Advanced AMIGA Architecture

Commodore Confidential

June 18, 1992

PREFACE

This document has been written with the intent of setting down on paper all of the workings of the next generation AMIGA chip set. It is the result of collaboration among members of IC, Systems, and Software design engineering groups. This is the sixth draft of the Advanced Amiga Architecture chip spec.

Comments are invited.

NOTES

CONTENTS

move to category: "ANDREA Testing Considerations" then add: "Syncronizing the Clocks"

1 INTRODUCTION.

The original AMIGA 1000 was designed to bring high performance state of the art technology to the general public. The machine included capabilities that were unattainable in other machines of its class, even at twice the price. However time moves on. Our competitors are now marketing systems with improved capabilities. In many cases, the AMIGA is no longer the obvious choice. Indeed in some areas the AMIGA is not the computer to use. This is especially true in some graphics intensive applications. The highres and AA chip sets go a step towards improving the situation. However these chip sets are just a stop gap measure and have been outdated even before release to manufacturing. A new machine is required to bring Commodore back to the front edge of technology. This document outlines the details of a new AMIGA chip set which will again make the AMIGA the obvious choice.

2 DESIGN GOALS.

Many approaches can be taken to solve any given engineering problem. Each of the approaches will capitalize on some design attribute to produce a system with a set of strengths, and a set of weaknesses. This section outlines the set of goals used to guide the design of the new system. With this set of goals established, proper decisions can be made when there are choices to be made between design alternatives.

2.1 No Orphaned Software.

One of the primary reasons that the A1000 did not sell as well as first predicted was that it had no software base. The computer buyer saw a machine with great potential, but he had a job to get done and could not find the applications software he needed to do it. So instead, he bought brand X which had a vast software base of just the application he was looking for. Now, after seven years in the market, the AMIGA has accumulated a substantial base of software. It is essential that any future AMIGA system be capable of executing this software.

There has been much debate in the Commodore Engineering Dept about "levels of software compatibility". Some are of the opinion that software compatibility can be maintained across foreign hardware implementations by providing system software calls which hide the hardware differences. In many cases, this philosophy is true. However, there are also many cases where the applications programmer can not afford to go thru general purpose system calls because of the overhead involved. There will always be those applications where every available ounce of power is needed and can only be gotten by going directly to the hardware. Programs which follow this practice should not be condemned; this type of software typically demonstrates the true power of the machine. System level calls are provided as a convenience to the programmer. He can use them if he wants and have a greater chance of being compatible with future software and hardware, or he can bypass the calls and use the machine for what its worth now. It is a primary goal of the new chip set not to orphan software which goes directly to hardware. Almost all of the old RGA registers have been maintained in the new architecture and will function identically to the old chip set. A list of those registers which will not be supported in the new chip set is given in appendix D. For the most part, these registers govern the workings of the super and ultra highres modes designed into the highres chip set. There are also some subtle differences between the workings of old RGA modes and the same modes of the new chip set. These differences will be covered in later sections of this document.

This goal gives us a starting point for the definition of the new chipset. One can envision the new chipset as a superset of the old. The benefits of this are twofold: 1) much of the new chip set is already known and therefore the user learning curve is reduced. 2) Much of the hardware used in the old chip set can be used in the new designs. There will be less reinventing of the wheel which will result in shorter design cycles.

It would be easy to carry the idea of backwards compatible hardware to an extreme. There will be an overwhelming desire, as the designs progress, to make the designs orthogonal by extending new modes of operation back into the old modes. This desire will be magnified by

the fact that the required logic to make the extensions would be trivial because most of the difficult hardware will be in place to support the new mode. These extensions will not be implemented. The idea is to encourage programmers to use the new modes as provided, and discourage use of old mode hardware. For example, it is desirable to eliminate the old Color registers at the first available opportunity. The reason for this is hardware overhead. Only 13 bits of useful information is packed into the old color registers, and this information does not map nicely into the new color registers which contain 25 bits of useful information. The sooner we can get rid of the old registers, the sooner we can get rid of the additional multiplexers and address decoders. Taking this approach in the design of the new chips gives Commodore a path of migration. Eventually, we will not have to support old mode hardware. We will then have an easier time of designing fourth generation Amiga custom chips to provide even better and greater things.

This document assumes that the reader has a thorough knowledge of the old chip set and it's workings. None of the old modes will be covered except to point out minor deviations in operation. The original AMIGA Hardware Manual should be referenced as a first step to understanding the new AMIGA chip set.

2.2 Flexibility.

Another goal of the new chip set is to support a wide variety of applications. The concept of one machine for all applications provides inefficient and less than optimal solutions. The low end user will end up paying more than he wants and the high end user will not have all the performance he wants. The new chip set will be flexible in that it will support many different system configurations. The chip set can be configured in an inexpensive low end system to drive low to medium resolution graphics displays or it can be configured in a high end system to drive low to high resolution graphics displays.

The chip set will support the use of two different types of CHIP RAM. The user can buy cheap standard DRAM, or he can buy fast VRAM to improve custom chip performance by a factor of as much as 8. (See section 3, 'Performance Comparisons'.)

Finally, the custom chip-processor interface is asynchronous. The new chip set can be used with a processor running at any clock speed.

2.3 More, Bigger, Better.

The new AMIGA chip set will improve almost every facet of the old chip set.

2.3.1 Increased Performance. -

In order to increase most aspects of system performance, bus bandwidth has been increased. A factor of 2 increase results by increasing the width of the databus to 32 bits. Another factor of 2 increase results by implementing a burst mode cycle in the RAM access scheme. Finally, a factor of 1.14 increase comes about because the total burst cycle time is 7/8's as long as two single fetch cycles.

All systems which use the new chip set will have available 4.6 times more bus bandwidth than that which was available to the old AMIGA systems. Beyond this, even more bus bandwidth can be obtained. Standard system DRAM can be replaced with VRAM. Video DRAM allows most graphics fetches to be offloaded onto the VRAM serial port. CPU, blitter, and other busmaster bandwidth is tremendously improved when graphics fetches are removed from the chip data bus.

The following list summarizes the performance figures of the charts given in section 3, 'Performance Comparisons'.

1. Higher CPU bandwidth. In a system running a 640 x 200 x 2 bitplane display, CPU bandwidth will increase between 3 and 6 times the old Amiga CPU performance. CPU bandwidth will increase between 4 and 9 times when the system is running 4 bitplanes on a 640 x 400 x 2 display.

2. Higher Graphics bandwidth. The new system is capable of generating much higher graphics resolution than the old system. A lowend configuration using fast page mode DRAM can generate displays of up to 800 x 560 x 9 bitplanes. Using VRAM, the same configuration can generate 800 x 560 x 13 bitplane displays, or it can display HYBRID pixels at 800 x 560 to provide 24 bit per pixel, true color displays. A highend configuration can do more. Using Fast Page Mode RAM it can put up 1280 x 1024 x 5 bitplane displays. When VRAM is used, 8 bitplanes can be displayed at 1280 x 1024 and HYBRID pixels can be displayed on 1024 x 768 monitors.

3. Higher Blitter bandwidth. Display vertical scroll speed can be used as a measure of blitter performance. The new chip set can scroll 640 x 200 x 2 screens about 6 times faster than the old Amiga. It can scroll 640 x 200 x 4 and 640 x 400 x 2 screens about 9 times faster than the old Amiga.

4. Higher Blitter Performance with Multiple Blit Chips. The new Amiga chip set supports multiple blitter chips in a Blitter per RAM bank configuration. In such a configuration it is possible to operate all of the blitter chips in parallel, each operating on a portion of the data. Blitter performance is therefore increased by a factor equal to the number of blitter chips in the system. For example, in a two blitter chip system, the performance of the display scroll benchmark given above would increase from 6 to 9 times faster to 12 to 18 times faster.

2.3.2 Increased Modality. -

Where appropriate, old Amiga modes have been extended, and new modes have been added.

2.3.2.1 New Blitter Modes. -

The new blitter is software compatible with the old blitter. Each of the old blitter register addresses are recognized and operate as before. When used in old mode, the blitter starts as before when the BLTSIZE or BLTSIZH register is written and operates on 16 bit data fetched a word (16 bits) at a time.

New functionality has been designed into the blitter. These functions are invoked by writing the BLTFUNCX register. In its new modes, the programming interface to the blitter is simplified. The blitter operates using pixel addresses, and blit sizes (width and height) are specified in pixels. No masking, modulus, or shift information is required from the programmer. In this mode, the blitter fetches memory in 32 bit units, and optionally uses page-mode burst accesses to increase blitter performance.

Line draw modes of the new Blitter have been extended to include a new "clip-rect" mode. This mode is useful in the Amiga windowing operating system environment, and significantly improves system performance when windows are sized, moved, popped, etc.

Using new modes, the blitter can be programmed to operate on data in the traditional way, one bit at a time, or it can be programmed to operate on chunky data. Chunky data is organized as 2, 4, 8, or 16 bits of contiguous pixel data. This organization is more appropriate for certain kinds of imaging applications than the traditional bitplane organization.

In addition to the extended blitter modes, new modes have been added to support imaging applications. The blitter is capable of sorting a region of any of the chunky pixels and it can also tally the number of such pixels after a sort has been done. The blitter is also capable of performing arithmetic add operations on chunky pixels.

2.3.2.2 New Coprocessor Functionality. -

The old coprocessor is a tool used to control screen based events. It is deficient when it attempts to control events unrelated to screen timing. A new coprocessor could be added to alleviate this shortcoming, but would be wasteful of silicon as only one coprocessor can use the bus at any given time. Instead, the old coprocessor has been modified to incorporate interrupt processing capabilities. In addition to it's normal functions, the coprocessor can recognize interrupts from the blitter so that it may be used to conveniently restart another blit operation.

In addition to the new interrupt processing capability of the copressor, the coprocessor has been extended to to support operations on the new 32-bit registers as well as retaining compatibility with old 16-bit operations. In 32-bit mode, the coprocessor's MOVE instruction has been extended to be a MOVE MULTIPLE instruction. This instruction is useful when it is necessary to load blocks of consecutive registers such as when updating the color table.

2.3.2.3 Extended Sound Capabilities. -

The new chip set provides enhanced audio. The number of audio channels has been extended from the original four to eight channels. The audio circuits are capable of sustained audio sample rates better than 50Khz with an analog output of 16 bit resolution, thus giving better than CD quality sound. Other new audio modes permit sample rates to 110Khz, and effective period resolution to 4.375nsec resolution.

2.3.2.4 Two And Four Megabyte Floppy Disk Drive Support. -

The new chip set supports the use of the old style 1 Megabyte disk drives as well as newer 2 Meg and 4 Meg drives. Systems using high density floppy drives have the ability to read and write disks of lower density.

The floppy control circuits have also been extended to directly support IBM floppy formats, RLL encoding schemes, and audio CD formats. It is possible to directly read and write IBM MFM sector data without going thru the time consuming process of inserting clock bits into the data stream.

2.3.3 Extended Graphics. -

The new Amiga chip set supports all of the original graphics modes and offers more. The following list summarizes new graphics modes offered by the chip set.

1. Colors. The color palette of the new chip set is 256 entries long by 24+1 bits wide. The +1th bit is a transparency bit which has been included to support the genlock modes of the old chip set. The palette is accessed when the graphics chip is in bitplane mode to provide a choice of 256 colors out of 16 Meg.

2. 128 Bit Sprites. Sprites have been extended to 128 bits.

3. 16 BitPlanes. An additional 10 bitplanes have been added to the graphics chip. It is possible to enable two playfields of 8 bitplanes each.

4. Chunky Pixels. A variety of chunky pixel modes have been added. Two-bit Half Chunky pixels are 2 bits deep and are used to address the color palette. Four-bit Half Chunky pixels are 4 bits deep and are used to address the color palette. Half Chunky pixels are 8 bits deep and are used to address the color palette. Chunky pixels are 16 bits deep. These pixels bypass the color palette to drive the color guns directly; 5 bits are applied to red, 5 bits to green, and 5 bits to blue (the extra bit is used as a 'ZD' bit for genlock applications). Hybrid pixels are similar to Chunky pixels except they are 24 bits deep and they don't have an extra bit for ZD (ZD is forced inactive in this mode). Hybrid pixels give the user a choice of 16 million display colors from a palette of 16 million.

5. Packed Pixels. Packed pixels form a compressed screen image which when decompressed and displayed, allow a high degree of color resolution, but at a fraction of the storage requirements. This makes packed pixels ideal for use with animation applications. Two types of packed pixel formats are provided. PACKLUT pixels decompress to eight bit half chunky pixels which are used to address the color palette. PACKHY pixels decompress to 24 bit pixels which bypass the color palette and drive the display directly. PACKHY pixels have a storage requirement of 4 bits per pixel giving a data compression ratio of 6, and PACKLUT pixels have a storage requirement of 2 bits per pixel giving a data compression ratio of 12.

6. High Resolution Displays. As mentioned earlier, the new chip set can drive high resolution displays as well as the standard NTSC and PAL displays. In a low end configuration, the chip set can drive 640 x 400, 800 x 560, and 800 x 600 monitors at frame rates of 50hz to 72hz, or higher. In a high end configuration, the chip set can drive 640 x 400, 800 x 560, 1024 x 768, and 1280 x 1024 monitors, again at a frame rate of 50hz to 72hz or higher. The graphs of section 3, 'Performance Comparisons', outline the number of bitplanes which can be displayed at each resolution.

7. Screen Promotion. Systems which are driving high resolution monitors have the capability to promote the displays of lower resolution monitors. This allows the user to run software written specifically for lowres graphics systems on his highres system.

3 PERFORMANCE COMPARISONS.

This section outlines graphics and memory bandwidth capabilities for both the new low end and high end Amiga systems. A display chart is presented for each system which indicates the display modes achievable in the system. The display chart is followed by a chart which indicates the maximum CPU bandwidth available when certain graphics modes are enabled. These bandwidth figures assume that all CPU accesses are burst mode accesses. Since this scenario is highly unlikely, true maximum CPU bandwidth will be somewhere between the numbers given and slightly under half that amount. The number of single access (not burst mode) RAM cycles available can be calculated from these charts using the following formulas:

Low End System- AVAILABLE CYCLES = CPU BANDWIDTH / 9.1428572.
High End System- AVAILABLE CYCLES = CPU BANDWIDTH / 9.1428572.
Note: the 9.1428572 comes from 4 bytes per longword times a factor of 2.2857143 for burst mode speed (one burst mode cycle takes 7/8's the amount of time of two single access cycles).
Old Amiga System- AVAILABLE CYCLES = CPU BANDWIDTH / 2.
Note: the 2 comes from 2 bytes per word.

Finally, a chart is included which gives a measure of blitter performance. These numbers indicate for certain display configurations(one Blitter Chip systems), how many display screens can be blitted each second(scroll speed). The numbers given can be multiplied by the number of Blitter Chips optionally included in the system to show what blitter performance would be in a Blitter expanded system. Like the CPU bandwidth charts, these charts assume that the blitter is getting all available RAM cycles after the display is satisfied.

Low End System, Fast Page Mode DRAM

Graphics Capabilities

Display Res.	Bitplane	HalfChunky	Chunky	Hybrid	PackLUT	PackHY
640/200	16	YES	YES	YES	YES	YES
704/200	16	YES	YES	YES	YES	YES
640/400	12	YES	YES	NO	YES	YES
800/560	10	YES	NO	NO	YES	NO
OLD AMIGA						
640/200	4	NO	NO	NO	NO	NO
704/200	4	NO	NO	NO	NO	NO
640/400	2	NO	NO	NO	NO	NO

CPU Bandwidth (Megabytes/sec)

Display	640x200	640x400	800x560
2 BITPLANE	31.30	29.88	27.90
4 BITPLANE	29.87	27.03	23.07
8 BITPLANE	27.02	21.33	13.41
16 BITPLANE	21.31	--	--
HALF CHUNKY	28.22	23.72	17.05
CHUNKY	23.82	14.94	--
HYBRID	19.19	--	--
OLD AMIGA			
2 BITPLANE	5.24	3.32	-
4 BITPLANE	3.31	-	-

Display Screens Blitted per Second(Scroll speed)

Display	640x200	640x400	800x560
2 BITPLANE	489.06	233.42	124.54
4 BITPLANE	233.37	105.58	51.49
8 BITPLANE	105.53	41.65	14.96
16 BITPLANE	41.61	--	--
HALF CHUNKY	110.22	46.33	19.03
CHUNKY	46.52	14.59	--
HYBRID	24.99	--	--
OLD AMIGA			
2 BITPLANE	81.80	25.93	-
4 BITPLANE	25.87	-	-

Low End System, Video DRAM

Graphics Capabilities

Display Res.	Bitplane	HalfChunky	Chunky	Hybrid	PackLUT	PackHY
640/200	16	YES	YES	YES	YES	YES
704/200	16	YES	YES	YES	YES	YES
640/400	16	YES	YES	YES	YES	YES
800/560	14	YES	YES	YES	YES	YES
OLD AMIGA						
640/200	4	NO	NO	NO	NO	NO
704/200	4	NO	NO	NO	NO	NO
640/400	2	NO	NO	NO	NO	NO

CPU Bandwidth (Megabytes/sec)

Display	640x200	640x400	800x560
2 BITPLANE	32.27	31.82	31.44
4 BITPLANE	31.81	30.90	30.15
8 BITPLANE	30.90	29.08	27.57
16 BITPLANE	29.07	25.43	--
HALF CHUNKY	32.61	32.50	32.41
CHUNKY	32.61	32.50	32.41
HYBRID	32.38	32.04	31.76
OLD AMIGA			
2 BITPLANE	5.24	3.32	-
4 BITPLANE	3.31	-	-

Display Screens Blitted per Second(Scroll speed)

Display	640x200	640x400	800x560
2 BITPLANE	504.23	248.56	140.35
4 BITPLANE	248.54	120.72	67.30
8 BITPLANE	120.70	56.80	30.77
16 BITPLANE	56.78	24.84	--
HALF CHUNKY	127.39	63.48	36.17
CHUNKY	63.70	31.74	18.08
HYBRID	42.17	20.86	11.82
OLD AMIGA			
2 BITPLANE	81.80	25.93	-
4 BITPLANE	25.87	-	-

High End System, Fast Page Mode DRAM

Graphics Capabilities

Display Res.	Bitplane	HalfChunky	Chunky	Hybrid	PackLUT	PackHY
640/200	16	YES	YES	YES	YES	YES
704/200	16	YES	YES	YES	YES	YES
640/400	16	YES	YES	YES	YES	YES
800/560	13	YES	YES	YES	YES	YES
1024/768	8	YES	YES	NO	YES	YES
1280/1024	5	YES	NO	NO	YES	NO
OLD AMIGA						
640/200	4	NO	NO	NO	NO	NO
704/200	4	NO	NO	NO	NO	NO
640/400	2	NO	NO	NO	NO	NO

CPU Bandwidth (Megabytes/sec)

Display	640x200	640x400	800x560	1024x768	1280x1024
2 BITPLANE	31.89	31.05	30.03	28.18	25.27
4 BITPLANE	31.05	29.37	27.33	23.63	17.82
8 BITPLANE	29.37	26.02	21.93	14.53	--
16 BITPLANE	26.01	19.32	--	--	--
HALF CHUNKY	30.40	28.09	24.84	18.76	9.01
CHUNKY	28.19	23.67	17.25	5.21	--
HYBRID	25.76	18.81	9.06	--	--
OLD AMIGA					
2 BITPLANE	5.24	3.32	-	-	-
4 BITPLANE	3.32	-	-	-	-

Display Screens Blitted per Second(Scroll speed)

Display	640x200	640x400	800x560	1024x768	1280x1024
2 BITPLANE	498.24	242.59	134.05	71.66	38.56
4 BITPLANE	242.56	114.75	61.00	30.05	13.59
8 BITPLANE	114.72	50.83	24.48	9.24	--
16 BITPLANE	50.80	18.87	--	--	--
HALF CHUNKY	118.77	54.86	27.72	11.93	3.44
CHUNKY	55.06	23.12	9.63	3.31	--
HYBRID	33.54	12.25	3.37	--	--
OLD AMIGA					
2 BITPLANE	81.83	25.96	-	-	-
4 BITPLANE	25.90	-	-	-	-

High End System, Video DRAM

Graphics Capabilities

Display Res.	Bitplane	HalfChunky	Chunky	Hybrid	PackLUT	PackHY
640/200	16	YES	YES	YES	YES	YES
704/200	16	YES	YES	YES	YES	YES
640/400	16	YES	YES	YES	YES	YES
800/560	16	YES	YES	YES	YES	YES
1024/768	11	YES	YES	YES	YES	YES
1280/1024	8	YES	YES	NO	YES	YES
OLD AMIGA						
640/200	4	NO	NO	NO	NO	NO
704/200	4	NO	NO	NO	NO	NO
640/400	2	NO	NO	NO	NO	NO

CPU Bandwidth (Megabytes/sec)

Display	640x200	640x400	800x560	1024x768	1280x1024
2 BITPLANE	32.28	31.83	31.47	30.93	30.39
4 BITPLANE	31.83	30.94	30.21	29.14	28.06
8 BITPLANE	30.93	29.15	27.69	25.56	23.39
16 BITPLANE	29.14	25.57	22.66	18.39	14.06
HALF CHUNKY	32.47	32.22	31.99	31.64	31.25
CHUNKY	32.44	32.15	31.87	31.42	30.90
HYBRID	31.96	31.20	30.53	29.47	--
OLD AMIGA					
2 BITPLANE	5.24	3.32	-	-	-
4 BITPLANE	3.32	-	-	-	-

Display Screens Blitted per Second(Scroll speed)

Display	640x200	640x400	800x560	1024x768	1280x1024
2 BITPLANE	504.36	248.69	140.49	78.67	46.38
4 BITPLANE	248.68	120.85	67.44	37.06	21.41
8 BITPLANE	120.84	56.93	30.91	16.25	8.92
16 BITPLANE	56.92	24.97	12.65	5.85	2.68
HALF CHUNKY	126.85	62.93	35.71	20.12	11.92
CHUNKY	63.36	31.40	17.79	9.99	5.89
HYBRID	41.62	20.32	11.36	6.25	--
OLD AMIGA					
2 BITPLANE	81.83	25.96	-	-	-
4 BITPLANE	25.90	-	-	-	-

4 THE SYSTEM.

This section discusses system level hardware issues. System clocks, the new DMA scheme, the low end chip configuration, the high end chip configuration, RAM support, and bus cycles are among the topics covered.

