block reads or writes using a tight software loop, e.g.

```
   loop: move.b SCCDATA, (a1)+;    Read data, store in buffer,
                              ;    advance pointer
         dbra d1, loop        ;    repeat for byte count
```

The WAIT line holds off the DTACK and prevents the read cycle from completing until the data is available. CAUTION: If the cycle does not complete within 16 microseconds (the length of the bus timeout), a Bus Error will occur. This means that this technique can only be used at speeds in excess of approximately 500K baud.

The SCC handles both asynchronous formats and synchronous byte-oriented protocols such as HDLC and IBM's SDLC.

Port B is configured to be a low speed RS232C serial port that can be used for connecting to a modem or a local mainframe. It is pinned out on a DB-9P connector in a way that is compatible with the Atari PC4. Modem control signals are derived directly from the 85C30 port B control lines. This port can operate with split transmit and receive baud rates.

The PCLK input to the SCC is 8 MHz. The RTxCA and RTxCB input is provided with a 3.672 MHz clock. The input to TRxCA comes from the low speed LAN connector. TRxCB is run at 2.4576 MHz.

3.3.1. SCC RS232 Port Pinout

The SCC RS232 serial ports are pinned out in DB-9P connectors in a way that is compatible with the Atari PC4. On the MegaSTe, the SCC port A RS232 connections are routed to a header on the motherboard. That header can be connected with a ribbon cable to a nine pin D connector located on the VME slot cover.

SCC RS232 Pinouts

pin	Port A (RS232 Mode)	Port B
1	Carrier Detect (I)	Carrier Detect (I)
2	Receive Data (I)	Receive Data (I)
3	Transmit Data (O)	Transmit Data (O)
4	Data Terminal Ready (O)	Data Terminal Ready (O)
5	Ground	Ground
6	Data Set Ready (I)	Data Set Ready (I)
7	Request to Send (O)	Request to Send (O)
8	Clear to Send (I)	Clear to Send (I)
9	--	--

3.3.2. LAN Connector Pinout

The moderate speed LAN connector is an 8 pin female mini-DIN.

SCC LAN Pinout (Port A)

pin	function
1	Output Handshake (DTR, RS423)
2	Input Handshake/External Clock
3	Transmit Data -
4	Ground
5	Receive Data -
6	Transmit Data +
7	<reserved>
8	Receive Data +

3.4. MFP

One 68901 Multi-Function Peripheral (MFP) controllers are used to provide system timers, a low speed RS232C serial ports, and an interrupt controller. The MFP is used in a way that is compatible with the ST. It provides both a serial port and interrupt control.

The baud rate clock for the MFP serial transmitter and receiver is derived from the timer D output of the MFP. Given the MFPs´ 2.4576 MHz clock, baud rates up to 19.2 Kbaud can be supported on the serial port.

3.4.1. MFP Serial Port Pinouts

The MFP serial port is pinned out in a DB-9P connector in a way that is compatible with the Atari PC4. The MFP serial ports has a complete complement of modem control lines compatible with the ST, but pinned out in a 9 pin D connector.

MFP Serial Port Pinout

pin	MFP-ST
1	Carrier Detect (I)
2	Receive Data (I)
3	Transmit Data (O)
4	Data Terminal Ready (O)
5	Ground
6	--
7	Request to Send (O)
8	Clear to Send (I)
9	Ring Indicator (I)

Note: The Ring Indicator (RI) signal is connected to bit 6 of the MFP General Purpose I/O Port (GPIP).

3.5. Parallel Printer Port

The MegaSTe architecture includes a bi-directional 8-bit parallel printer port that implements a subset of the Centronics standard. This interface is through the General Instruments AY-3-8910 / Yamaha YM-2149 Programmable Sound Generator (PSG) chip. It is pinned out in a DB25 in a way that is a subset of the Atari PC4. The Centronics STROBE signal is generated from a PSG bit. The Centronics BUSY signal from the printer connects to one of the parallel input lines of the MFP to permit interrupt driven printing. Eight bits of read/write data are handled through I/O port B on the PSG at a typical data transfer rate exceeding 4000 bytes/second.

3.6. Keyboard Interface

The MegaSTe keyboard interface is completely compatible with the ST/MEGA computers. The keyboard is equipped with a combination mouse/joystick port and a joystick only port. The keyboard transmits encoded make/break key scan codes (with two key rollover), mouse/trackball data, joystick data, and time-of-day. The keyboard receives commands and sends data via bidirectional communication implemented with a MC6850 Asynchronous Communications Interface Adapter (ACIA). The data transfer rate is 7812.5 bits/second. All keyboard functions, such as key scanning, mouse tracking, command parsing, etc. are performed by a HD6301V1 8-bit microcomputer unit. (See the Atari, Intelligent Keyboard (ikbd) Protocol, February 26, 1985.)

3.6.1. Mouse and Joystick Interface

The Atari two-button mouse is a mechanical, opto-mechanical, or optical mouse with the following minimal performance characteristics: a resolution of 100 counts/inch, a maximum velocity of 10 inches/second, and maximum pulse phase error of 50%. The joystick is a four direction switch-type joystick with one fire button.

3.7. ROM Cartridge

The MegaSTe's cartridge port is fully compatible with ST cartridges. The cartridge is physically connected through a 40 pin card edge connector ROM cartridge slot. Cartridge ROMs are mapped to a 128K memory region starting at 0x00FA0000, extending to 0x00FBFFFF.

4. Video Subsystem

The MegaSTe video subsystem is identical to that of the STe. The resolution modes are identical with that of the ST and Mega ST series, but the functionality is enhanced by permitting video display memory to start on any even word boundary (instead of only on 256 byte boundaries), and by adding the capability of scrolling horizontally on a one-pixel basis. The MegaSTe also contains four bits of information for each color, instead of three.

4.1. Video Configuration

The various modes available on the MegaSTe are:

ST mode mode bits	resolution	planes	palette (CLUT entries)	colors DACs
00	320x200	4	16	4096/4-bits
01	640x200	2	4	4096/4-bits
10	640x400	1	-	Monochrome

As the table indicates, the modes are set through either the Shift Mode Register. 16 word-wide registers comprise the ST Color Palette (also known as the Color LookUp Table - CLUT). Contained in each entry are 12 bits of color: 4-bits each for red, green, and blue. Therefore, a total of 4096 possible color combinations (16 x 16 x 16) are selectable for each entry. However, in order to maintain compatibility with the ST (which had 3 bits for each color, right aligned within nybbles), the high-order bit of each color is actually the least significant bit of color information. The following diagram compares the CLUT entries for the ST/Mega ST and the STe/MegaSTe:

```
       1 1 1 1   1 1
       5 4 3 2   1 0 9 8   7 6 5 4   3 2 1 0

ST :   x|x|x|x | x|2|1|0 | x|2|1|0 | x|2|1|0|
                   Red       Green      Blue

STe:   x|x|x|x | 0|3|2|1 | 0|3|2|1 | 0|3|2|1|
                   Red       Green      Blue
```

Mode 00 (320x200x4) can index all sixteen palette colors, while mode 01 (640x200x2) can index just the first four (Reg0 - Reg3) palette colors. The monochrome mode (10 - 640x400x1) bypasses the color palette and is instead provided with an inverter for inverse video controlled by bit 0 of palette entry 0.

4.2. Video RAM/Controller/Display Interface

Video display memory is configured as logical planes (1, 2, 4, or 8) of interwoven 16-bit words of contiguous memory to form one 32,000 byte physical plane starting at any 8 byte boundary. The starting address of display memory is loaded into the Video Base High, Video Base Mid, and Video Base Low Registers. This register is loaded into the Video Address Counter (High/Mid/Low) at the beginning of each frame. The address counter is incremented as the Bit-Map planes are read.

BitMap planes are transferred to the video chip (shifter) buffer 16-bits at a time. The shifter then loads the video shift register where one bit from each plane is shifted out and collectively used as the index (plane 0 appears first in RAM and provides the least significant bit of each pixel) to a specific ST Palette Register (depending on the Shift Mode).

4.2.1. Horizontal Scrolling

Two additional registers serve to implement a horizontal smooth scroll capability. The horizontal pixel scroll register specifies a pixel offset of 0 to 15 at which to begin display. Increasing this value by one will have the effect of scrolling the entire display one pixel to the left. The other register is the extra line width register. This register contains a number of words that is added to the ending address of each display line to get the beginning address of the next display line. It has the effect of putting an undisplayed area to the right of the video screen. By varying the horizontal pixel scroll register and the video base registers, the display video screen can be used as a horizontally scrollable "window" into that area.