4.1 System Clocks.

For reasons covered later in this section, it is not possible to retain the old clock scheme to drive the new chip set. The new chip set will use two fundamental clock frequencies during operation. This section discusses the motivation of the new clock scheme.

4.1.1 LowRes/HighRes Pixels. -

The original AMIGA chip set introduced the concept of highres and lowres pixels. The chipset was intended to be used in a game machine connected to NTSC or PAL television sets. Lowres pixels are used to drive televisions and have twice the frequency of the TV color burst signal. Analysis of standard DRAM timings show a good match to the color burst period. Therefore, the DRAM cycle was made half the frequency of lowres pixel frequency and formed the basis for system timing. The designers of the original chip set realized that providing only 320 pixels per line would severely limit the machine's use. Highres pixels were an easy extension made to answer the needs of the serious computer user. We've come to accept highres pixels as the defacto standard. The A500, A1000, and A2000 are 640x200 machines.

The new AMIGA chip set follows the original scheme. All systems have the ability to display both lowres and highres pixels where highres displays have twice the resolution of lowres displays. For example, VGA+ systems will be able to generate highres 800x560 displays as well as lowres 400x560 pixel displays. The new chip set does not directly support the ECS or AA chip set super highres and ultra highres display modes. System software must make the changes necessary (as outlined in section 5.1, 'Hard Monitor Control Stops') to drive displays of these resolutions.

Most events in the old chipset were timed by counting lowres or pairs of lowres pixels. By keeping the relationship of system events to pixel timing, it is possible to retain software compatibility and the old software model yet change pixel frequencies to accommodate highres displays. Some system events did not count pixels but rather counted a clock of a known frequency. All of the disk, blitter, audio, uart, and pot functions fall into this category. The disk and blitter functions use a 28Mhz 'CLK28' clock for timing purposes. The audio, uart, and pot functions use a 14Mhz 'BUSCLK' clock. The frequency of both CLK28 and BUSCLK remains constant in each of the system configurations. As the original CCK system clock can be derived from both CLK28 and BUSCLK, software compatibility is retained in these functions.

4.1.2 Pixel Timing No Longer Linked To RAM Timing. -

Because of the necessary timing differences between monitors of various resolutions, the relationship between DRAM timing and pixel timing can no longer be maintained. Chip bus cycles will run asynchronous to pixel based events. There will be a system bus clock, and a system pixel clock. Where necessary, synchronizing circuits will be used to interface bus clock events to pixel clock events. In many cases, synchronizing circuits are not necessary because the registers are written well in advance of or read long after the pixel event the register services, thus giving plenty of time for the register to stabilize. In most cases when synchronizing circuits are necessary, they will be invisible to applications software.

4.1.3 Horizontal Line Counter. -

The horizontal counter of the AAA chip set functions somewhat differently than past revisions of the Amiga chip set. Specifically, it incorporates two additional low order counter bits. For example, when the graphics mode selected is highres NTSC (640x200), the old chip counted CCK(3.579Mhz) clock pulses. In the same graphics mode using the new chip set, 14.318Mhz clock pulses are counted. Inclusion of these bits permits the specification of events to a finer time resolution in several of the new and extended display modes. These bits will not affect the functionality of existing software. When old RGA horizontal compare registers are written, data is loaded to match the most significant part of the counter and the new low order bits of the counter are effectively ignored.

4.2 Low End System.

The new Amiga chip set has been designed for flexibility. A low end configuration of the chip set can be designed to provide an A500 type machine. The following block diagram shows the new chip set in a low end configuration. The primary components are:

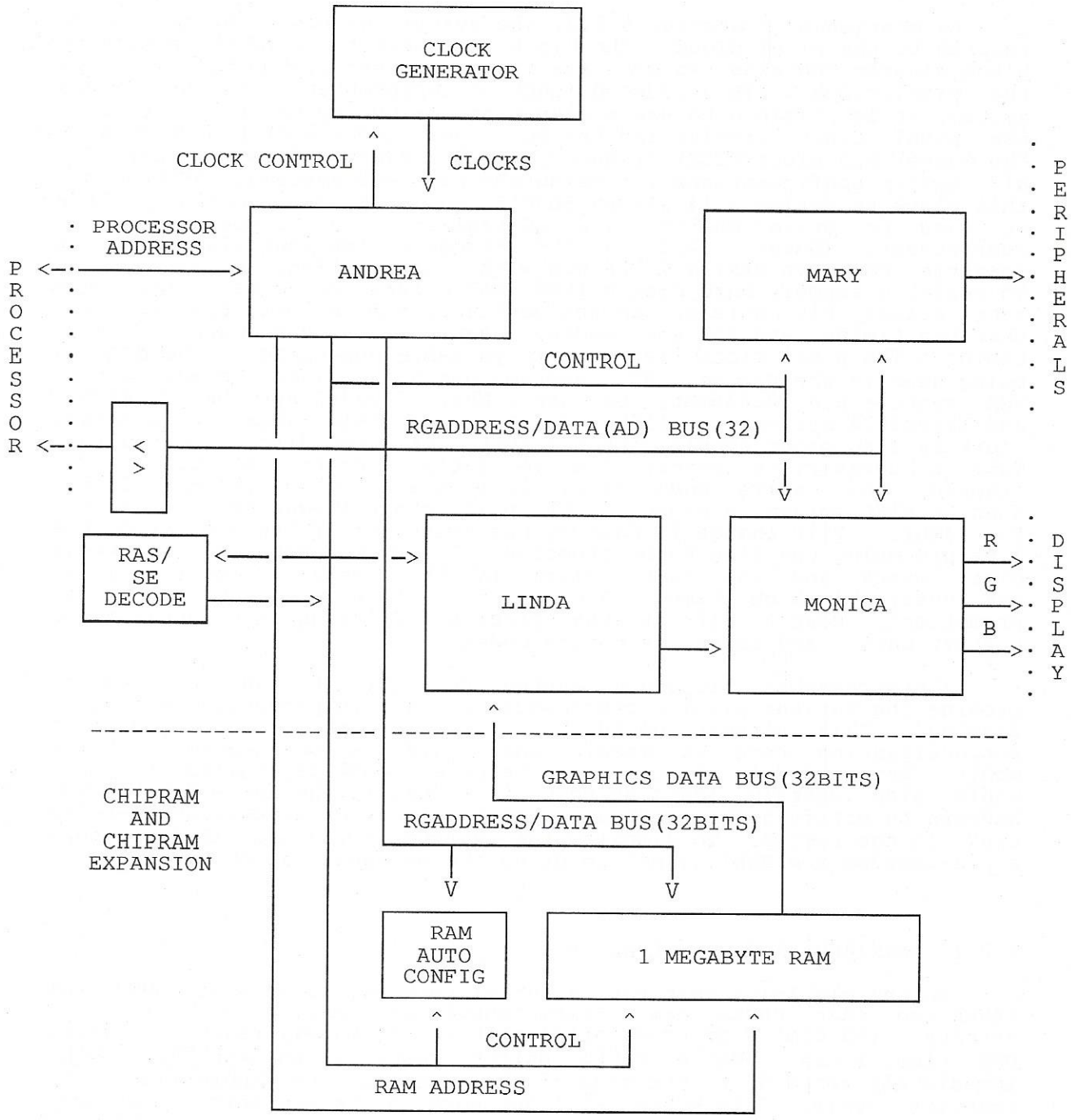
ANDREA - Replaces the old Agnus chip. Does address generation for the chip DMA channels, controls DRAM accesses, and provides the blitter and coprocessor functionality.

MARY - Replaces the old PAULA chip. This chip is responsible for floppy disk control, audio, uart, interrupt generation, pot, and mouse control.

LINDA - Provides a line data buffer not found in the old chip set. This chip buffers display data on a line by line basis to allow higher effective display bandwidth for higher resolution displays and at the same time permitting a degree of compatibility with old software.

MONICA - Replaces the old DENISE chip. This chip processes all display data including Sprite information.

Other elements shown on the block diagram are a clock generation block which provides the memory and pixel clocks, decoder blocks which decode RAS and SE RAM control signals, a processor databus interface/FIFO control chip, RAM auto configuration logic, and chip RAM. Appendix C shows a detailed block diagram of the low end system.



Block Diagram of Low End System Configuration

4.2.1 Bus And Pixel Clocks. -

As mentioned in section 4.1.2, the system bus clock is no longer related to the pixel clock. The Clock Generator block of the preceding block diagram contains two crystals to provide the system bus clock and the pixel clock. (In a reduced function implementation of the low end system, it is possible to use a single frequency crystal to drive both the pixel clock circuits and the bus clock circuits of the chip set.) The master bus clock(MCLK) frequency is 57.2727Mhz and is constant for all system configurations including the high end systems. ANDREA uses this clock to derive a 14.318Mhz BUSCLK, and a 28.636Mhz CLK28. CLK28 is used to drive MARY's disk controller, the blitter, and the coprocessor. (Note: this is the standard configuration. It is possible to drive MARY's CLK28 pin with a higher frequency clock so as to reliably support hard disk drives, DAT drives, and other high data rate serial bit stream storage devices.) BUSCLK forms the basis for chip bus timing, and for the audio, uart, pots, and mouse circuit timing. The pixel clock frequency is variable depending on the monitor being used in the system. NTSC systems use a 14.318Mhz pixel clock, VGA systems use 28.636Mhz, VGA+ use 36Mhz, 1024x768 systems use 64Mhz, and 1280x1024 systems use 110Mhz (all at 60hz field rates). The pixel clock is also programmable within a system of given display resolution. This is necessary to permit low resolution screen promotion. For example, the native mode pixel clock of a 1024x768 system is 64Mhz. When an NTSC screen is promoted, the pixel clock frequency is changed to 44Mhz. This change is made by the processor or coprocessor on the line preceding the line to be promoted. The switch between the 64Mhz pixel clock and the 44Mhz clock is then made automatically at horizontal blanking time. Section 5.5, 'Low Resolution Screen Promotion', covers all of the pixel clock frequencies necessary to support native and screen promotion modes.

A programmable frequency synthesizer circuit can be used to provide the various pixel clocks needed for screen promotion on a given display. This circuit would be incorporated into a monitor personalization card or module and would be changed to match the monitor being used in the system. The pixel clock personalization card would also contain logic which will respond to the new MONITORID RGA address to notify system software what type monitor is currently being used in the system. To simplify the external circuitry, MARY provides a pre-decoded MONIDEN* signal to drive the personalization card.

4.2.2 MARY/ANDREA Serial Link. -

In the old Amiga chip set, a serial port exists between AGNUS and PAULA so that PAULA can inform AGNUS of DMA channels which need service. The timing of this serial port is intimately related to fixed DMA time slots. While PAULA shifts requests to ANDREA, ANDREA immediately responds to the request by running the appropriate DMA transfer cycle. This scheme will not work in the new chip set because it cannot be guaranteed that any given cycle will be available to satisfy a request. The new chip set incorporates an on demand prioritized DMA request scheme as described in section 4.6, 'On Demand DMA'. The serial link between ANDREA and MARY has been changed accordingly. Rather than waiting until horizontal blank time to inform ANDREA of all DMA requests accumulated during the previous line, MARY sends requests to ANDREA as they are generated. The requests are accumulated and sent to ANDREA as follows.

After system reset, the serial link between MARY and ANDREA is constantly running and sending data, even when there are no dma requests being made. In Mary, audio requests go into two 8 bit shift registers shifted when the lower 2 bits of MARY's audio counter equals 3 (at a 3.5Mhz rate). The requests are picked off various taps of the shift register so they go out of the link at the correct bit position in the transfer. Disk requests go into a two deep fifo loaded by an internal MARY signal, dskdr. The output of this fifo gets multiplexed onto DMAL at the appropriate bit positions (see following bit position chart).

Andrea is synchronized to the serial link by two special mark bits. There are no dma requests at power up time during reset. A 5 bit counter in Andrea counts to 31 and stops until two '1s' are present separated by 3 bits (bits in between are not checked). Since this is a synchronous system, the serial channel should never lose sync after the first null transfer following reset. If the channel should lose sync, it will recover in at most two complete transfer cycles which contain no dma requests. After sync, the counter in Andrea will be a few ticks behind the one in Mary. A 4 bit shift register takes in the data from DMAL, and every 4 bits a nibble of data is latched, either into the disk request fifo or into four of the audio request storage registers. The disk fifo is only loaded when the request number is non zero. Acknowledgement from inside Andrea that the cycle has occurred unloads the disk fifo or resets the audio request flip-flops appropriately. The following list shows the relative position of each request bit in the serial data stream.

Bit Position	Meaning
0	aud0dsr, Audio channel 0 (re)start request
1	aud0dr, Audio channel 0 empty
2	aud1dsr, Audio channel 1 (re)start request
3	aud1dr, Audio channel 1 empty
4	aud2dsr, Audio channel 2 (re)start request
5	aud2dr, Audio channel 2 empty
6	aud3dsr, Audio channel 3 (re)start request
7	aud3dr, Audio channel 3 empty
8	dskreq(2), Disk Request Control Bit 2
9	dskreq(1), Disk Request Control Bit 1
10	dskreq(0), Disk Request Control Bit 0
11	0, this bit always zero
12	dskreq(2), Disk Request Control Bit 2
13	dskreq(1), Disk Request Control Bit 1
14	dskreq(0), Disk Request Control Bit 0
15	0, this bit always zero
16	aud4dsr, Audio channel 4 (re)start request
17	aud4dr, Audio channel 4 empty
18	aud5dsr, Audio channel 5 (re)start request
19	aud5dr, Audio channel 5 empty
20	aud6dsr, Audio channel 6 (re)start request
21	aud6dr, Audio channel 6 empty
22	aud7dsr, Audio channel 7 (re)start request
23	aud7dr, Audio channel 7 empty
24	dskreq(2), Disk Request Control Bit 2
25	dskreq(1), Disk Request Control Bit 1
26	dskreq(0), Disk Request Control Bit 0
27	1, sync, this bit always one
28	dskreq(2), Disk Request Control Bit 2
29	dskreq(1), Disk Request Control Bit 1
30	dskreq(0), Disk Request Control Bit 0
31	1, sync, this bit always one

ANDREA DMA operation for Audio Start Requests(dsr) and Empty Requests(dr)

AUDnDSR	AUDnDR	Action
0	0	no request
0	1	AUDnDAT=*AUDnPTR++;
1	0	error
1	1	AUDnDAT=*AUDnPTR; AUDnPTR=AUDnBKUP;

ANDREA DMA operation for Disk Requests

dskreq(2)	dskreq(1)	dskreq(0)	Action
0	0	0	no DMA request
0	0	1	DSKDAT= *DSKHDPT ; DSKHDPT++;
0	1	0	DSKDAT= *(DSKHDPT=DSKBKPT) ; DSKHDPT++;
0	1	1	DSKDAT= *(DSKBKPT=DSKHDPT) ; DSKHDPT++;
1	0	0	*(DSKPT=DSKBKPT)=DSKDATR ; DSKPT++;
1	0	1	DSKDAT= *DSKPT ; DSKPT++;
1	1	0	DSKDAT= *(DSKPT=DSKBKPT) ; DSKPT++;
1	1	1	*DSKPT= DSKDATR ; DSKPT++;

Maximum Latency of Audio DMA request channels.
(Request generated in MARY to request received by Andrea; does not account for additional latency which may be incurred while ANDREA completes the current cycle in progress.)

channel	delay(14Mhz ticks)
0	35
1	31
2	31
3	27
4	35
5	31
6	31
7	27

Link Delay (request in Mary to request in Andrea)

	BUSCLK cycles	usec
disk min	9	0.6
max	20	1.4
audio min	27	1.9
max	35	2.5

4.2.3 ANDREA/LINDA Serial Link. -

Linda requires certain bitplane control register (BPLCON) information in order to perform all of her functions. Linda pin limitations do not permit access to the AD bus such that Linda could latch the BPLCON register data herself. Andrea pin limitations do not permit the necessary control information to be transferred directly to Linda via dedicated signal lines. Since most of the necessary data can only change on a line by line basis, a serial link between the two chips has been implemented to transfer the slow changing data at the beginning of each scan line. The information transferred on the serial link is: high-resolution or low-resolution (HI/LO RES), number of bitplanes used (BPU4-0), odd playfield display mode (OBPLMODE2-0, called T2-0 (for type) on the link), Switch, Direction, Line Repeat (called RP1-0 on the link), and PF1H0, PF2H0 (playfield's 1 and 2 scroll lsb). The data sent to LINDA is the same data ANDREA uses to make the current lines graphics fetch. In otherwords, the register is accessed after the TransferValues latch.

LINDA uses HI/LO when transferring information to MONICA. When HI/LO indicates low resolution, data is sent to MONICA at half the frequency that it is send when in High res mode.

BPU (bitplanes used) tells LINDA how many bitplanes of data have been buffered. LINDA uses this when the graphics mode is bitplane to help time exactly when data is to be sent to MONICA. BPU also has significance when it contains a zero. BPU zero indicates that no data is to be sent to MONICA this scan line. The graphics type does not have to be bitplane for this to be the case. The BPU field is forced all zeros during scan lines which make up vertical blanking.

T2-0 is decoded by LINDA to determine the graphics data type as follows:

T2-0	Graphics Type.
000	Bitplane mode.
001	HYBRID mode.
010	CHUNKY mode.
011	PACKLUT mode.
100	4-Bit Half Chunky.
101	HALFCHUNKY mode.
110	2-Bit Half Chunky.
111	PACKHY mode.

LINDA needs to know the graphics type when receiving data fetched from memory as well as when accessing the buffered data for presentation to MONICA.

SWITCH is a signal LINDA uses to determine that she should switch RAM banks, and begin filling new line data into the buffer she just finished sending to MONICA. SWITCH is normally set each scan line. It is not set on certain scan lines when the graphics system is in screen promotion mode. When this is the case, LINDA continues to fill the RAM bank she was filling last scan line, and she resends the same buffer to MONICA she sent last scan line. Stated another way, SWITCH is asserted each scan line that ANDREA's "line advance counter" rolls over to 0 (see section 5.5.2, 'Hardware Behind Screen Promotion').

DIR is the direction signal. When this bit is set, graphics data is captured from the VDATA bus and presented to the GDATA bus. (Normally, graphics data is captured from the GDATA bus and presented to the VDATA bus.) This signal indicates the graphics subsystem is in framegrabber mode.

RP indicates how many scan lines each buffer of graphics data is sent to MONICA. LINDA only uses this information when the graphics mode is one of the packed pixel types. The following table indicates how many times each line is sent to MONICA when the display mode is a packed pixel type.

RP1,0	ACTION
00	Each line is repeated 4 times.
01	Each line is repeated 3 times.
10	Each line is repeated 2 times.
11	Each line is displayed once.

PF1H0, PF2H0 are the least significant bits of playfield's 1 and 2 graphics scroll control field. LINDA uses these bits to determine which graphics bits, even or odd, to extract from the incoming data stream in a dual (high-end) system. When these bits are zero, the even LINDA chip extracts even pixels, and the odd LINDA chip extracts odd pixels. When these bits are one, the even LINDA chip extracts odd pixels, and the odd LINDA chip extracts even pixels.

Serial data is communicated to LINDA via the TDATA signal, and clocked into LINDA with the TCLK signal. Appendix A shows a timing diagram which depicts the transfer, and the following list shows the relative position of each data bit in the serial data stream.

Bit Position	Meaning	
0	HI/LO RES.	HIRES of BPLCON0.
1	BPU0.	BPU0 of BPLCON0.
2	BPU1.	BPU1 of BPLCON0.
3	BPU2.	BPU2 of BPLCON0.
4	BPU3.	BPU3 of BPLCON0.
5	BPU4.	BPU4 of BPLCON0.
6	T0.	OBPLMODE0 of BPLCON3.
7	T1.	OBPLMODE1 of BPLCON3.
8	T2.	OBPLMODE2 of BPLCON3.
9	SWITCH.	Switch RAM banks. Generated from TransferValues.
10	DIR.	DIR of BPLCON3. (Framegrabber mode.)
11	RP0.	LINERPT0 of BPLCON3.
12	RP1.	LINERPT1 of BPLCON3.
13	PF1H0.	PF1H0 of BPLCON1.
14	PF2H0.	PF2H0 of BPLCON1.

4.2.4 GDATA Bus. -

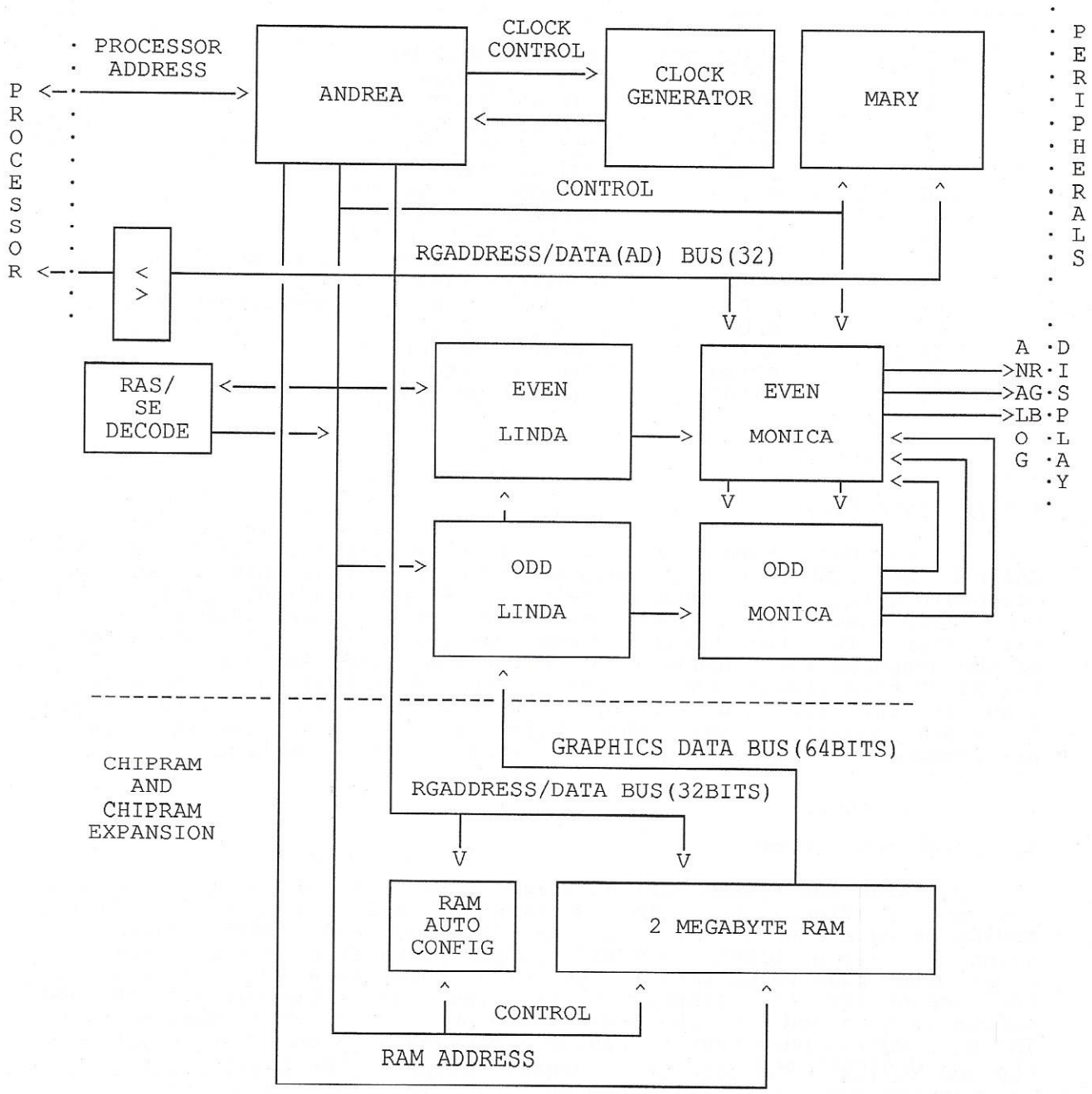
The low end system block diagram shows a graphics data bus between CHIPRAM and LINDA, the line data buffer chip. It is this bus which is responsible for the tremendous savings of AD bus bandwidth indicated in the Performance Comparisons charts of section 3 when VRAM is used in the system. This bus is tied to the VRAM serial port and removes most of the graphics fetch bandwidth from the RGAddress/Data (AD) bus. This bus is 32 bits wide in the low end system. When Fast Page Mode DRAM is used in the low end system, this bus is tied to the AD bus via tri-state buffer chips. When this is the case, overall system performance is somewhat degraded by graphics fetch overhead.

4.3 High End System.

The high end system configuration is essentially the same as the low end configuration. The differences are 1) a pair of LINDA and MONICA chips are used instead of one LINDA and one MONICA chip. By using a pair of LINDA and MONICA chips, pairs of pixels are processed in parallel thus permitting the use of a conventional CMOS process in the design of the graphics chips. The digital RGB bus from the odd MONICA chip is fed into the digital RGB bus of the even MONICA chip. The even MONICA chip then multiplexes between it's onboard RGB data and the odd MONICA's RGB data on alternate halves of the pixel clock cycle to provide composite analog RGB out. (It is also possible to feed the digital RGB data from both MONICA chips to external circuits such as some sort of Graphics Postprocessor element for subsequent multiplexing into a composite video stream.) 2) The minimum amount of RAM used in a high end configuration is 2 megabytes and RAM expansion occurs in 2 megabyte increments. More RAM as well as more RAM bandwidth is required to hold and fetch the bitplanes of a high resolution display.

The high end system is capable of generating all the displays achievable in the low end system: 640x200, 640x400, and 800x560. It can also generate displays for 1024x768 and 1280x1024 monitors. The block diagram on the following page shows the high end system

configuration.



Block Diagram of High End System Configuration

4.3.1 Dual LINDA, MONICA, And RAM. -

Dual LINDA, MONICA, and RAM chips are required to accommodate the high speeds associated with high resolution graphics displays. Current CMOS technology limitations prohibit the design of a single chip capable of the high frequencies involved. A pair of LINDA/MONICA chips are operated in parallel and thereby cut the effective operation frequency in half. The speed requirements of high resolution graphics display generation is met without taxing available technology. Section 6.1, 'Pixel Clocks', covers the design considerations necessary to make the single chip low end system identical from a software point of view to the dual chip high end system.

4.3.2 Half Speed Pixel Clocks. -

Preceding sections indicate that the horizontal counters operate at full pixel frequency. This would mean that the highend 1280x1K system operates at a 110Mhz pixel clock rate. Slower clocks are desirable from both a technological point of view as well as an FCC point of view. The highend system has been designed to use a pixel clock = pixel freq / 2. For example, a high end system driving an NTSC monitor will use a 7.159Mhz pixel clock whereas the lowend NTSC system will use a 14.318Mhz pixel clock. To account for this halving of the clock frequency and to provide a consistent software model between low and high end systems, the least significant horizontal counter bit output will be disabled and replaced with the pixel clock. The next most significant counter bit carry input will be driven by the pixel clock. This will make the counter twice as fast as its lowend counterpart, and make up the difference in clock frequencies. ANDREA has an input DUAL/SINGLE* which tells her when to make the change. Other circuits affected by the half speed pixel clock will be covered in subsequent sections of this document.

4.3.3 Graphics Postprocessors In The High End System. -

It may be desirable to incorporate some form of post processing graphics element in future high end systems. This capability is supported by both the even and odd chips to drive it's digital RGB data out rather than using the normal mode of operation where-by the even chip receives the odd chip's digital RGB information on the digital RGB bus. When this mode of operation is invoked, the post processing element assumes the responsibility for serializing the even/odd pixels into a composite pixel stream, and for providing the required DAC functions. All AAA supported graphics modes are available in this mode, including HAM mode. The postprocessor need only be concerned with the afore-mentioned serialization and DAC functions, and with whatever function it has been installed to implement.

The postprocessor receives even and odd pixels in digital form from the even and odd MONICA chips, serializes the pixels, performs whatever additional operations are required on the digital data, and then converts the three 8 bit digital colors of each pixel to analog form for delivery to a monitor. A pair of pixels is latched into the postprocessor with each rising edge of PCLK. The PIXMUX signal provided by the odd MONICA chip defines which pixel to convert and display at any given time. The pixel coming from the odd MONICA chip is converted and displayed first. It will be necessary to pipeline digital color information two clock cycles before display in order to

match sync signal timing. If this timing is not possible, it will be necessary to intercept and modify sync signal timing.

The graphics chips are faced with fast input to output propagation delay specs when the chips are programmed to operate at maximum resolution. Data is output from LINDA on a rising edge of PCLK and latched into MONICA on the next rising edge of PCLK. Similarily, data is output from MONICA on a rising edge of PCLK, and latched into the graphics postprocessor on the next rising clock edge. This means a clock in to data out delay of 15ns to provide 3ns setup time to the postprocessor element.

4.4 Processor Interface.

The new chip set has been designed to interface to one of Motorola's 32 bit microprocessors. It may be possible to use the chip set in an ultra low end system which uses a 68000 microprocessor. In such a system, external logic would be required to interface the 16 bit processor bus to the 32 bit chip bus. RAM width in such a system would still be 32 bits to satisfy chip access requirements.

4.4.1 Asynchronous Design. -

It is desirable to have the ability to permit use of an arbitrary frequency processor clock. With this ability, Commodore can provide a family of machines to give a choice of processor performance to the user. Perhaps a low cost 16Mhz machine can be designed which would allow a user, at his discretion, to upgrade to a 25Mhz, or 33Mhz processor.

Attempting to support an arbitrary processor bus frequency with the chip set is unrealistic. To accommodate arbitrary processor frequencies, the processor interface to the chip set has been designed as an asynchronous interface. Appendix A contains bus timing diagrams. Two of these diagrams show processor accesses of CHIPRAM: one diagram shows a single cycle access, and another shows a 68030 burst mode access. The relative timing shown in these diagrams is fixed regardless of processor frequency. External hardware must be designed to interface the processor bus to the timings shown in appendix A.

4.4.2 Smart Processor Databus Interface Controller. -

The asynchronous nature of the processor interface tends to prevent the use of 68030 synchronous burst mode cycles. Since this cycle has the potential of doubling processor bus bandwidth, some means of synchronizing the processor bus to the chip bus must be incorporated. The following scheme outlines one method of achieving this goal, and could be implemented as a gate array. (Note: the systems group will be responsible for assuring that the processor is properly interfaced to the new chip set. ANDREA is responsible for synchronizing the AS* signal from the processor to the BUSCLK.)

A bidirectional FIFO chip can be implemented to interface the processor to the new custom chips. This chip decouples processor accesses from chip processing.

In general, when the processor executes a write cycle to the chip address space, the FIFO intercepts and possibly caches the data being written at the processor speed, then outputs this data to the chip RAM at the BUSCLK rate. When such a cache function is designed into a system, the FIFO chip must not be permitted to cache certain of the chip registers. In particular, the FIFO must not cache any of the DMAEN, INTENA or INTREQ registers. Instead, the FIFO chip should either wait state the processor until the chip bus has been granted to the processor and the transaction has completed, or it should issue a retry sequence to the processor (see below). As an example of why certain registers are not cachable, consider the following example: The processor is attempting to modify the INTREQ register to clear out an interrupt request bit for some function it has just finished servicing. If the new INTREQ register data were cached by the FIFO chip and the processor allowed to continue, it may be possible that the FIFO chip will not gain access to the chip bus for an extended period of time while higher priority DMA channels are serviced. When this is the case, the processors IPL signals will continue to reflect a pending interrupt, as the clearing write to the INTREQ register has been postponed. As soon as the processor subsequently executes a return from interrupt instruction, the interrupt routine will be again be re-entered, but this time a system error has occurred. It may be better to only permit caches to chip RAM only, and avoid caches to register data.

Processor reads from the chip bus are simpler than writes. Normally, the processor will be wait stated (unless running at 14Mhz or less, and the chip bus is immediately available) while the FIFO accepts chip data at the chip bus speed and then provides this data to the processor at the processor speed.

Both read and write cycles begin when the processor makes an access to chip address space. The FIFO chip monitors the processor address lines and the processor's ECS and OCS signals to determine this. When these control signals indicate the processor is about to begin a chip bus access, the FIFO chip asserts the NEED* signal and possibly asserts the BURSTRQ* signal to ANDREA to indicate a cycle request and define if the request is a single cycle request or burst cycle request (BURSTRQ* = 0, processor requesting a burst cycle. BURSTRQ* = 1, processor requesting a single cycle.). The falling edge of AS* qualifies the NEED* and BURSTRQ* signals and causes processor address, size, and read/write signals to be latched into ANDREA. Therefore, these signals must be valid at the time that AS* for the cycle falls. The same time ANDREA latches address information, she also latches the REG/RAM* address decode signal from GARY(maybe the FIFO chip will replace GARY). During the next half cycle of BUSCLK (BUSCLK=0), ANDREA will arbitrate the next RAM access cycle. If there are no higher priority DMA requests at that time, the processor will be awarded the next bus cycle. If ANDREA discovers that there is an outstanding higher priority DMA request, then ANDREA will assert the GOAWAY* signal to the FIFO which will cause the FIFO to generate a processor retry sequence. When this happens, latched address information is invalidated in ANDREA, and the processor must retry the access again to be serviced. When the processor makes a request that can be served on the following cycle, the processor is wait stated (unless it is running at a slow enough speed). The FIFO chip will keep the request line asserted until ANDREA responds with a BG*(bus grant). The transfer is complete a fixed amount of time after that as indicated by the timing diagrams of Appendix A.

Section 4.6.3, 'DMA Priorities' shows the relative priority levels of all the chip DMA channels. Note that processor priority is either lowest or second lowest depending the Blitter Nasty and Processor Nasty bit settings. This is the software model of DMA priorities. In reality, next cycle bus arbitrations are made every BUSCLK cycle. Initially, the processor priority is low. When the processor makes a bus request, next cycle bus arbitration is made with the processor set at this low setting. If a higher priority DMA request is outstanding, then GOAWAY is asserted and the processor is (presumably) forced to retry the access. If no higher priority DMA request is outstanding, then the processor is given the next cycle. This is achomplished by raising the processor priority for the duration of the access. When the bus arbitration which actually does choose the next DMA channel is executed, the processor will probably be highest. 'Probably' instead of 'will' was stated here because the processor priority will be raised to be under overlay priority. This means that the processor may wait state during the remainder of the cycle in which the request was made, plus the duration of a burst mode overlay access before actually given the bus. Note that the processor is given priority over RAM refresh. This works because refresh is made to withstand a delay of up to 7usec from the time a refresh request is made to the time the refresh is satisfied. However, the overlay channel cannot afford such a delay so it is given highest priority. Once the processor cycle has been executed, processor priority is lowered to normal so that new higher priority DMA requests may be served.

It should be mentioned that ANDREA satisfies burst mode cycles in the same manner the 68030 expects. The first longword fetched is from the address presented to ANDREA. The following three longwords are from locations sequentially following the first, with a modulos 4 rollover occurring on address bits A2 and A3. See the 68030 Users Manual for a detailed explanation.

It is not possible to terminate a burst mode cycle prematurely. Four longwords will always be transferred.

4.4.3 Bus Sizing. -

Both the 68020 and 68030 have built in dynamic bus sizing capabilities. Refer to the 68020 or 68030 Users Guide for a detailed description of operation. What this means is that the processor is capable of reading or writing bytes, words, and longwords to any byte address. Obviously if the processor attempts to write a longword to an odd byte address, more than one bus cycle will be required. The processor will write as many bytes as it can during the first cycle by indicating the byte address it is writing to and the number of bytes being transferred. It will then execute another cycle to write the remaining bytes.

Fully supporting this capability in the custom chips would make the chips' databus interface needlessly complex. However, it must be at least partially supported because code written for the A500, A1000, and A2000 talk to the custom chips using word addresses. It is also conceivable that old programs talk to the chips using longword accesses to do such things as fill the color registers. In a 68000 system, longword transfers are accomplished by doing two word transfers to word addresses.

The chip databus interface can be kept relatively simple, yet support any old addressing mode using the following scheme. When accesses are made to old RGA addresses, the processor is told that this address has a 16 bit peripheral chip attached to it. (The processor will also be told that RGA addresses are noncachable. GARY is responsible for communicating this information to the processor.) Depending on the transfer size, the processor will do one of two things.

When the access is a 16 bit write, the processor will put the same word on both the upper and lower words of the data bus. The custom chips will latch data from the part of the bus indicated by their input A1(address bit A1). A1=0 indicates the custom chips should latch old RGA data from CHIPDATA31-16, and A1=1 indicates the custom chips should latch old RGA data from CHIPDATA15-0. Note that latching data this way is consistent with latching data resulting from an old mode coprocessor 16 bit MOVE instruction.

When the access is a read from an old RGA address, the custom chips will respond by placing the same word on both the upper and lower words of the data bus. In this way, the processor can access the data from the upper word. If this was an ANDREA initiated DMA 'early read' cycle, ANDREA will strobe the appropriate CAS* signals(depending on address bit A1) to cause the word to be transferred into the correct word of RAM.