4.3. Monitor Connector

The video output is provided on a 13-pin DIN connector compatible with the ST, STe, and Mega ST series computers. Either Color or Monochrome monitors can be used.

Pin	Function
1	Audio Out
2	Composite Video
3	General Purpose Output (??)
4	Monochrome Monitor Detect
5	Audio Input
6	Green
7	Red
8	Peritel Power (??)

9	Horizontal Sync
10	Blue
11	Monochrome Out
12	Vertical Sync
13	Ground

5. Music Subsystem

The MegaSTe architecture extends the music subsystem presently available on the ST/MEGA computers. The MegaSTe mixes the output of the existing ST PSG sound system with a new DMA-driven dual-channel D-to-A subsystem. The MegaSTe combines these two sources for simple beeps and sends the resulting audio through the audio out line of the monitor interface. In addition, the output can be connected to an external stereo amplifier for high-fidelity sound.

The MegaSTe is also equipped with a Musical Instrument Digital Interface (MIDI) which provides high speed serial communication of musical data to and from more sophisticated synthesizer devices.

5.1. Programmable Sound Generator

The ST sound system using the General Instruments AY-3-8910 / Yamaha YM-2149 Programmable Sound Generator is present in the MegaSTe. The YM-2149 Programmable Sound Generator produces music synthesis, sound effects, and audio feedback. With an applied clock input of 2 MHz, the PSG is capable of providing a frequency response range between 30 Hz (audible) and 124 KHz (post-audible). The generator places minimal amount of processing burden on the main system (which acts as the sequencer) and has the ability to perform using three independent voice channels. The three sound channel outputs are mixed together and sent to the volume and tone control chip.

(Reference Engineering Hardware Specification of the Atari ST Computer System, page 10.)

5.2. DMA Sound

The MegaSTe also includes a new DMA-driven sound subsystem that allows the playback or synthesis of complex waveforms at a variety of sampling rates.

5.2.1. Overview

Sound in the form of digitized samples is stored in system memory. These samples are fetched from dual-purpose memory during horizontal blanking (transparent to the processor) and provided to a digital-to-analog converter (DAC) at a constant sample frequency specified by the user. The output of DAC is then low pass filtered to a frequency equal to forty percent of the sample frequency by a four pole switched capacitor low pass filter. The signal is further filtered by a two pole fixed frequency (15 kHz) low pass filter and provided to a National LMC1992 Volume / Tone Controller. Finally, the output of this device is available at a pair of RCA jacks and the audio out line of the video connector.

Two channels are provided. They are intended to be used as the left and right channels of a stereo system when using the raw audio outputs from the machine. Of course, they are mixed together when fed to the video monitor speaker. A mono mode is provided which will feed the same data to both channels simultaneously. The only restriction placed on mono mode is that there must be an even number of samples (see data format section for details).

5.2.2. Data Format

Each sample is stored as an eight bit quantity, the most significant bit is the sign and the other seven bits are magnitude. In the stereo scheme there is one word per sample, the upper byte contains the left channel sample and the lower byte contains the right channel sample. In the mono scheme bytes are accessed sequentially. However, they are still fetched a word at a time. Therefore, there must be an even number of samples.

A group of samples is called a frame. A frame may be played once or can automatically be repeated forever. Frames occupy a contiguous block of memory and are specified by their starting and ending addresses. The ending address is the address of the last sample + 2. An external clock is provided to timer A of the ST MFP at the end of each frame. This can be used as an interrupt. This pulse is also exclusive OR'ed with the monochrome monitor detect bit, whose transistion can generate an interrupt on bit 7 of the MFP-ST General Purpose I/O Port. Frames may be linked together by defining a new frame while the current frame is being played. The new frame will begin at the end of the current frame.

As an example, suppose you have three frames (A, B, and C) and we want to play frame A once, then play frame B 5 times, and finally play frame C twice. To accomplish this you can do the following:

1. Setup frame A.
2. Write 3 to the sound DMA control register to start playing with repeat.
3. Setup timer A to use an external clock, initialize its count to 5, and have it interrupt when count = 0.
4. Setup frame B.