This scheme also works when the processor attempts to transfer longwords to/from the old RGA addresses. When the processor tries to write a longword to an old RGA address, the addressed custom chip will latch data from CHIPDATA31-16 because A1 will be zero. At the same time that this first word is transferred, the processor is informed that this address is a 16 bit port. This will cause the processor to put the other word on both the upper and lower words of the bus and do a 16 bit write to the odd word address. The custom chip will get this word from the lower portion of the bus because A1 will be 1. Processor longword reads from old RGA addresses will work similarly to writes.

Many new RGA addresses have been defined to support new modes of the next generation chip set. These addresses are longword addresses and all of the data bus will be used in the transfer. When the processor talks to one of these registers, it will be told that it's talking to a 32 bit peripheral.

As in the A1000 family, it is illegal to attempt to send bytes to an old RGA address. (There is always an exception, this one is BLPCON0L. In all other cases, the custom chips treat all relevant data as valid. Therefore arbitrary byte writes may cause unpredictable results.) In the next generation of custom chips, it is illegal to write a word or byte to one of the new RGA addresses. All transfers to new RGA addresses are longword transfers to longword addresses.

When the processor is addressing CHIPRAM, all 680n0 transfer modes are supported. When the processor is writing RAM, ANDREA will strobe only those CAS* signals indicated by A0, A1, SIZ0, and SIZ1. When the processor is reading RAM, ANDREA will strobe all CAS* lines to provide the entire longword from the associated longword address.

4.4.4 The 68040. -

Motorola was not as aggressive with it's 68040 data bus design as it has been in the 68020 and 68030 designs. Data duplication for misaligned or word or byte transfers is not done on the 68040 as it had in the 68020 and 68030 designs. What this means is that Motorola has finally said the heck with eight bit/sixteen bit peripheral chips. This new design will have an impact on our systems.

For the most part, the 68040 can be used directly with the AAA chip set, as was described in the previous section. The 68040 will issue to the bus the same type of transfer cycles the Copper issues. Old 16-bit RGA accesses will work for the same reasons that they work in 68020 and 68030 systems. The custom chips use the A1 signal to determine which word of the bus to read from, and the custom chips always duplicate 16-bit data on the upper and lower words of the bus. (Although the 68040 does not duplicate data in write operations, it does transfer byte and word data to/from the appropriate byte or word position on the data bus.) There is one transfer type which will fail in a system built as decribed in the previous section. That is the case where software written for 68000 systems used longword move instructions to load two successive 16-bit RGA registers. Since it is not possible to tell the 68040 that it is talking to a 16-bit peripheral, the 68040 will simply issue one transfer and think it's finished. However, only the even word will actually be transfered to the custom chips.

There are two possible solutions to this problem. One, we can say 'the heck with it', there aren't enough software packages doing this kind of thing to cause significant problems. Two, we can design the FIFO gate array chip to recognize when this type of transfer is occuring (if size indicates a longword transfer but address indicates old RGA space) and intercept the portion of the data which would otherwize be lost. The FIFO would then lock out further processor accesses and independently run a second transfer to complete the operation. The system designer responsible for the FIFO chip must make the appropriate decision.

4.4.5 RMC Cycles -

No predefined custom chip semaphores exist. No memory or register locations exist which have this kind of meaning to the processor. The custom chips, ANDREA in particular, will not respond to the processor RMC or LOCK signals in any way. It will be possible for the processor to begin a Read Modify cycle by reading a location of CHIPRAM and then see that location changed as the result of a Blit before the modifying write cycle is complete. It is the programmers responsibility to assure this and similar sequence of events do not occur. It is the responsibility of other hardware components (GARY) to deny other processor bus masters access to CHIPRAM during read modify write cycles.

4.4.6 Cache. -

It is very likely that systems which use the new chip set will incorporate an MMU and Cache. As was mentioned earlier, the Gary chip will use hardware to prevent the Motorola MMU from caching any of the custom chip registers. This is not the case with the chip RAM space,

and chip RAM data may be freely cached. System software should manage this ability carefully. For example, the processor should never have the capability of caching copper lists.

4.5 Burst Mode Cycles.

In order to generate more system bus bandwidth, the new chip set makes use of a burst mode transfer cycle whenever possible.

4.5.1 Blitter, Graphics Overlay, And Processor Burst Cycles. -

The standard burst cycle used by the processor follows the burst cycle format specified in the 68030 Users Manual. Basically, this cycle was designed to exploit quicker cycle times achievable in Nibble Mode, Fast Page Mode, or Static Column Mode DRAMs. (AAA directly supports Fast Page Mode DRAM's, and Fast Page Mode VRAM's.) One and three quarters bus cycle times are used in a burst mode fetch, to transfer four longwords of data. Thus, burst mode fetches allow bus bandwidth to be more than doubled. The reader is referred to the 68030 Users Manual for a complete description of the burst mode cycle used to satisfy processor burst requests.

Blitter, Sprite, and graphics Overlay burst cycles differ from processor cycles with respect to addressing, and in the case of Blitter and Sprite burst cycles, with respect to the total number of transfers possible. The modulos 4 rollover circuitry which is applied to address bits A2 and A3 when servicing a processor burst cycle is disabled during Blitter, Sprite, and Graphics Overlay burst cycles. This implies certain restrictions with respect to data alignment as will be covered in the discussions of subsequent paragraphs.

Graphics Overlay data must be located in memory such that all transfer cycles start at an even four longword address. Graphics Overlay data is always fetched using a four transfer burst cycle. Note that this requirement dictates that each scan line of Graphics Overlay data stored in memory contain an integer multiple of 128 pixels.

Sprite burst mode cycles can be shorter than the processor and Graphics Overlay burst cycles. The Burst64 control bit in the extended Sprite control register enables 64 bit Sprites, and causes two transfer burst cycles. The Burst128 control bit in the extended Sprite control register enables 128 bit Sprites, and causes four transfer burst cycles. See section 8.9, 'Burst Mode Sprites', for a description of certain restrictions which apply when burst mode Sprites are enabled.

4.5.2 Graphics Burst Cycles Using Fast Page Mode DRAM. -

It is possible to get even more bandwidth out of Fast Page Mode DRAMs than the 2.28x increase described in the previous section. The theoretical limit is related to the Page Mode or CAS cycle time spec'ed by the RAM being used, as well as the number of words transferred. In systems which use Fast Page Mode RAM, the limit is about 4x. Extended burst cycles are used when it is necessary to transfer large contiguous blocks of data. They can be used to burst up to 512(in the case of 1 Meg DRAM) words of data. Extended burst cycles are used in systems using Fast Page Mode DRAM to permit generation of some of the high

resolution displays.

Although extended burst cycles greatly improve the graphics capabilities of low end systems using Fast Page Mode RAM, they have the disadvantage of hogging the bus for relatively long periods of time. This means that the system can be slower to react to events than it would be if extended burst cycles were not used because the bus will not be relinquished until the transfer is complete. For example, to fetch a bitplane for an 800x560 monitor, the transfer will take as much as 2.1 usec or 7.25 bus cycles. This delay will not degrade system performance. To the contrary, since more data is being transferred in a shorter period of time, overall performance is actually increased.

Chunky pixel graphics could present a problem in light of the extended burst cycle. It would take 14.3 usec to fetch a high end 800x16 bit Chunky pixel line using an extended burst cycle. Some of the DMA channels could break if they are denied bus access for periods of time exceeding 6 to 7 usec. For this reason, extended burst cycles of chunky pixel transfers are interruptable in lieu of higher priority DMA requests. Chunky pixel graphics performance suffers slightly because of this. See section 4.6.6, 'DMA Channel Bandwidth Requirements'.

4.6 On Demand DMA.

New chip modes as well as the use of extended burst mode RAM cycles make the retention of the old fixed time slot DMA scheme impossible. The new system will use an on demand DMA scheme.

4.6.1 Motivation For New DMA Scheme. -

The new audio and disk modes provided by the chip set require much more data than that required by the old chip set. Compared to the old chip set, new audio modes (using only 4 of the 8 available channels) demand as much as 1.6 times as many cycles per second, new disk modes demand as much as 2 times as many cycles per second, and the new CD modes require 5 times as many cycles per second. New graphics modes use an even higher percentage of bus cycles as the old chip set. Burst mode cycles must be used wherever possible in order to buy back some of the bus bandwidth lost to the new modes. Use of burst mode transfers makes it difficult to dedicate any given cycle to a DMA channel as there could be a burst transfer in progress when that cycle occurs.

4.6.2 Processing DMA Requests. -

The new chip set will service DMA requests as soon as possible after they are made. Section 4.2.2, 'MARY/ANDREA Serial Link', discussed how MARY conveys DMA requests to ANDREA. Section 4.4, 'Processor Interface', describes how the processor makes DMA requests. ANDREA contains sufficient information to determine when other DMA channels need service. All sources of DMA requests are prioritized by ANDREA at the start of each bus cycle and the highest priority request is awarded the cycle.

4.6.3 DMA Priorities. -

The following list shows the relative priorities of each of the DMA channels.

Overlay Plane
DRAM Refresh
Interrupt Transfer
Disk
High Priority Bus Request (external request)
Audio 0
Audio 1
Audio 2
Audio 3
Audio 4
Audio 5
Audio 6
Audio 7
Sprite 0
Graphics

Order of Fetch to LINDA

HYBRID (Interruptable) or	CHUNKY (Interruptable) or	BITPLANE
RED Plane (Use BPL5PT)	Use BPL1PT	Bitplane16
GREEN Plane (Use BPL3PT)		Bitplane15
BLUE Plane (Use BPL1PT)		Bitplane14
		Bitplane13
		Bitplane12
		Bitplane11
		Bitplane10
		Bitplane9
		Bitplane8
		Bitplane7
		Bitplane6
		Bitplane5
		Bitplane4
		Bitplane3
		Bitplane2
		Bitplane1

Sprite 1
Sprite 2
Sprite 3
Sprite 4
Sprite 5
Sprite 6
Sprite 7
Coprocessor
Processor (Bus Request, external request)
Blitter

Two new bits have been added to the DMACONX register, PROPRI and BLTPRI. PROPRI, processor priority bit, gives the processor priority over the blitter. It is set on power up reset. Whenever BLTPRI is set and PROPRI is cleared, the relative priority levels between the processor and blitter indicated above are switched. The blitter will have priority over the processor. Whenever PROPRI is set and BLTPRI is cleared, the processor will have priority over the blitter. When both PROPRI and BLTPRI are cleared, the priority between processor and blitter will toggle each time the current high priority channel is serviced (the toggle will not occur if the channel serviced is not one of blitter or processor). This allows both processor and blitter to time share the bus. A final possible state which can occur is when both PROPRI and BLTPRI are set. When this occurs, the blitter will be the higher priority channel. Assigning the priority this way permits a

cheap way of acheiving compatibility when old software sets the BLTPRI bit after new software has set PROPRI.

Two special registers exist which grant unconditional processor access to the chip bus regardless of how BLTPRI is set. The intention behind these registers is that some priority adjust mechanism must exist which allows the processor timely access to the chip bus in order to service a pending interrupt request. If this feature did not exist and the blitter nasty bit is set, the interrupt routine would have to wait until current blitter activity is finished before being serviced. (Note that this feature will only boost processor priority above blitter priority. The processor may still have to wait for display or other high priority DMA to complete before being granted the bus.) The special registers are PRITEST and PRISET, Priority Test and Priority Set. Processor addresses will always be monitored for the occurrence of an access to these registers. When such an event occurs, for that one chip register access, the processor will be given priority over the blitter, regardless of how the PROPRI and BLTPRI bits of DMACONX are set. Reading PRITEST returns the PROPRI and BLTPRI bits while writing PRISET writes new values into these bits. An interrupt routine demanding service can then read the values of these priority bits, set processor priority for the remainder of interrupt routine processing, and then restore previous values of the bits upon completion of interrupt routine processing.

COPPRI, or the copper priority bit allows the copper full access to the bus after all higher priority requests have been serviced. When this bit is set, the coprocessor will starve the blitter and cpu of chip bus access until coprocessor goes inactive by executing a WAIT instruction, or until the coprocessor itself resets this bit. To achieve a situation which is more compatible to pre-ECS Amiga operation, the COPPRI bit is cleared. When this is the case, the coprocessor bus request signal is disabled every other bus cycle. A divide by four has been implemented which divides BUSCLK down to a 3.579Mhz clock, the bus cycle rate. When COPPRI is clear, the output of this divider is logically 'anded' with the coprocessor bus request signal. This allows the copper access to only 'odd' memory cycles, similar to old chip set operation. See section 5.7.3, 'Incompatibilities Between New Coprocessor and Old Coprocessor'. COPPRI is reset on power up, and whenever a coprocessor interrupt is generated. It is saved as part of the coprocessor status word and restored upon exit from the interrupt routine. If a copper interrupt routine wishes to execute in copper nasty mode, it should set this bit as it's first instruction.

The High Priority Bus Request is activated when ANDREA's HIGHRQ* input is asserted by external hardware. When this occurs, and when ANDREA can service the request (the request becomes highest priority), ANDREA asserts HIGHBG* (triggered by a rising edge of BUSCLK) and at the same time tristates the bus control signals specified in the following list. This list also indicates which output pins contain built-in pullup resistors. These pullups are included so that the custom bus master designer is not forced to drive all of the tristated signals if his application does not require it.

Signal Tristated	Pullup on Output
XADDR(3:0)	
AD(31:0)	
RAMADDR(9:0)	
CASH(3:0)	Yes
CASL(3:0)	Yes
RAS*	Yes
LATCH*	Yes
DT/OE*	
RR/W*	
ALE	Yes
A1	
A2	

At this point, external hardware is the chip bus master. It can hold the bus and execute as many memory cycles as desired by continuing to assert HIGHRQ*. The custom chips will regain control of the bus one BUSCLK cycle (following the next rising edge of BUSCLK) after the external hardware deasserts HIGHRQ*. The external hardware must tristate it's bus control signals at that time.

In the event there is a need to design more than one high priority chip bus master, it is possible for external logic to daisy chain several bus request signals so that all bus masters can gain high priority access to the chip bus. The user of this bus request input is forwarned that if he implementes a chip RAM bus hog, system performance will be severely degraded, quite possibly to the point of system failure. Note that it is also possible to daisy chain the processor bus request signal with others in order to obtain more than one low priority bus request line. In this case, external logic must provide the appropriate REG/RAM* signal and processor address, while ANDREA will drive RAS*, CAS*, etc as specified for the processor.

The Interrupt Transfer DMA channel is used to transfer processor interrupt information to MARY. It replaces a signal which, in the old chip set, communicated to PAULA that the blitter was finished and a level 3 interrupt should be generated. This function has been made a DMA channel in the new system to free an ANDREA signal pin. Whenever a blit operation finishes, this DMA request is made. ANDREA satisfies the request by setting the BLIT bit in MARY register INTREQ, and resetting the internal blitter finished flag. This DMA channel is also used to set the Graphics Data Overrun bit when ANDREA determines this error condition has occured.

4.6.4 Graphics Overrun Interrupt. -

A previous section mentioned that a graphics interrupt is generated whenever ANDREA fails to complete a graphics transfer during a scan line. This gives system software a signal that DMA bandwidth has been exceeded and it should react with some sort of system requester to the user to reduce the number of open DMA channels. Reacting to this signal will prevent the undesirable system attribute of a broken up screen.

Generation of the Graphics overflow interrupts is complex. In the simple case, it is simply a matter of detecting that the graphics DMA circuits have not completed fetching the last line of data when it came time to begin fetching the next line of data. In other cases, graphics

overflows are not so simple to detect. For example, it is permissible to continue fetching a line of data after the occurance of the next line if the system is set up in one of the screen promotion modes, and LINDA is busy redisplaying the last line of data. It is also permissible to continue fetching data if the system is currently programmed to display one of the PACKED pixel types. In this case, LINDA generates four lines of display data from one line of fetched information in non-promoted mode, and eight, twelve, or sixteen lines of display data when the system is in screen promotion mode repeating each line two, three, or four times. The rule for generating a graphics overflow interrupt is:
((still fetching last line of data) AND (not displaying a PACKED pixel) AND (ANDREA's signal TRANSFERVALUES occured (ie the secondary repeat counter hit 0))) OR ((still fetching last line of data) AND (displaying a PACKED pixel) AND (TRANSFERVALUES occured 4 times (in BPLCON3) times)).

4.6.5 DMA Limitations. -

In the old system, limitations of the system were well defined by DMA slots. There were a fixed number of slots allocated to the monitor so that fixed resolution displays could be generated. At the same time the disk and audio channels were guaranteed access to the bus so that they would always work at any monitor resolution. In some graphics modes, some of the Sprite DMA channels were used by graphics fetches so the corresponding Sprites became unusable. This same phenomenon occurs in the new system, except to a much higher degree. Now, there are no fixed DMA slots. It is possible to enable so much audio, disk, and graphics activity that there is not enough bandwidth available to accommodate everything. When this happens, the system will fail. Because of the way DMA priorities have been setup, the failure will not cause data loss. Instead, the graphics will fail (the screen will break up). The hardware checks for the occurance of this situation. When the hardware determines that graphics is failing, a Graphics Data Overrun interrupt is generated. When systems software receives this interrupt, it should immediately take corrective action. The graphics mode selected should be dropped back to some minimal configuration (perhaps a one bitplane screen should be brought up) until the present DMA load is resolved; perhaps some sort of system requester could be used to query the user as to which job to shut down. In any event, this discussion highlights the need for some sort of systems software DMA management. Such management software could prevent any DMA failure from ever occurring by monitoring system DMA load, and denying user access to those DMA channels which would overload the system.

The following section outlines bandwidth requirements for each of the DMA channels. Following that are two sections which show graphics modes possible, and CPU bandwidth remaining to the system when Audio, Disk, and Sprite DMA channels are enabled for worst case DMA bus hits(Sprites are assumed square 32x32x2bits, disk activity is continuous to a 4 meg drive, and four audio channels are putting out 16 bit samples at 51Khz).

4.6.6 DMA Channel Bandwidth Requirements. -

The following list shows or gives equations for the bandwidth requirements of each of the DMA channels. These constants and equations should be consulted to determine if it is possible to use a

given set of system resources at the same time. For example, the equations show that it is possible to display a 24 bit HYBRID screen at 640x400 in a system using Fast Page Mode DRAM. However, the equations also show that it is not possible to put up the same display when a 4 megabyte floppy drive is simultaneously being accessed.

Some constants:

Bus cycle time - 279.3651ns
Standard burst cycle - 69.841274ns
Video RAM shift cycle - 34.920637ns

	Pixel Frequency	Horizontal Frequency
NTSC	14.318181 Mhz	15734.3 hz
VGA	28.636363 Mhz	31468.5 hz
VGA+	36 Mhz	34816.3 hz
1Kx768	64 Mhz	48338.4 hz
1280x1K	110 Mhz	66465.3 hz

DMA Access Requirements

Channel	Access Time	Access Period
Refresh	1 cycle	15.625 usec
Disk		
Old GCR	1 cycle	64 usec
Old MFM	1 cycle	32 usec
New GCR	1 cycle	128 usec
New MFM(1 Meg)	1 cycle	64 usec
New MFM(2 Meg)	1 cycle	32 usec
New MFM(4 Meg)	1 cycle	16 usec
Audio	1 cycle	2 / SAMPLE FREQ
Overlay Plane	2 cycles	1 / PIXEL FREQ * 128
Sprite(32 bit)	2 cycle	1 / HORI. FREQ
Graphics		
Bitplane	4 * BUS CYCLE + (LINE LENGTH - BITS PER CYCLE) / BITS PER CYCLE * BURST CYCLE^	1 / HORI. FREQ
HalfChunky	4 * BUS CYCLE + (LINE LENGTH - PIXELS PER CYCLE) / PIXELS PER CYCLE * BURST CYCLE^	"
Hybrid	(4 * BUS CYCLE + (LINE LENGTH - PIXELS PER CYCLE) / PIXELS PER CYCLE * BURST CYCLE^) * 3	"

^ - BURST CYCLE = Standard Burst Cycle when Fast Page Mode RAM is used,
= Video RAM Shift Cycle when VRAM is used.