5. Go do something else until interrupted.

6. Setup frame C.

7. Setup timer A count to 2.

8. Go do something else until interrupted.

9. Write 1 to the sound DMA control register to cause playing to stop at the end of the frame.

In this example no mention is made of setting the sample rate, volume or tone controls. It's assumed that all of these have been set up ahead of time. It should be obvious how this example can be extended to allow volume or tone to be modified at specific points during playback.

Note If we had loaded the sound DMA control register with a 1 in step 2, frame A would have been played once and sound would have been disabled. A zero can be written to the sound DMA control register at any time to stop playback immediately.

5.2.3. MICROWIRE Interface

The MICROWIRE interface provided to talk to the National LMC1992 Computer Controlled Volume / Tone Control is a general purpose MICROWIRE interface to allow the future addition of other MICROWIRE devices. For this reason, the following description of its use will make no assumptions about the device being addressed.

The MICROWIRE bus is a three wire serial connection and protocol designed to allow multiple devices to be individually addressed by the controller. The length of the serial data stream depends on the destination device. In general, the stream consists of N bits of address, followed by zero or more don't care bits, followed by M bits of data. The hardware interface which has been provided consists of two 16 bit read/write registers. One data register which contains the actual bit stream to be shifted out and one mask register which indicates which bits are valid.

Let's consider a mythical device which requires two address bits and one data bit. For this device the total bit stream is three bits (minimum). Any contiguous three bits of the register pair may be used. However, since the most significant bit is shifted first, the command will be received by the device soonest if the three most significant bits are used. Let's assume: 01 is the device's address, D is the data to be written, and X's are don't cares. Then all of the following register combinations will provide the same information to the device.

```
1110 0000 0000 0000   Mask
01DX XXXX XXXX XXXX   Data

0000 0000 0000 0111   Mask
XXXX XXXX XXXX X01D   Data

0000 0001 1100 0000   Mask
XXXX XXX0 1DXX XXXX   Data

0000 1111 1111 0000   Mask
XXXX 01XX XXXD 0000   Data

1111 1111 1111 1111   Mask
01XX XXXX XXXX XXXD   Data
```

The mask register needs to be written before the data register. Sending commences when the data register is written and takes approximately 16uS. Subsequent writes to the data and mask registers are blocked until sending is complete. Reading the registers while sending is in progress will return a snapshot of the shift register shifting the data and mask out. This means that you know it is safe to send the next command when these registers (or either one) return to their original state. Note that the mask register does not need to be rewritten if it is already correct. That is, when sending a series of commands the mask register only needs to be written once.

5.2.4. Volume and Tone Control

The LMC1992 is used to provide volume, tone, and mixing control. This part is talked to using the MICROWIRE interface. The device has a two bit address field, address = %10, and a nine bit data field. There is no way of reading the current settings.

The input selector is used to enable and disable mixing the output of the GI PSG with the DMA sound. After reset, the input is grounded, and should be switched to either states 1 or 2 during initialization to avoid level mismatches during later switching.

Data Field

```
011 DDD DDD   Set Master Volume
    ||| |||
    000 000   -80 dB
    010 100   -40 dB
    101 XXX     0 dB

101 XDD DDD   Set Left Channel Volume
     || |||
            00 000   -40 dB
            01 010   -20 dB
            10 1XX     0 dB

    100 XDD DDD   Set Right Channel Volume
         || |||
         00 000   -40 dB
         01 010   -20 dB
         10 1XX     0 dB

    010 XXD DDD   Set Treble
          | |||
          0 000   -12 dB
          0 110     0 dB (Flat)
          1 100   +12 dB

    001 XXD DDD   Set Bass
          | |||
          0 000   -12 dB
          0 110     0 dB (Flat)
          1 100   +12 dB

    000 000 0ss   GI PSG Sound Enable
             ||
             00   disabled, unbiased
                   (reset state)
             01   enabled
             10   disabled, biased
```

Note: The volume controls attenuate in 2 dB steps. The tone controls attenuate in 2 dB steps at 50 Hz and 15 kHz.

5.3. Musical Instrument Digital Interface (MIDI)

The MIDI allows the integration of the MegaSTe series with music synthesizers, sequencers, drum boxes, and other devices possessing MIDI interfaces. High speed (31.25 Kbaud) serial communication of keyboard and program information is provided by two ports, MIDI OUT and MIDI IN (the MIDI OUT also includes MIDI THRU data).

The MIDI communicates through the MC6850 Asynchronous Communications Interface Adapter (ACIA) to the system bus. The data transfer rate is a constant 31.25 Kbaud of 8-bit asynchronous data.