There are four checks which must be done to determine if a given system configuration and set of enabled DMA channels can coexist.

First check: determine if there is enough horizontal line time to fetch all the graphics data. (Graphics fetch time does not include Sprite DMA channel fetch time.) Use the equations to calculate how much time is required to fetch each plane. Add two bus cycle times to the result to account for the worst case situation where the graphics fetch is delayed a cycle because a burst mode cycle is only half complete when the graphics fetch request is made(it is necessary to add two cycles here because the equations assume a worst case fetch where a RAM page boundary is crossed before all data for the plane is retrieved). Multiply this number by the number of planes being displayed.

Second check: total all bus cycle time used by the DMA channels and make sure this total is less than a horizontal display line. The bus cycle time used by each plane of graphics fetches is two cycles worst case when fetching from VRAM. (The time given by the equations show how long it then takes to shift this data into LINDA.) When Fast Page Mode RAMs are used, the times given by the equations must be used.

Third check: Make sure that the sum of all higher priority DMA access time is less than each lower priority DMA channel's access period. This calculation shows whether or not a DMA channel will fail because it has been deprived of the bus too long while higher priority DMA channels are being serviced.

Fourth check: Make sure that the fetch time needed to satisfy any DMA request is less than all higher priority DMA channel's access period. This check is necessary when the system configuration uses Fast Page Mode RAM. In this case, graphics fetches hog the bus until the entire plane has been fetched. For example, in a low end, Fast Page Mode RAM, 800x560 system the bus will be unavailable for 2.03 microseconds while each bitplane is being fetched. Once such a fetch has started, it will execute to completion. No higher priority DMA channel can terminate such a fetch prematurely. Note that Chunky pixel fetches could tie up the bus for extremely long periods of time. For this reason, it is possible to terminate Chunky fetches prematurely to allow service to a higher priority DMA channel. When this is done, additional RAS precharge overhead time must be added to the time given by the equations. The amount of time which must be added is one cycle time for each higher priority DMA request which can occur while the DMA channel is being fetched. The net effect is that maximum display resolution possible is somewhat reduced. Only systems using Fast Page Mode RAM and only graphics DMA channels cause this long DMA fetch time. All other DMA channels and the graphics DMA channel of VRAM systems are serviced with one or two bus cycles.

4.6.7 System DMA Limitations With Fast Page Mode DRAM. -

Low End System, Fast Page Mode DRAM
Graphics Capabilities when 4 50Khz Audio, 4 Meg Disk, and Sprites are Enabled

Display Res.	Bitplane	HalfChunky	Chunky	Hybrid	PackLUT	PackHY
640/200	16	YES	YES	YES	YES	YES
704/200	16	YES	YES	YES	YES	YES
640/400	9	YES	NO	NO	YES	NO
800/560	8	YES	NO	NO	YES	NO

CPU Bandwidth (Megabytes/sec)

Display	640x200	640x400	800x560
2 BITPLANE	30.23	28.81	26.83
4 BITPLANE	28.8	25.96	22.0
8 BITPLANE	25.95	20.26	--
16 BITPLANE	20.24	--	--
HALF CHUNKY	27.15	--	15.98
CHUNKY	22.75	13.87	--
HYBRID	18.12	--	--

High End System, Fast Page Mode DRAM
Graphics Capabilities when Audio, Disk, and Sprites are Enabled

Display Res.	Bitplane	HalfChunky	Chunky	Hybrid	PackLUT	PackHY
640/200	16	YES	YES	YES	YES	YES
704/200	16	YES	YES	YES	YES	YES
640/400	14	YES	YES	YES	YES	YES
800/560	11	YES	YES	NO	YES	YES
1024/768	6	YES	NO	NO	YES	NO
1280/1024	3	NO	NO	NO	NO	NO

CPU Bandwidth (Megabytes/sec)

Display	640x200	640x400	800x560	1024x768	1280x1024
2 BITPLANE	30.82	29.98	28.96	27.11	24.2
4 BITPLANE	30.43	28.3	26.26	22.56	--
8 BITPLANE	28.3	24.95	20.86	--	--
16 BITPLANE	24.94	--	--	--	--
HALF CHUNKY	29.33	27.02	23.77	17.69	--
CHUNKY	27.12	22.6	16.18	--	--
HYBRID	24.69	17.74	--	--	--

4.6.8 System DMA Limitations With Video DRAM. -

Low End System, Video DRAM
Graphics Capabilities when Audio, Disk, and Sprites are Enabled

Display Res.	Bitplane	HalfChunky	Chunky	Hybrid	PackLUT	PackHY
640/200	16	YES	YES	YES	YES	YES
704/200	16	YES	YES	YES	YES	YES
640/400	16	YES	YES	YES	YES	YES
800/560	13	YES	YES	YES	YES	YES

CPU Bandwidth (Megabytes/sec)

Display	640x200	640x400	800x560
2 BITPLANE	31.2	30.75	30.37
4 BITPLANE	30.74	29.83	29.08
8 BITPLANE	29.83	28.01	26.5
16 BITPLANE	28.0	24.36	--
HALF CHUNKY	31.54	31.43	31.34
CHUNKY	31.54	31.43	31.34
HYBRID	31.31	30.97	30.69

High End System, Video DRAM
Graphics Capabilities when Audio, Disk, and Sprites are Enabled

Display Res.	Bitplane	HalfChunky	Chunky	Hybrid	PackLUT	PackHY
640/200	16	YES	YES	YES	YES	YES
704/200	16	YES	YES	YES	YES	YES
640/400	16	YES	YES	YES	YES	YES
800/560	16	YES	YES	YES	YES	YES
1024/768	11	YES	YES	YES	YES	YES
1280/1024	8	YES	NO	NO	YES	NO

CPU Bandwidth (Megabytes/sec)

Display	640x200	640x400	800x560	1024x768	1280x1024
2 BITPLANE	31.21	30.76	30.4	29.86	29.32
4 BITPLANE	30.76	29.87	29.05	28.07	26.99
8 BITPLANE	29.86	28.08	26.62	24.49	22.32
16 BITPLANE	28.07	24.5	21.59	17.32	12.99
HALF CHUNKY	31.4	31.15	30.92	30.57	30.18
CHUNKY	31.37	31.08	30.8	30.35	--
HYBRID	30.89	30.13	29.46	28.4	--

4.7 RGA Bus.

Pin limitations of the new chip set require that the RGA bus be multiplexed onto the chip data bus. The timing diagrams of Appendix A show when the bus contains RGA addresses, and when the bus contains data. The falling edge of ALE, Address Latch Enable, latches the RGA address into the custom chips. RGA addresses are guaranteed to be valid 30ns before ALE falls. When ALE falls, ANDREA tristates the data bus meaning RGA addresses hold for 0ns from the falling edge of ALE. If the access is a register read operation, the responding chip should drive data onto the bus after the next falling edge of BUSCLK. Data must be valid on the bus 30ns after that time. If the access is a register write operation, data should be latched by the chip with the rising edge of ALE. ANDREA contains a "soft latch" mechanism which holds data on the bus for 30nsec from the latching edge of ALE. (An exception to this is the write of MONICA's overlay plane BPLnDAT registers. When the overlay plane registers are being burst filled, MONICA uses the rising edge of LATCH* to latch data.)

In the old chips, there were several strobe RGA addresses which were used to synchronize events within the chips. In particular, these strobe addresses were STREQU, STRVBL, STRHOR and STRLONG. With the new On Demand DMA scheme, it is not possible to generate these RGA strobes at exactly the same moment from line to line. Instead, identical functionality is provided with the ANDREA output signals HRESET, and VBLANK. The old RGA addresses (STREQU, etc) have not been used in the new chip set.

The old chip set had a special bus cycle called an early read cycle. This cycle was used by the blitter destination DMA channel, and by the disk read DMA channel. This cycle was used to guarantee data setup time to the RAMs before CAS*. The bus timing of the new chip set handles this kind of cycle by delaying the fall of CAS* 35ns from the point when CAS* falls during register write operations.

4.8 RGA Address Space.

The following chart shows the new RGA address space. Appendix E gives an address sequential list of all the RGA registers, and Appendix F gives an alphabetic list of all registers and a description of all the register bits.

RGA Memory Map(Base address is 00DFFnnn)

Address	Description
000-1FF	Old RGA Addresses. 256 word addresses. All transfers to this address space are 16 bit word transfers.
200-7FF	32 Bit Control Registers. 384 longword addresses. All transfers to this address space are 32 bit longword transfers. These registers extend old modalities to 32 bits, and define new modes which didn't exist in past revisions of the AMIGA chipset.
800-BFF	New Color Registers. 256 longword addresses. All transfers to this address space are 32 bit longword transfers. These are the read addresses for COLOR0 thru COLOR255.
C00-FFF	New Color Registers. 256 longword addresses. All transfers to this address space are 32 bit longword transfers. These are the write addresses for COLOR0 thru COLOR255.

4.9 RAM Configurations.

In order to satisfy as broad a spectrum of users as possible, the AAA chip set has been designed to accept and use more than one type of DRAM. This approach allows the chip set to be used in as low cost a system as possible, or in a higher performance system which uses more expensive VRAM. With appropriate systems design, it is possible to build a system capable of mixing low performance Fast Page Mode DRAM with high performance Video RAM. A user could initially start with a cheap system, and then later buy a high performance VRAM expansion module to upgrade his system.

4.9.1 RAM Cycle Timing. -

All chip RAM is cycled with a period of 279.3648nsec. This is the same period as was used in the old chip set. As has been previously mentioned, the new chip set supports a burst mode transfer. A full burst mode transfer takes 7/8's the amount of time as two single access transfers but accesses 4 times the amount of data. See the timing diagrams of Appendix A for a detailed look at the new bus cycle.

4.9.2 Standard Fast Page Mode DRAM. -

Fast Page Mode DRAM permits the design of cheap systems. All fetches made from this type RAM chip are either single cycle or burst type fast page mode. The standard burst mode fetch transfers 4 longwords of data in a 1.75 bus cycle period. Graphics fetches are made with an extended burst cycle and transfer 4 longwords each 280nsec period. CPU, Blitter, and other DMA channel bandwidth suffers when graphics fetches are made from this type of RAM, but the overall system performance is much better than that which can be achieved in the old Amiga systems.

4.9.3 VRAM. -

Video DRAM provides an isolated graphics data bus. This isolated bus allows simultaneous graphics transfers and CPU or other DMA transfers. The effective system performance is therefore almost doubled. Transfer rates on the graphics data bus is twice that which can be achieved in the Fast Page Mode system; higher resolution graphics can be generated. Because the parallel port of VRAMs used in the AAA system support Fast Page Mode cycles, standard burst cycles are still available for CPU and Blitter operations.

4.9.4 Mixed RAM Systems. -

It is possible to configure a system (both low and high end) with both VRAM and DRAM. ANDREA uses RAM attribute data fetched at system reset time in order to determine what type of RAM is being accessed. When a given fetch is a graphics mode fetch and starts at an address within a VRAM block of memory, then a shift register load sequence is initiated. It is possible that a page boundary will be crossed while the graphics data is being shifted into the graphics chips. When this occurs, ANDREA initiates another fetch to retrieve the remaining data.

Without proper software precautions in place, it is possible that a page boundary crossing could be made across two adjacent RAM modules. ANDREA has a limitation that in such an event, all RAM accessed for the graphics fetch must be of the same type. In other words, all graphics data must reside entirely within VRAM, or entirely within DRAM. System software must guarantee this when graphics memory allocations are made. There are no compatibility problems associated with this restriction. This is because in pre-ECS systems, chip RAM was limited to 1Meg. Therefore old software would not attempt to access more than 1Meg. Each RAM bank in the new system contains a minimum of 1Meg. Old software will execute entirely out of the bottom Megabyte of chip RAM when run on a AAA machine.

4.9.5 Chip RAM Address Space. -

The new chips have been designed to operate in one of two distinct systems: in a low-end system configuration, or in a high end system configuration. Low-end systems can access up to 8 megabytes of chip RAM, while high-end systems can access up to 16 megabytes of chip RAM. The location of chip RAM address space with respect to the host processor is arbitrary from the chip set's point of view. The chip set however, views chip RAM as a contiguous block of memory starting at address 0. It believes that low-end system chip RAM is expanded in 1 megabyte increments, and that high-end system chip RAM is expanded in two megabyte increments.

The RAMATTR register (see appendix F) contains information which defines the users' RAM configuration to the system. This register is automatically written at system reset time. It contains eight fields, one field for each expansion increment possible. Each of these fields contain information which indicate: 1) whether or not a memory module is present in the corresponding address segment, 2) the number of physical address bits needed to properly interface to the memory module, 3) whether or not the memory module contains a BLIT chip, and 4) whether the RAM present at the corresponding address segment is standard Fast Page mode DRAM, or Fast Page mode VRAM.

Certain dynamic RAM chips available today, or possibly available within the lifetime of this chip set cannot be made to fit into the expansion scheme outlined above. Some RAM's can only be used if the expansion increment is greater than 1 megabyte. It is still possible to use these dynamic RAM chips in AAA system designs. Large RAM chips can be accomodated by designing the auto config logic on the expansion card to drive multiple RAMATTR nybbles to match the number of one megabyte segments which exist in the module. For low-end system designs, the expansion module is designed to drive as many RAMATTR nybbles as there are one megabyte increments of physical memory. For high-end system designs, the expansion module should drive as many RAMATTR nybbles as there are two megabyte increments of physical memory. In these cases, each RAMATTR nybble driven must contain identical data.

The following table lists some examples of existing, as well as theoretically possible dynamic RAM chips which can be used in AAA system designs.

Example Part	Number of bits	Number of Address Lines	Configuration	Required Number of chips	Expansion Increment
Fujitsu 81C258	256K	9	256K x 1	32	1M byte
Toshiba 524256	1M	9	256k x 4	8	1M byte
OKI 514251	1M	9	256K x 4	8	1M byte
NEC 482234	2M	9	256K x 8	4	1M byte
Mitsubis 482257	2M	9	256K x 8	4	1M byte
NEC 424260	4M	9	256K x 16	2	1M byte
???	4M	10	1M x 4	8	4M byte
???	8M	9	256K x 32	1	1M byte
???	8M	10	1M x 8	4	4M byte
???	16M	10	1M x 16	2	4M byte
???	32M	10	1M x 32	1	4M byte

The following table lists some examples of existing as well as theoretically possible dynamic RAM chips which cannot be used in AAA system designs.

Example Part	Number of bits	Number of Address Lines	Configuration	Required Number of chips	Expansion Increment
NEC 41264	256K	8	64K x 4	8	256K byte
Hitachi 53462	256K	8	64K x 4	8	256K byte
???	4M	11	4M x 1	32	16M byte
???	16M	11	4M x 4	8	16M byte
???	16M	12	16M x 1	32	64M byte

4.9.6 Timing Implications Of Burst Mode Accesses. -

The burst mode timing diagram of Appendix A shows that the RAM's are cycled at the spec rate of 70ns when in burst mode. Under these conditions, data is guaranteed to be valid on each rising edge of CAS* when total rise and fall delay of CAS* is less than 10ns(and the total bus load is <100pFd+2TTL loads). This timing is OK for most systems, but is too slow when the user is using a multiple blitter system. In this case, the Blit chips stand directly in the data path and therefore their input to output propagation delay must be added to the access time. Since it is not desirable to make the burst cycle longer, a scheme has been incorporated to account for the additional delay which must be added to the cycle.

Basically, burst mode accesses made to the RAMs are pipelined. Data transferred from the RAM chips to the chipdata bus is latched by the Blit chip with a rising edge of CAS*. The data is then passed to the chipdata bus while CAS* is high and latched into ANDREA, MONICA, or the processor FIFO chip with a rising edge of LATCH*. This rising edge of LATCH* coincides with the next falling edge of CAS*, so data must be transfered before the RAMs are enabled to drive the next word. This is fine when there is a Blit chip in the system to intercept and pass the data, but leads to potential problems when there is no Blit chip in the data path. What would happen in this case is that the RAM chips will go high-Z when CAS* goes high. The receiving chip will be forced to latch data from a floating bus. This is a reliability hazzard. It would be possible for ANDREA to advance LATCH* when she determines that the addressed RAM does not contain a Blit chip. This would mean that

all destination chips be capable of accepting data at variable times with respect to BUSCLK. To keep the interface as simple as possible, a different scheme has been implemented. LATCH* timing remains constant with respect to BUSCLK. To assure that valid data is latched into the destination, ANDREA latches the databus when she determines she is addressing RAM with no Blit chip. When CAS* goes high, ANDREA drives the data she latched onto the bus. The drivers used for this function are not normal databus drivers, but instead drivers of a size just big enough to assure that valid data is latched by the receiving chip. RAM specs indicate that the data pins may leak up to +-10uamps of current when they are tristated. As there could be up to 8 chips on the databus, these sustaining drivers must be capable of delivering at least 80 uamp of current. Designing them to provide 1-2ma of sustaining current will assure plenty of margin, yet allow the value lathced to be overdriven.

4.9.7 Automatic Configuration. -

The new chips support use of two types of RAM. In addition to this a user can expand his system with memory cards which contain external BLIT chips. ANDREA and software must know what kind of RAM is in the system, either DRAM or VRAM. Software must also know whether or not the system contains external BLIT chips. ANDREA performs an automatic configuration sequence when the system is powered up to determine what the memory configuration is.

ANDREA uses the first memory cycle following a system reset to read configuration data about chip memory. This cycle follows the same timing as a normal RAM read cycle. It differs from a RAM read cycle in that instead of the CAS* lines being asserted, CONFIGRD* is asserted. CONFIGRD* is a signal which specifies to auto config logic built into each RAM module or card that it is to drive it's configuration data onto the AD bus. It has the same timing characteristics as would normally be seen on the CAS* signals. The printed circuit mother-board must be designed such that each memory module drives onto a different portion of the AD bus when CONFIGRD* is asserted. See the regbits description of RAMATTR (Appendix F) for bit positions each slot is to drive. The mother-board should also incorporate logic such that zeros are driven during CONFIGRD* cycles when no RAM expansion board has been plugged into a slot.

Auto config data is written into RAMATN. ANDREA uses the data to determine how to access RAM when performing bitplane DMA fetches. RAMATN data indicates how much chip RAM is available, whether or not external BLIT chips are present, and what kind of RAM is present at each segment of memory, DRAM or VRAM.

4.9.8 RAM Configuration Restrictions. -

The chip set provides the systems designer with a high degree of flexibility with respect to combinations of RAM chips which can be designed to plug into a given system. However, this flexibility is limited: certain restrictions apply.

At auto config time, the chip set automatically adjusts the size of the RAMADDR bus to be 9 bits wide, or 10 bits wide. The 'Address Lines Valid' bit of the nybble read from RAM module location 0 (RAMATRN register bit 2) during the CONFIGRD* cycle specifies which to use. All

RAM modules designed to function in a given system must use this number of address lines. It is not possible to design two RAM modules, one which uses a 1M x 4 RAM chip (10 bit address bus), and one which uses a 256K x 16 RAM chip (9 bit address bus) for use in the same system.

It is not possible is use a RAM module in a high-end system (dual video chips) when that module has been designed for use in a low-end system. The opposite is also true.

Expansion must be made in a contiguous manner, without 'holes' in the address range. It is not possible to plug one expansion module into the system at the second memory segment, and another module at the fourth memory segment without also providing a module at the third memory segment.

It is not possible to use the external BLIT chip feature when the RAM type used requires expansion increments of other than one megabyte.

4.10 FrameGrabber Interface.

1meg VRAMs support serial port write operations as well as standard read operations. This capability opens up the possibility for a new type of Amiga FrameGrabber device. Two features make this idea worth pursuing. 1) The device does not require it's own frame buffer RAM. Instead it uses chip VRAM, thus allowing a significant cost savings 2) Due to the data path utilized in the transfer, every field of data is captured. The FrameGrabber provides real time framegrabbing capabilities.

The 'DIR' bit in BPLCON3 causes the custom chips to go into framegrabber mode and begin capturing externally generated video data. When this bit is reset, the chips are in normal display mode, and Andrea controls the fetch of video data from RAM for display. When 'DIR' is set, Andrea commands the Linda chip to capture data from the VDATA bus (the bus between Linda and Monica), and then present that data to the GDATA bus for write into the serial port of the VRAM. Andrea provides all necessary control signals to drive the serial port write cycles.

Only VRAM, the 1200 (ANDREA) chip, and the 1202 (LINDA) chip are required to support framegrabber mode. It is possible to design a stand-alone framegrabber using these components, a simple composite video to RGB converter, and (when limited color depth pixel capture is desired) a simple multiplexing circuit which serializes pixels smaller than 24 bits into 32 bit groups. This configuration can also be cheaply extended to provide limited display functionality. Instead of including the 1201 (MONICA) chip, one need only provide a set of RGB DAC's for 24-bit pixel display. 16-bit chunky pixels could also be displayed with the addition of a simple de-multiplexing circuit.

The AAA framegrabber mode can prove useful in CDTV applications. It is desirable to interface CDTV to off the shelf MPEG chips so as to provide full motion video functionality. With a mimimum of discrete component overhead (quite possibly none), it is possible to connect the digital video output bus of Motorola's MCD1450, or C-Cube's CL450 chip directly to the AAA framegrabber interface. This provides a cost effective method for injecting an MPEG video stream into the Amiga video stream.

4.10.1 Theory Of Operation. -

The block diagrams of Appendix C shows where the external video source enters the system when the chips are in Framegrabber mode. Support circuitry not shown in the block diagrams consists of: 1) A composite video to RGB converter to precondition the data. 2) Hsync and Vsync extraction circuitry. 3) Latches and associated control logic to buffer from one to eight (dependent on pixel mode selected, and system configuration (low or high end)) successive pixels for presentation to Linda and Monica.

When the external framegrabber circuits are included in the system, the chips must be programmed to operate in Genlock mode. Genlock mode provides the necessary mechanism for the graphics subsystem to lock onto the external video source. (Recall that when the chip set is in Genlock mode, ie ERSY=1, Andrea's Hsync and Vsync pins become inputs, and are used to reset the horizontal and vertical counters.) During operation, the external framegrabber circuitry is providing video data to both Linda and Monica in the same format and with the same timing that Linda provides graphics data during normal display mode. Monica accepts this data and in turn sends it to the display as if it were coming from Linda (in fact, Monica is not aware that there is such a thing as framegrabber mode). Linda, inputs the same data and saves it in a line buffer. On the next scan line, under the control of Andrea, and possibly at the same time she is inputing the next lines' video data, Linda accesses the line buffer data and presents that scan line of video data to the GDATA bus for storage in VRAM.

Andrea generates VRAM addresses for storage of the video data in much the same manner as the way she generates addresses during normal display modes. Depending on the pixel type, the bitplane pointer register(s) are accessed to determine the starting address for the scan line, and incremented as each RAM address is written. At the end of each scan line, the bitplane modulus register is added to the bitplane pointer register to advance the pointer to the first location to be written on the next scan line. After the entire display field has been processed, the bitplane pointer register must be reset back to the first line of the display field, or initialized some some other area of memory when a sequence of display fields is to be captured. Typically, the copper is programmed to perform this reset operation. To use framegrabber mode for single field capture, software simply programs the graphics registers as if any of the normal display modes were being programmed, with the exception that the ERSY bit in BPLCON0 is set (Genlock mode), the DIR bit of BPLCON3 is set (framegrabber mode), and DDFSTRT and DDFSTOP are adjusted according to the following table.

Graphics Mode	DDFSTRT and DDFSTOP Adjustment Factor for Framegrabber Mode
Low-End Hybrid	normal display setting + 8
Low-End Chunky	normal display setting + 2
Low-End Half-Chunky	normal display setting + 2
High-End Hybrid	normal display setting + 14
High-End Chunky	normal display setting + 4
High-End Half-Chunky	normal display setting + 4

This being done, the system will capture every field of video generated by the external framegrabber circuits. When software wishes to exit capture mode and redisplay the last captured field, say due to the press of a mouse button, the software must first wait until Vsync so as to complete capture of the current screen. At that point, in addition

to the normal resetting of bitplane pointer registers, the DIR bit of BPLCON3 is cleared, and the DDFSTRT and DDFSTOP registers are skewed back to the normal display settings. (See section 6.7, 'Graphics Programming', for directions on how to establish normal display settings for DDFSTRT and DDFSTOP.) All other graphics control registers have already been set up for the current graphics display mode, and therefore do not need to be modified. The source of display data is now the VRAMs. When it is again time to enter capture mode, software should wait until Vsync, set the DIR bit, and modify the DDFSTRT and DDFSTOP registers to again provide a framegrabber mode fetch window.

4.10.2 External Framegrabber Circuitry. -

As has been previously mentioned, any system which incorporates a framegrabber device must be programmed to operate in genlock mode. The framegrabber circuitry will be constantly asserting XCLKEN and XCLK to the clock generation circuits, and constantly asserting Vsync and Hsync to Andrea. See section 5.4, 'Genlock Mode'.

The external framegrabber circuits accept control signals from Linda, a DIR control signal, a OT(2:0) control bus, a OHI/LO* control signal, and a DFW (delayed fetch window) control signal. The DIR signal indicates when Linda is in capture mode, and therefore when the circuitry should be asserting (DIR=1, assert) video data onto the VDATA bus. The OT bus indicates the expected pixel type. External framegrabber circuitry must format and present pixel data according to this bus (same value programmed into the OBPLMODE field of the BPLCON3 register).

OT(2:0) = 1, Hybrid pixels
OT(2:0) = 2, Chunky pixels
OT(2:0) = 5, Half-Chunky pixels.

The OHI/LO* control signal indicates whether the expected video data is in high resolution format, or low resolution format. The DFW signal indicates that portion of the scan line that the external framegrabber circuits are to go active and present display data to the AAA chips. External framegrabber circuits must monitor one additional signal when deciding how to format and present pixel data, DUAL/SING*. This signal indicates whether the system is configured as a low end or high end system. In the high-end system, twice as much data (64 bits) is sent to the graphics chips each transfer as in the low-end system.

The external framegrabber circuits are responsible for the generation of a control signal, CAPEN in addition to presenting pixel data. This signal must be driven when DIR is high and when DFW is asserted (tristated when DIR is low). CAPEN indicates to the graphics chips the cycles when valid pixel data is to be latched from the VDATA bus. DFW indicates to the external framegrabber circuits that portion of the display line to capture. DFW is derived by the graphics chips from the DDFSTRT and DDFSTOP registers, and when programmed properly defines a window which is several cycles advanced from the window specified by the DIWSTRT and DIWSTOP (display window) registers. Timing diagrams in Appendix A show expected framegrabber circuit responce to the DIR and DFW signals for each of the supported pixel modes.

4.10.3 Framegrabber Limitations. -

The chip framegrabber circuits have been set up to be as flexible as possible with respect to overscanned lines, however there are limitations. It is not possible to enable LINDA's framegrabber circuits for entire display lines. External framegrabber circuits should not assert CAPEN during the horizontal sync and blanking regions, and software should not set DDFSTRT and DDFSTOP to specify capture for entire display lines. To prevent malfunction in framegrabber mode, the DDFSTRT and DDFSTOP registers should be set to disable DFW for a minimum of twenty-four PCLK cycles in low-end systems, and a minimum of twelve PCLK cycles in high-end systems. This inactive period should occur during the HSYNC interval of the scan line.

Framegrabber mode can only be used in systems which contain VRAM.

The graphic chip VDATA bus can only be operated in voltage mode (TTL logic levels) when framegrabber mode is enabled.

Due to the nature of Framegrabber mode, the system display monitor must be capable of running at the same horizontal and vertical frequencies as the external video source being captured. It is not possible to enter any of the screen promotion modes when the system includes framegrabber circuitry. In the same light, it is not possible to enter any of the screen promotion modes when the system includes a genlock device.

Hybrid, Chunky, and 8 bit half-chunky graphics can be selected for use with framegrabber mode. There is no built in hardware support for 8 bit half-chunky framegrabber mode. If this mode is desired, the systems designers must provide a mechanism for calculating best fit pixels for use in the Color Lookup Table. An interface to the RGADDRESS/DATA bus would also have to be provided in order to transfer LUT information to MONICA at the beginning of each scan line. Half-Chunnky framegrabber mode does provide an efficient and cheap means for framegrabbing black and white images. In this case, costly best fit pixel estimation logic would not be necessary. Instead, one would map captured 8-bit luminence level values directly into LUT addresses. The LUT would then be programmed by software in advance to contain corresponding sequentially increasing gray level values.

4.10.4 Genlock Capabilities. -

The FrameGrabber has marginal Genlock capabilities. The overlay bitplane available in chunky pixel modes could be used for single color text overlays. (The overlay bitplane is fetched over the RGADDRESS/DATA bus and will not interfere with FrameGrabber operation.)

4.11 Custom Chip Pin Descriptions.

This section describes each pin of each chip in the new custom chip set.

IF "ADDR = RAM" * REG/RAM* * NEED* * AS* THEN TEST
IADDR = A23-A12 = Ø

4.11.1 ANDREA (1200). -

ANDREA provides the same functionality as the old AGNUS. This chip contains 126 signal pins and is packaged in a 144 pin QFP. Following is a description of each of ANDREA's IO pins.

PADDR0-23	Input	Processor Address Bits 0 thru 23.
SIZ0-1	Input	Processor Size Bits 0 and 1.
PR/W*	Input	Processor Read Write signal.
REG/RAM*	Input	Register Enable. An upper address decode from GARY. When high, this signal indicates the processor is accessing an RGA address. When low this signal indicates the processor is accessing RAM.
NEED*	Input	Processor cycle request. This signal indicates the processor is requesting the bus. Depending on BURSTRQ, either a single cycle or burst cycle will be granted when the bus becomes available.
AS*	Input	Address Strobe. This is the processor address strobe, used to latch processor addresses and qualify the internal processor request signal. This signal is sychronized to BUSCLK by sampling only while BUSCLK is low.
GOAWAY*	Output	Go Away. This signal indicates that the processor cannot be granted the next bus cycle as requested so external control logic should execute a processor cycle retry sequence.
BG*	Output	Bus Grant. A signal indicating the processor has been granted the next bus cycle(could be either single cycle or burst cycle depending on the request made).
BURSTRQ*	Input	Burst Request. When this signal is high and NEED* is asserted, the processor is requesting and will be granted a burst cycle.
LATCH*	Tristate Output with pullup	Data Latch. This signal is used to latch data into a data bus interface holding buffer. The rising edge of this signal should be used to latch the data. This signal is tristated during HIGHBG* cycles.
RAMADDR0-9	Tristate Output	RAM Address 0 thru 9. Multiplexed RAM address. RAMADDR9 is only used when RAM chips requiring 10 address lines are designed into the system. The RAMATTR register is accessed to make this determination, and controls the address multiplexing. This bus is tristated during HIGHBG* cycles.
CASH*0-3	Tristate Output with pullup	Column Address Strobes for Lower Longword. There is one CAS* signal for each byte in the longword to support processor byte addressing capabilities. This group of CAS signals is used to drive all accesses made in the low end system, and accesses to even longwords in the high-end system. These signals are tristated during HIGHBG* cycles.
CASL*0-3	Tristate Output with pullup	Column Address Strobes for Upper Longword. This group of CAS* signals are only used in high end systems. There is one CASH* signal for each byte of the longword. This CAS* bus is used to drive all odd longword accesses in the high-end system. These signals are tristated during HIGHBG* cycles.
HIGHRQ*	Input	High Priority Bus Request. This request line supports external logic which needs access to the chip data bus.
HIGHBG*	Output	High Priority Bus Grant. A signal indicating that the external device can assume control of the RAM bus.

CONFIG_RD*	Output	RAM Configuration Read. ANDREA asserts this signal on the first bus cycle after a system reset in order to read RAM configuration data. When asserted, CONFIG_RD* has the same timings as CAS* signals.
RR/W*	Tristate Output	RAM Read Write. The read write signal which controls the AD RAM port. This signal is tristated during HIGHBG* cycles.
DT/OE*	Tristate Output	Data Transfer / Output Enable. A control signal for VRAM. This signal is tristated during HIGHBG* cycles.
LATCHSE	Output	Latch Serial Port Enable decode. When this signal is asserted, the information on XADDR0-3 is decoded and latched by external hardware to enable the serial port of a given VRAM bank.
SEEN*	Output	SE Decoder Enable. When this signal is asserted, the output of the external SE decoder circuit is enabled so that one SE line can drive low. When deasserted, all SE outputs from the external decoder are driven high.
RAS*	Tristate Output with pullup	Row Address Strobe. This signal is used by an external one of eight decoder to enable RAS* signal generation and drive one of the 8 possible RAM banks. This signal is tristated during HIGHBG* cycles.
BRAS*	Output	Blitter Row Address Strobe. This signal is asserted rather than the normal RAS* signal whenever an external Blit chip RAM access is run. The normal RAS* signal is disabled in this case. Driving BRAS* for external Blitter cycles prevents the ANDREA internal blitter from trashing parts of memory as a result of the cycle.
RFSH	Output	Refresh. A signal indicating that this access is a RAM refresh operation and all RAS* signals should be asserted when RAS* falls.
XADDR0-2	Tristate Output	Expansion Address. This bus contains a hex value which indicates which of the 8 RAM banks is being addressed. This bus is tristated during HIGHBG* cycles.
DUAL/SING*	Input	Dual / Single. This input signal tells ANDREA whether she is in a low end, single chip or high end dual chip system.
TDATA	Output	Graphics Type. This is a serial link to LINDA. The following information is transferred from ANDREA to LINDA on this link: HIRES/LORES, five bits of BPU, three bits of type data(indicating graphics type), SWITCH(a bit telling LINDA to switch RAM banks), DIR(a bit which tells LINDA data is flowing to or from RAM), RP0 and 1 (LINEPRT0 and 1 of BPLCON3), PF1H0, and PF2H0. This data is transferred after HTOTAL, but before DFTCHWIN is asserted.
TCLK	Output	Clocks LINDA's TDATA port. Only active when data is being transmitted.
DFTCHWIN	Output	Data Fetch Window. This signal is derived directly from the values programmed into the DDFSTRT and DDFSTOP register. It provides LINDA with a time reference and indicates when LINDA is to begin processing a new scan line. Due to timing restrictions in LINDA, this signal is only changes on falling edges of PCLK.
SC	Output	Shift Clock. This signal shifts data out of (or into in framegrabber mode) the VRAM serial port.
SCACT	Output	Shift Clock Active. This signal is asserted by ANDREA when SC to the VRAMs is active. It is used by LINDA to determine when to latch graphics data into the chip. When graphics data is being fetched from Fast Page Mode

		RAM rather than VRAM, SCACT is asserted when CAS* is being driven.
SHFTCLK	(C) Output	This is a free running clock which usually has the duty cycle, phase, and frequency of SC. The frequency of this clock can change on the fly to match CAS* signal timing. This clock is driven to full CMOS voltage levels (min VSS+.2v to VCC-.2v).
A2	(C) Tristate Output	Bitplane Starting Address bit 2. The old AMIGA chip set allowed bitplanes to start on any word boundary. LINDA uses A2 and A1 for data alignment purposes. A2 is latched at the output driver with the falling edge of RAS* and held until the next falling edge of RAS*.
A1	(C) Tristate Output	Address bit 1. This signal indicates which part of the data bus (upper or lower word) contains valid data when old RGA registers are written. It also specifies to LINDA which word boundary to begin bitplane alignment to. A1 is latched at the output drivers with the falling edge of RAS* and held until the next falling edge of RAS*. This output must drive full the CMOS level (min VSS+.2v to VCC-.2v).
NXTPLANE	Output	Next Plane. This signal tells LINDA that she should begin inputting data for the next bitplane.
BUSCLK	(C) Output	Bus Clock. Used along with ALE to synchronize bus read and write transfers. This clock is derived by dividing MCLK by 4. This output must drive full the CMOS level (min VSS+.2v to VCC-.2v).
ALE	(C) Tristate Output with pullup	Address Latch Enable. RGA addresses are latched into the custom chips from the data bus (AD) with each falling edge of ALE. Register read or register write data is valid on the bus when ALE is low. Register write data is latched into ANDREA when ALE goes high. This output is tristated during HIGHBG* cycles. It drives full the CMOS levels (min VSS+.2v to VCC-.2v).
AD0-31	I/O	Register Address / Data bus. Multiplexed RGA / Data bus.
RESET*	Input	Reset. This is the system reset signal. This signal must be held asserted for a minimum of 10 msec at system power-up time. (This required power on reset sequence is not to be confused with a processor reset sequence, known to be of shorter duration.)
VBLANK*	Output	Vertical Blanking. This signal causes MARY to generate a vertical interrupt if that interrupt has been enabled. VBLANK* replaces the old STREQU and STRVBL RGA address strobes. When the system is in NTSC or PAL mode, this signal occurs at the same horizontal count that the strobe addresses were written in the old system.
HRESET	Output	Horizontal Reset. This signal indicates the start of a new horizontal line. This signal is asserted while Andrea's counter is at count 8. The hardwired decode 'COMP_LINE_INC' is used to generate this signal.
BURST*	Output	This signal enables external hardware to drive color burst.
HSYNC*	I/O	Horizontal Sync. Input in GENLOCK mode.
VSYNC*	I/O	Vertical Sync. Input in GENLOCK mode.
CSYNC*	Output	Composite Sync.
CBLK*	Output	Composite Blanking. This signal is generated by logically 'OR'ing the vertical and horizontal blanking signals. It is used by MONICA to determin when the display is to be blanked.
LPEN*	Input	Light Pen. When this signal is asserted, the horizontal

		and vertical counters are latched into the light pen register.
PCLK	(C) Input	Input Pixel Clock. This clock has the pixel clock frequency in the low end system, and one half the pixel clock frequency in the high end system. This clock is used to drive the horizontal and vertical counters, and all pixel related timing. This input is driven to a full CMOS level(min VSS+.2v to VCC-.2v).
MCLK	(C) Input	Master Clock. This 57.2727Mhz clock is used to drive all of the bus related timing, and to derive CLK28 and BUSCLK. This input is driven to a full CMOS level(min VSS+.2v to VCC-.2v).
CLK28	(C) Output	Color Clock 28Mhz. This clock is derived by dividing MCLK by 2. This output must drive full the CMOS level(min VSS+.2v to VCC-.2v).
DMAL	Input	DMA Link. This is the serial link between MARY and ANDREA.
FREQCTRL0-2	Output	Frequency Control. This control bus is used to select a pixel clock frequency appropriate for the native or screen promotion mode currently selected.

The following table shows the ANDREA (1200) pin assignments.