(Reference Engineering Hardware Specification of the Atari ST Computer System, pages 11 and 17 for more information on the MIDI and ACIA.)

6. VMEbus

The MegaSTe provide for I/O expansion by implementing the industry standard VMEbus, revision C.1. The MegaSTe has one single-high VMEboard backplane. The interface is limited to A24/D16 slave-only cards

6.1. System Controller

The main system board serves as the VMEbus system controller (a slot 1 "card") and implements the following functions:

- IACK* daisy-chain driver
- global SYSCLK (16 MHz, independent of processor speed)
- global VMEbus time-out that drives BERR*

The IACK* daisy-chain driver is designed to meet the VMEbus specification requirements.

The interface is compatible with the VME specification, but the following constraints should noted:

- No bus arbitration is supported. BR0*, BR1*, BR2*, and BR3* are connected together and pulled up by a 1K resistor to Vcc. BG0IN*, BG1IN*, BG2IN*, and BG3IN* are connected together and pulled up by a 1K resistor to Vcc. BBSY* and BCLR* are each pulled up by a 1K resistor to Vcc but are not otherwise driven. BG0OUT*, BG1OUT*, BG2OUT*, and BG3OUT* are not connected.

- The interrupt lines IRQ1* through IRQ7* can each be used, and are each pulled up by a 1K resistor to Vcc. IRQ3*, IRQ5*, and IRQ6* can also be driven low by the system. The SYSFAIL* signal is also pulled up by a 1K resistor and can generate a level 7 system interrupt when asserted by a card. IACK* and IACKIN* are driven by the system. A card should not drive these signals. IACKOUT* is not connected. The status word supplied by the card during the interrupt acknowledge cycle is used as the 68030 interrupt vector. For compatibility with Atari products, the vector supplied must not be 0xFF. All VME bus and system interrupts are independently maskable in the SCU.

- SYSCLK is driven with a 16.021226 MHz clock, independent of which CPU clock speed is selected. SERCLK and SERDAT* are not connected. The ACFAIL* signal is driven low by the system when the power supply is not stable. ACFAIL* will be asserted 1 mS before the power supply leaves the regulated range. It is pulled up by a 1K resistor.

- AM0, AM1, AM2, and AM4 are driven by the system. AM3 and AM5 are connected together and pulled up by a 1K resistor to Vcc. This implementation allows standard Supervisor and Non-Privileged Program and Data accesses, and Short Supervisory and Non-Privileged accesses. Block transfers are not supported. LWORD* is pulled up by a 1K resistor but not otherwise driven.

- The BERR* and SYSRES* signals are connected directly to the system bus error and reset signals. The bus error timer implemented on the system board will time out and generate a bus error if the card does not assert DTACK* within 255 cycles of the 16MHz clock after the VME AS* falls. The SYSRES* generated when the processor executes a RESET instruction may be as short as 16 uS long. Both signals are pulled up by 1.2K resistors and can be driven low by the system as well as by the card. The +5VSTDBY signal is connected to +5V. DTACK* is pulled up by a 1K resistor.

- All other signals on the connectors comply with VME functionality, but with electrical limitations on current drive and termination. There is no termination in the system other than the pullups specifically mentioned above. All outputs have at least 1 LSTTL drive capability and no input presents more than 2 LSTTL loads.

6.2. Address Partitioning

The MegaSTe's A24/D16 VMEbus interface is fixed at 0x00A00000-0x00DEFFFF. A16 cards are addressed from 0x00DF0000 - 0x00DFFFFF.

6.3. Read-Modify-Write Cycles

The bus can not be arbitrated away from the 68000 if it is in the midst of a read-modify-write cycle.

6.4. VME Interrupter

The system can write to an I/O address to generate a level 3 interrupt on the VMEbus. It can monitor a status register that indicates when that interrupt has been acknowledged and serviced. An I/O address contains a read/write status/control port, only the least significant bit of the least significant byte is defined. When set to 1, it generates a VMEbus level 3 interrupt. When cleared, the interrupt request is taken away.

Note that the level 3 interrupt must be masked off (either by setting the processor's IPL or by masking the interrupt in the system controller) or the 68000 will be immediately interrupted.

The system board responds to a VMEbus interrupt acknowledge cycle with the status ID of 0xFF.