Andrea Pin Assignments

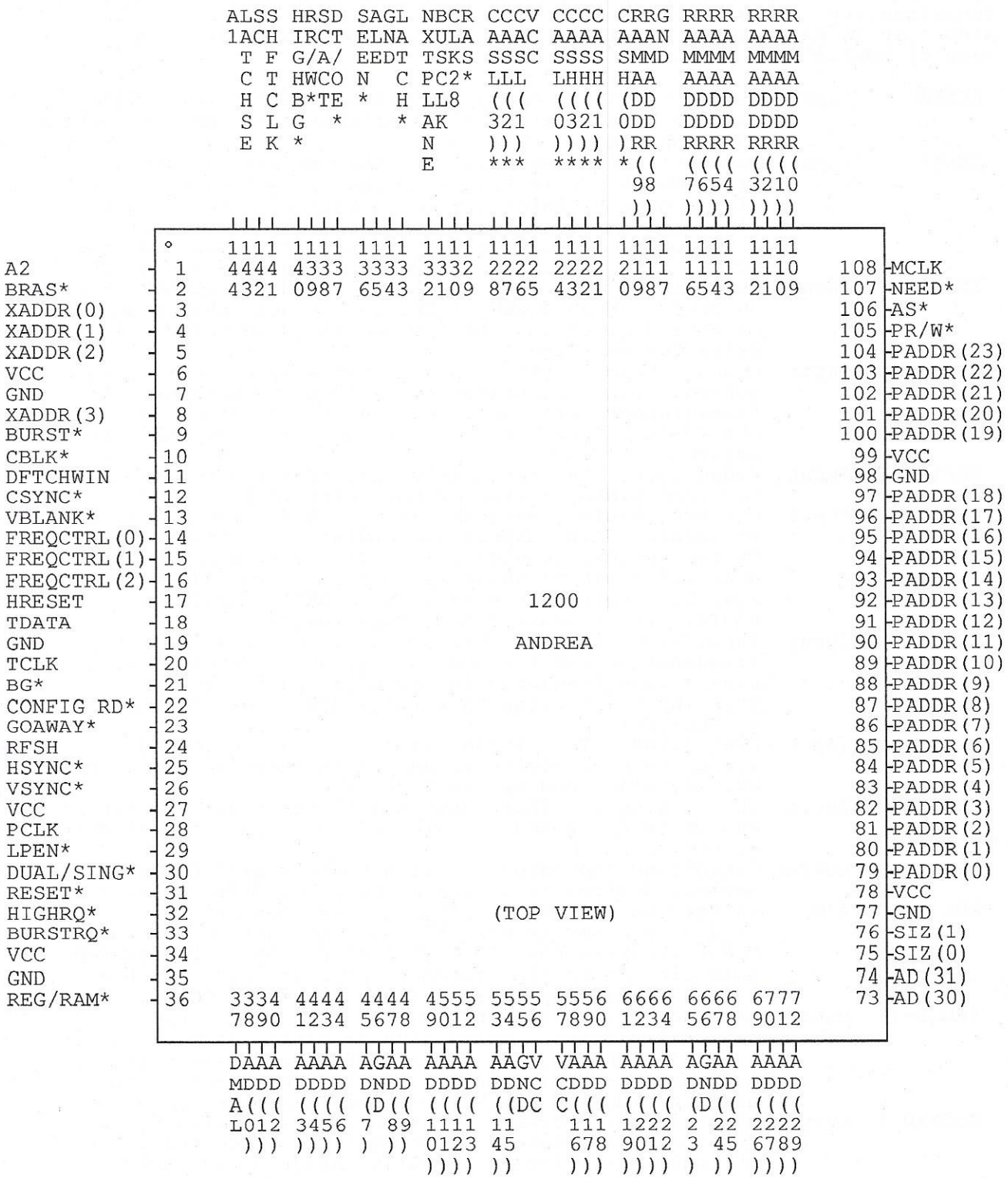
1	A2	73	AD(30)
2	BRAS*	74	AD(31)
3	XADDR(0)	75	SIZ(0)
4	XADDR(1)	76	SIZ(1)
5	XADDR(2)	77	GND
6	VCC	78	VCC
7	GND	79	PADDR(0)
8	XADDR(3)	80	PADDR(1)
9	BURST*	81	PADDR(2)
10	CBLK*	82	PADDR(3)
11	DFTCHWIN	83	PADDR(4)
12	CSYNC*	84	PADDR(5)
13	VBLANK*	85	PADDR(6)
14	FREQCTRL(0)	86	PADDR(7)
15	FREQCTRL(1)	87	PADDR(8)
16	FREQCTRL(2)	88	PADDR(9)
17	HRESET	89	PADDR(10)
18	TDATA	90	PADDR(11)
19	GND	91	PADDR(12)
20	TCLK	92	PADDR(13)
21	BG*	93	PADDR(14)
22	CONFIG_RD*	94	PADDR(15)
23	GOAWAY*	95	PADDR(16)
24	RFSH	96	PADDR(17)
25	HSYNC*	97	PADDR(18)
26	VSYNC*	98	GND
27	VCC	99	VCC
28	PCLK	100	PADDR(19)
29	LPEN*	101	PADDR(20)
30	DUAL/SING*	102	PADDR(21)
31	RESET*	103	PADDR(22)
32	HIGHRQ*	104	PADDR(23)
33	BURSTRQ*	105	PR/W*

34	VCC
35	GND
36	REG/RAM*
37	DMAL
38	AD(0)
39	AD(1)
40	AD(2)
41	AD(3)
42	AD(4)
43	AD(5)
44	AD(6)
45	AD(7)
46	GND
47	AD(8)
48	AD(9)
49	AD(10)
50	AD(11)
51	AD(12)
52	AD(13)
53	AD(14)
54	AD(15)
55	GND
56	VCC
57	VCC
58	AD(16)
59	AD(17)
60	AD(18)
61	AD(19)
62	AD(20)
63	AD(21)
64	AD(22)
65	AD(23)
66	GND
67	AD(24)
68	AD(25)
69	AD(26)
70	AD(27)
71	AD(28)
72	AD(29)
106	AS*
107	NEED*
108	MCLK
109	RAMADDR(0)
110	RAMADDR(1)
111	RAMADDR(2)
112	RAMADDR(3)
113	RAMADDR(4)
114	RAMADDR(5)
115	RAMADDR(6)
116	RAMADDR(7)
117	GND
118	RAMADDR(8)
119	RAMADDR(9)
120	CASH(0)
121	CASH(1)
122	CASH(2)
123	CASH(3)
124	CASL(0)
125	VCC
126	CASL(1)
127	CASL(2)
128	CASL(3)
129	RAS*
130	CLK28
131	BUSCLK
132	NXTPLANE
133	LATCH*
134	GND
135	ALE
136	SEEN*
137	DT/OE*
138	SCACT
139	RR/W*
140	HIGHBG*
141	SHFTCLK
142	SC
143	LATCHSE
144	A1

Power Legend:

VCC	digital +5 volts
GND	digital 0 volts

ANDREA (1200) pin diagram.



4.11.2 MONICA (1201). -

MONICA is the chip which does most of the old DENISE functionality. This chip has 131 signal pins and DAC biasing and power pins. It is packaged in a 144 pin QFP. Following is a description of each of MONICA's IO pins.

RESET*	Input	Reset. This is the system reset signal. This signal must be held asserted for a minimum of 10 msec at system power-up time.
CBLK*	Input	Composite Blanking. This is the composite blank signal from ANDREA. This signal is used to define those areas of the display which are to be blanked. This signal is also used in conjunction with the CONSPRT bits of SPRxCTLX to enable the display of sprites outside the normal display window defined by DIWSTRT and DIWSTOP.
CSYNC*	Input	Composite Sync. This input is needed to support Sync on Green. When SOGEN of BPLCON2 is set, this signal is passed directly into the DAC (no pipelining) to drive the sync level.
CAPEN	Input	Capture Enable. This is an optionally current based control signal generated by LINDA or by external framegrabber circuitry. MONICA latches video data from the VDATA bus each rising edge of PCLK that CAPEN is asserted.
VDATA0-31	Input	Video Data. The optionally current based video data bus from LINDA or framegrabber circuitry.
IEN	Input	Current Enable. When this pin is high, the logic levels on CAPEN, VDATA, HAMCI, and BLUTAI are differentiated by two current levels: 1.25mA min to 4.5mA max = logic one, 125uA min to 500uA max = logic zero. When IEN is low, the logic levels on CAPEN, VDATA, HAMCI, and BLUTAI are standard TTL voltage levels.
PCLK (C)	Input	Input Pixel Clock. This clock has the pixel clock frequency in the low end system, and one half the pixel clock frequency in the high end system. This input is driven to a full CMOS level(min VSS+.2v to VCC-.2v).
EVEN/ODD*	Input	Even / Odd. This signal is used in a high end dual chip system to tell MONICA if she is to process even or odd overlay plane and Sprite pixel data.
DUAL/SING*	Input	Dual / Single. This input signal tells MONICA whether she is in a low end, single chip or high end dual chip system.
ZD	Output	Background Indicator. This output is used by external Genlock devices to enable an external video source.
ZDX	I/O	Background Indicator. This pin is used primarily in dual systems to pass ZD information from the even MONICA chip to the odd MONICA chip. In single systems, this pin can be programmed to pass an 'audio enable' signal to genlock devices (see GAUD of BPLCON0).
HAMC0-1	Output	HAM Control. When MONICA is in HAM mode, these optionally current based bits originate from bitplanes 5 and 6. When MONICA is in HAM8 mode, these bits originate from bitplanes 1 and 2. This control bus is used in high end systems to allow HAM mode.
BLUTA0-7	Output	Bitplane LUT Address. This optionally current based bus is the bitplane address which is presented to the LUT when the priority circuits choose a playfield. This bus is normally used to permit the display of HAM mode in high end systems. It could also be used by other circuitry to enable special display modes.

HAMCI0-1	Input	Other MONICA HAM Control. This optionally current based control bus contains the same data as exists on HAMC0-1, but is originating from the other MONICA (high end system use only). These inputs should be driven low for low end system use.
BLUTAI0-7	Input	Other MONICA Bitplane LUT Address. This optionally current based control bus contains the same data as exists on BLUTA0-7, but is originating from the other MONICA (high end system use only). These inputs should be driven low for low end system use.
DB0-7	I/O	Digital Blue. In single systems, output or optionally tristated (see footnote). In dual systems, the even chip is normally output, and the odd chip is normally input. The even chip passes it's pixel data to the odd chip for multiplex into a serial pixel stream, followed by digital to analog conversion. The odd chips' analog blue can then be sent directly to the monitor. Tying PIXMUX high with a 10K pullup resistor forces both even and odd chips to be outputs in dual systems.
BLUE	Output	Analog Blue. Capable of driving a 37.5 ohm load. This signal should be disabled (see footnote) when the digital blue bus is programmed as an output.
DG0-7	I/O	Digital Green. In single systems, output or optionally tristated (see footnote). In dual systems, the even chip is normally output, and the odd chip is normally input. The even chip passes it's pixel data to the odd chip for multiplex into a serial pixel stream, followed by digital to analog conversion. The odd chips' analog blue can then be sent directly to the monitor. Tying PIXMUX high with a 10K pullup resistor forces both even and odd chips to be outputs in dual systems.
GREEN	Output	Analog Green. Capable of driving a 37.5 ohm load. When the digital green bus is programmed as an output, this pin is driven to a TTL voltage level indicating the state of the SOGEN bit in BPLCON2. This allows external circuits to support Sync on Green monitors.
DR0-7	I/O	Digital Red. In single systems, output or optionally tristated (see footnote). In dual systems, the even chip is normally output, and the odd chip is normally input. The even chip passes it's pixel data to the odd chip for multiplex into a serial pixel stream, followed by digital to analog conversion. The odd chips' analog blue can then be sent directly to the monitor. Tying PIXMUX high with a 10K pullup resistor forces both even and odd chips to be outputs in dual systems.
RED	Output	Analog Red. Capable of driving a 37.5 ohm load. This signal should be disabled (see footnote) when the digital red bus is programmed as an output.
A1	(C) Input	Address bit 1. This input indicates which part of the data bus (upper or lower word) contains valid data when old RGA registers are written. This input is driven to a full CMOS level(min VSS+.2v to VCC-.2v).
BUSCLK	(C) Input	Bus Clock. Used along with ALE to synchronize bus read and write transfers. This input is driven to a full CMOS level(min VSS+.2v to VCC-.2v).
LATCH*	Input	Data Latch. This signal is used to control latching of write data when MONICA's overlay plane and sprite registers are being burst mode written (for all other register writes, the rising edge of ALE is used to latch data). The rising edge of this signal is used

		to latch the data.
ALE	(C) Input	Address Latch Enable. RGA addresses are latched into the chip from the data bus(AD) with each falling edge of ALE. Register read or register write data is valid on the bus when ALE is low. Register write data is latched into the chip when ALE goes high. This input is driven to a full CMOS level(min VSS+.2v to VCC-.2v).
AD0-31	I/O	Register Address / Data bus. Multiplexed RGA / Data bus.
HRESET	Input	Horizontal Reset. This signal is generated by ANDREA and is used to reset MONICA's horizontal counter. This signal replaces the old STRHOR RGA address. Note that for software compatibility reasons, the same difference in horizontal count which exists in the old chip set has been retained in the new chip set.
PIXMUX	I/O	Tie low to use internal DAC. Tie high with 10K resistor to enable digital pixel bus as outputs (see footnote).
VREF	Input	1.235 volt reference used to establish DAC operating point.
IREF	Input	Current reference (typically 416uA to drive 37.5 ohm load) used to set total DAC current.
AVCC	Power	DAC VCC.
AGND	Power	DAC GND.
IVCC	Power	Current mode output pin supply. Tie to VCC.
IGND	Power	Current mode output pin supply. Tie to GND.

Footnote: Chip power consumption is expected to be significant when high resolution monitors are used in the system. Careful system design can limit chip power.

Single (low-end) systems design.
To use the internal color DAC's, disable the digital color bus by programming a low into 'DIGON' of the MONTEST register. (Note, it is not possible to disable the DR(7), DG(7), DB(7), or DB(0) pins in the low-end system.) Drive PIXMUX low.

To use external color DAC's, disable the internal DAC's by pulling PIXMUX high with a 10K resistor, tying IREF high, and by grounding the AVCC pin. Enable the digital color bus by programming a high into 'DIGON' of the MONTEST register.

Dual (high-end) systems design, using internal DAC's.
In this system configuration, the even chip DAC's are disabled and it's digital outputs are enabled to pass digital RGB data to the odd chip for conversion. The odd chip's digital bus become inputs, and it's DAC's are enabled to drive the monitor. (Note, in this configuration it is not possible to drive the odd chip's DR(7), DG(7), DB(7), and DB(0) pins to the outside world.) Tie both chip's PIXMUX pin low. Disable the even chip's DAC's by tying it's IREF pin high, and by grounding it's AVCC pin.

Dual (high-end) systems design, using external DAC's.
In this system configuration, the DAC's on both chips are disabled and the digital bus of both chips are enabled to pass digital RGB data to external circuitry for digital to analog conversion. The DAC's are disabled by tying the IREF pins high, and by grounding the AVCC pins. The digital busses are enabled by tying the PIXMUX pins high, each with their own 10K resistor. The PIXMUX pin of the odd chip becomes an output in this configuration, and overdrives the 10K pullup to provide a multplexer control signal to the external even pixel/

odd pixel parallel to serial convertor circuit. When the odd chip PIXMUX pin is high, the even chip digital pixel data should be displayed. When the odd chip PIXMUX pin is low, the odd chip digital pixel data should be displayed.

Values programmed into the 'DIGON' bit of the MONTEST register are ignored in dual systems.

HAM mode is fully supported in both of the dual system configurations described above.

Digital pixel data emerges from the 1201 chip two cycles sooner than when analog pixel data would appear. Systems which disable analog pixels and postprocess digital pixel data should provide two cycles of additional delay before display, or software should reprogram values in the sync control registers to accomodate the timing provided by the external circuitry.

The following table shows the MONICA (1201) pin assignments.

Monica (1201) Pin Assignments

1	GND	73	IGND
2	AD(23)	74	RESET*
3	AD(7)	75	IEN
4	AD(22)	76	EVEN/ODD*
5	AD(6)	77	DUAL/SING*
6	AD(21)	78	CSYNC*
7	AD(5)	79	CBLK*
8	AD(20)	80	HRESET
9	AD(4)	81	IVCC
10	VCC	82	HAMC1
11	AD(19)	83	HAMC0
12	AD(3)	84	BLUTA(7)
13	AD(18)	85	BLUTA(6)
14	AD(2)	86	BLUTA(5)
15	AD(17)	87	BLUTA(4)
16	AD(1)	88	BLUTA(3)
17	AD(16)	89	BLUTA(2)
18	AD(0)	90	BLUTA(1)
19	BLUTAI(0)	91	BLUTA(0)
20	BLUTAI(1)	92	VCC
21	BLUTAI(2)	93	CAPEN
22	BLUTAI(3)	94	VDATA(29)
23	BLUTAI(4)	95	VDATA(28)
24	BLUTAI(5)	96	VDATA(31)
25	BLUTAI(6)	97	VDATA(30)
26	BLUTAI(7)	98	GND
27	HAMCI0	99	VDATA(18)
28	HAMCI1	100	VDATA(19)
29	DB(0)	101	VDATA(20)
30	DB(1)	102	VDATA(21)
31	DB(2)	103	VDATA(22)
32	DB(3)	104	VDATA(23)
33	GND	105	VDATA(24)
34	DB(4)	106	VDATA(25)
35	DB(5)	107	VDATA(26)
36	DB(6)	108	VDATA(27)
37	VCC	109	VDATA(17)
38	DB(7)	110	VDATA(16)

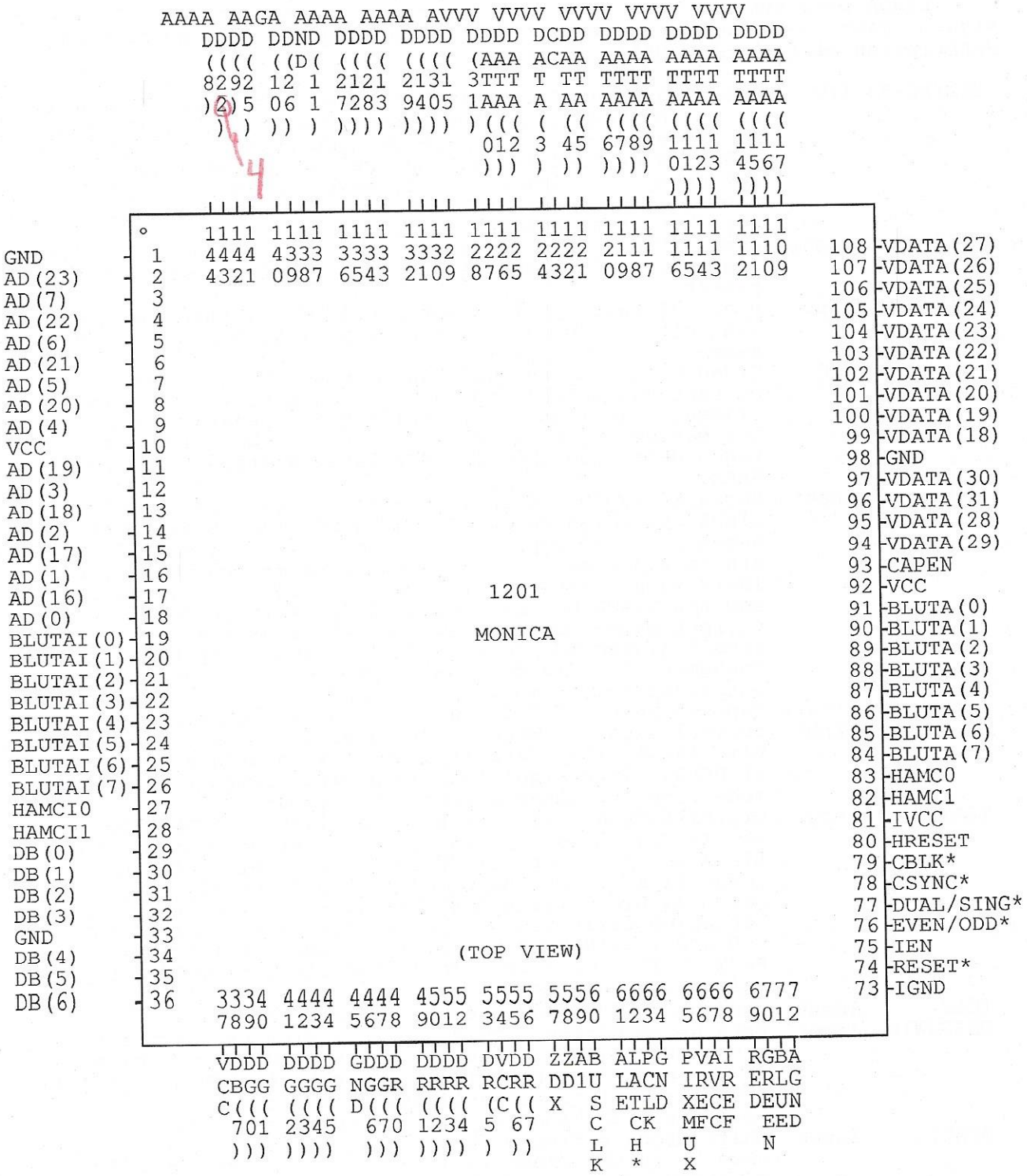
39	DG(0)
40	DG(1)
41	DG(2)
42	DG(3)
43	DG(4)
44	DG(5)
45	GND
46	DG(6)
47	DG(7)
48	DR(0)
49	DR(1)
50	DR(2)
51	DR(3)
52	DR(4)
53	DR(5)
54	VCC
55	DR(6)
56	DR(7)
57	ZDX
58	ZD
59	A1
60	BUSCLK
61	ALE
62	LATCH*
63	PCLK
64	GND
65	PIXMUX
66	VREF
67	AVCC
68	IREF
69	RED
70	GREEN
71	BLUE
72	AGND

111	VDATA(15)
112	VDATA(14)
113	VDATA(13)
114	VDATA(12)
115	VDATA(11)
116	VDATA(10)
117	VDATA(9)
118	VDATA(8)
119	VDATA(7)
120	VDATA(6)
121	VDATA(5)
122	VDATA(4)
123	VCC
124	VDATA(3)
125	VDATA(2)
126	VDATA(1)
127	VDATA(0)
128	AD(31)
129	AD(15)
130	AD(30)
131	AD(14)
132	AD(29)
133	AD(13)
134	AD(28)
135	AD(12)
136	AD(27)
137	AD(11)
138	GND
139	AD(26)
140	AD(10)
141	AD(25)
142	AD(9)
143	AD(24)
144	AD(8)

Power legend:

VCC	digital	+5 volts
GND	digital	0 volts
AVCC	analog	+5 volts
AGND	analog	0 volts
IVCC	digital	+5 volts (powers current mode outputs)
IGND	digital	0 volts (powers current mode outputs)

MONICA (1201) pin diagram.



4.11.3 LINDA (1202). -

LINDA provides the Line Data buffer function. The chip has 114 signal pins and is packaged in a 144 pin QFP. Following is a description of each of LINDA's IO pins.

GDATA0-63	I/O	Graphics Data Bus. This bus comes from the serial port of system VRAM in a VRAM system, or from the data port of system DRAM in a Fast Page Mode system. Bits 31 thru 0 should be tied low in the low end system as they are not used. In normal display mode (TDATA 'DIR' bit = forward), this bus is an input. In framegrabber mode, this bus is an output.
EVEN/ODD*	Input	Even / Odd. This signal is used in a high end dual chip system to tell LINDA if she is to buffer even or odd pixels.
DUAL/SING*	Input	Dual / Single. This input signal tells LINDA whether she is in a low end, single chip or high end dual chip system.
VDATA0-31	I/O	Video Data. In normal display mode, this optionally current based bus outputs video data to MONICA. In framegrabber mode (TDATA 'DIR' bit = reverse), this bus becomes a TTL level input bus. LINDA captures video data from this bus for later storage into system VRAM.
IEN	Input	Current Enable. When this pin is high, the logic levels LINDA generates on CAPEN and the VDATA bus are current based and differentiated by two current levels: 1.25mA min to 4.5mA max = logic one, 125uA min to 500uA max = logic zero. When IEN is low, the logic levels on CAPEN and the VDATA bus are standard TTL voltage levels. Current mode should be enabled whenever possible to reduce system EMF noise, and to reduce chip power consumption. IEN must be driven low (voltage mode) to use framegrabber mode.
CAPEN	Tristate Output	Capture Enable. This is an optionally current based control signal. When this signal is a logic 1, MONICA will latch video data from LINDA with each rising edge of PCLK. This signal drives during normal display mode, and tristates during framegrabber mode.
TDATA	Input	Graphics Type. This is a serial link from ANDREA. The following information is transferred from ANDREA to LINDA on this link: HIRES/LORES, five bits of BPU, three bits of type data(indicating graphics type), SWITCH(a bit telling LINDA to switch RAM banks), DIR(a bit which tells LINDA data is flowing to or from RAM), RP0 and 1 (LINEPRT0 and 1 of BPLCON3), PF1H0, and PF2H0. This data is transferred after HTOTAL, but before DFTCHWIN is asserted.
TCLK	Input	Used to shift data into the TDATA port.
DFTCHWIN	Input	Data Fetch Window. This signal indicates the start of a new scan line. It is primarily used to determine when to begin sending new line video data to MONICA. Due to timing restrictions in LINDA, this signal is only permitted to change on falling edges of PCLK.
SCACT	Input	Shift Clock Active. This signal is asserted by ANDREA when SC to the VRAMs is active. It is used by LINDA to determine when to latch graphics data into the chip. When graphics data is being fetched from Fast Page Mode RAM rather than VRAM, SCACT will be asserted when CAS* is being driven to satisfy graphics transfers.
SHFTCLK(C)	Input	This is a free running clock which usually has the

		duty cycle, phase, and frequency of SC. The frequency of this clock can change on the fly to match CAS* signal timing. This clock is used to drive the input side of LINDA. It must be driven to full CMOS voltage levels (min VSS+.2v to VCC-.2v).
A1-2	Input	Bitplane Starting Address bits 1 and 2. The old AMIGA chip set allowed bitplanes to start on any word boundary. A1 and A2 tell LINDA the word address so that she can align GDATA input data appropriately.
NXTPLANE	Input	Next Plane. This signal tells LINDA that she should begin inputting data for the next bitplane.
RESET*	Input	Reset. This is the system reset signal. This signal must be held asserted for a minimum of 10 msec at system power-up time.
PCLK (C)	Input	Input Pixel Clock. This clock has the pixel clock frequency in the low end system, and one half the pixel clock frequency in the high end system. This clock is used to drive the output side of LINDA. This input is driven to a full CMOS level(min VSS+.2v to VCC-.2v).
DIR	Output	Direction. This control signal tells external framegrabber circuitry when the system is in capture mode, and therefore when the circuitry system should be driving pixel data onto the VDATA bus.
OT(2:0)	Output	Pixel Type. This control bus tells external framegrabber circuitry what kind of pixel data (Hybrid, Chunky, or Half-Chunky) the system is expecting.
OHI/LO*	Output	Hi-res / Low res. This signal tells external framegrabber circuitry whether the expected pixel data is high resolution or low resolution.
DFW	Output	Display Fetch Window. In framegrabber mode, this signal specifies the region of the display line when MONICA is receiving display data from the VDATA bus. It is used by external framegrabber circuitry to synchronize to the active portion of the scan line, and to enable VDATA bus drivers.

The following table shows the LINDA (1202) pin assignments.

Linda Pin Assignments

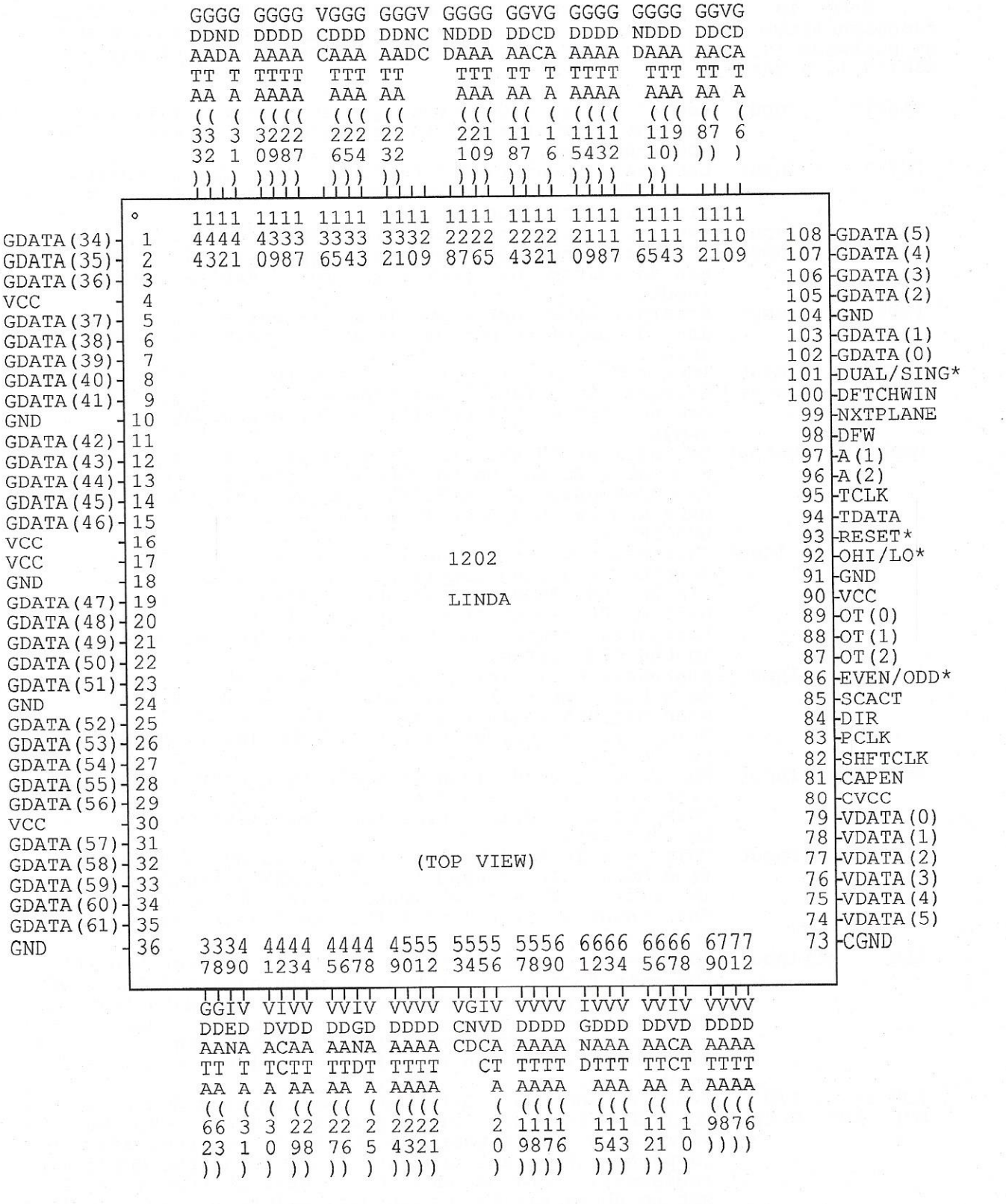
1	GDATA(34)	73	CGND
2	GDATA(35)	74	VDATA(5)
3	GDATA(36)	75	VDATA(4)
4	VCC	76	VDATA(3)
5	GDATA(37)	77	VDATA(2)
6	GDATA(38)	78	VDATA(1)
7	GDATA(39)	79	VDATA(0)
8	GDATA(40)	80	CVCC
9	GDATA(41)	81	CAPEN
10	GND	82	SHFTCLK
11	GDATA(42)	83	PCLK
12	GDATA(43)	84	DIR
13	GDATA(44)	85	SCACT
14	GDATA(45)	86	EVEN/ODD*
15	GDATA(46)	87	OT(2)
16	VCC	88	OT(1)
17	VCC	89	OT(0)
18	GND	90	VCC
19	GDATA(47)	91	GND
20	GDATA(48)	92	OHI/LO*

21	GDATA(49)
22	GDATA(50)
23	GDATA(51)
24	GND
25	GDATA(52)
26	GDATA(53)
27	GDATA(54)
28	GDATA(55)
29	GDATA(56)
30	VCC
31	GDATA(57)
32	GDATA(58)
33	GDATA(59)
34	GDATA(60)
35	GDATA(61)
36	GND
37	GDATA(62)
38	GDATA(63)
39	IEN
40	VDATA(31)
41	VDATA(30)
42	IVCC
43	VDATA(29)
44	VDATA(28)
45	VDATA(27)
46	VDATA(26)
47	IGND
48	VDATA(25)
49	VDATA(24)
50	VDATA(23)
51	VDATA(22)
52	VDATA(21)
53	VCC
54	GND
55	IVCC
56	VDATA(20)
57	VDATA(19)
58	VDATA(18)
59	VDATA(17)
60	VDATA(16)
61	IGND
62	VDATA(15)
63	VDATA(14)
64	VDATA(13)
65	VDATA(12)
66	VDATA(11)
67	IVCC
68	VDATA(10)
69	VDATA(9)
70	VDATA(8)
71	VDATA(7)
72	VDATA(6)
93	RESET*
94	TDATA
95	TCLK
96	A(2)
97	A(1)
98	DFW
99	NXTPLANE
100	DFTCHWIN
101	DUAL/SING*
102	GDATA(0)
103	GDATA(1)
104	GND
105	GDATA(2)
106	GDATA(3)
107	GDATA(4)
108	GDATA(5)
109	GDATA(6)
110	VCC
111	GDATA(7)
112	GDATA(8)
113	GDATA(9)
114	GDATA(10)
115	GDATA(11)
116	GND
117	GDATA(12)
118	GDATA(13)
119	GDATA(14)
120	GDATA(15)
121	GDATA(16)
122	VCC
123	GDATA(17)
124	GDATA(18)
125	GDATA(19)
126	GDATA(20)
127	GDATA(21)
128	GND
129	VCC
130	GND
131	GDATA(22)
132	GDATA(23)
133	GDATA(24)
134	GDATA(25)
135	GDATA(26)
136	VCC
137	GDATA(27)
138	GDATA(28)
139	GDATA(29)
140	GDATA(30)
141	GDATA(31)
142	GND
143	GDATA(32)
144	GDATA(33)

Power Legend:

VCC	digital	+5 volts	
GND	digital	0 volts	
CVCC	digital	+5 volts	(powers internal clock drivers)
CGND	digital	0 volts	(powers internal clock drivers)
IVCC	digital	+5 volts	(powers current mode outputs)
IGND	digital	0 volts	(powers current mode outputs)

LINDA (1202) pin diagram.



4.11.4 MARY (1203R0). -

MARY is the chip which performs most of the old PAULA functionality. MARY contains 82 signal and analog power pins, and will be packaged in an 144 pin PQFP. Following is a description of each of MARY's IO signals.

RESET*	Input	Reset. This is the system reset signal. This signal must be held asserted for a minimum of 10 msec at system power-up time.
INT7*	Input	External, re-entrant interrupt input from expansion bus. There are no custom chip provisions for disabling this input.
INT6*	Input	External interrupt input from expansion bus.
INT5*	Input	External interrupt input from expansion bus. There are no custom chip provisions for disabling this input.
INT4*	Input	External interrupt input from expansion bus. There are no custom chip provisions for disabling this input.
INT2*	Input	Interrupt input from 8520 port chips.
INT1*	Input	External interrupt input from expansion bus. There are no custom chip provisions for disabling this input.
DMAL	Output	Serial link to ANDREA. This serial port conveys DMA request information to ANDREA. Data is transmitted synchronously with BUSCLK. Data is shifted out of MARY and latched into ANDREA on each rising edge of BUSCLK.
VBLANK*	Input	This negative true signal is used to generate the vertical blanking interrupt. It replaces the old STREQU and STRVBL RGA strobes. When the system is in NTSC or PAL mode, this signal will occur at the same horizontal count that the strobe addresses were written in the old system.
A1	(C) Input	Address bit 1. This input indicates which part of the data bus (upper or lower word) contains valid data when old RGA registers are written. This input is driven to a full CMOS level(min VSS+.2v to VCC-.2v).
BUSCLK	(C) Input	Bus Clock. Used along with ALE to synchronize bus read and write transfers. This input is driven to a full CMOS level(min VSS+.2v to VCC-.2v).
CLK28	(C) Input	This is a 28.63636Mhz clock which is derived by ANDREA from MCLK. It is used to drive MARY's new disk interface. This clock must be synchronous to BUSCLK. This input is driven to a full CMOS level(min VSS+.2v to VCC-.2v).
ALE	(C) Input	Address Latch Enable. RGA addresses are latched into the chip from the data bus(AD) with each falling edge of ALE. Register read or register write data is valid on the bus when ALE is low. Register write data is latched into MARY 35nsec after ALE goes high. This input is driven to a full CMOS level(min VSS+.2v to VCC-.2v).
AD0-31	I/O	Register Address / Data bus. Multiplexed RGA / Data bus.
MONIDEN*	Output	MONITORID Enable. This signal is simply an RGA decode which goes active when the MONITORID register address is present(When the processor is reading the MONITORID register). MARY decodes this signal so that it will not be necessary to design external hardware to perform

		this function. Mary asserts this signal while ALE is low.
RxD	Input	Serial port receiver input.
TxD	Output	Serial port transmitter output.
RxD2	Input	Second Serial port receiver input.
TxD2	Output	Second Serial port transmitter output.
LFORWARD*	Input	Left joystick forward signal. (M0V) Also used for mouse control.
LBACK*	Input	Left joystick back signal. (M0H) Also used for mouse control.
LLEFT*	Input	Left joystick left signal. (M0VQ) Also used for mouse control.
LRIGHT*	Input	Left joystick right signal. (M0HQ) Also used for mouse control.
RFORWARD*	Input	Right joystick forward signal. (M1V) Also used for mouse control. Signal is ignored in audio sampling mode.
RBACK*	Input	Right joystick back signal. (M1H) Also used for mouse control. Signal is ignored in audio sampling mode.
RLEFT*	Input	Right joystick left signal. (M1VQ) Also used for mouse control. Signal is ignored in audio sampling mode.
RRIGHT*	I/O	Right joystick right signal. (M1HQ) Also used for mouse control. In audio sampling mode, this pin becomes
DKRD*	Input	Disk Read Data. Raw serial read data normally from a floppy disk drive or a CD player.
DKWD*	Output	Disk Write Data. Serial write data to the floppy disk drive.
DKWE	Output	Disk Write Enable. Enables write data to disk.
PLLRST*	Input	For test only, should be tied high in the system. Resets the Disk Controller Phase Locked Loop circuits.
POTLX	I/O	Left Pot X. Pot function is disabled when chip is in audio sampling mode.
POTLY	I/O	Left Pot Y. Pot function is disabled when chip is in audio sampling mode.
POTRX	I/O	Right Pot X. Also accepts an audio waveform when the chip is in audio sampling mode.
POTRY	I/O	Right Pot Y. Also accepts an audio waveform when the chip is in audio sampling mode.
AUDDIGL	Output	Digital audio output for left channel. Synchronous serial output. Data is shifted out on each rising edge of BUSCLK.
AUDDIGR	Output	Digital audio output for right channel. Synchronous serial output. Data is shifted out with each rising edge of BUSCLK.
AUDDIGLD	Output	Load pulse for left and right digital outputs. The rising edge of this signal causes digital audio data to be latched into the parallel portion of an external serial to parallel converter.
AUDL	Output	Left Analog Audio Channel.
IDUMPL	Output	'Inverse' Left Analog Audio Channel current. Should be tied to DACGND in the system.
AUDR	Output	Right Analog Audio Channel.
IDUMPR	Output	'Inverse' Right Analog Audio Channel current. Should be tied to DACGND in the system.
AUDREFL	Input	Voltage reference for the left analog audio channel. Mary's internal reference generator can be over-driven by sourcing this pin(if need be).
AUDREFR	Input	Voltage reference for the right analog audio channel. Mary's internal reference generator can be over-driven by sourcing this pin(if need be).

IPL*0-2	OD	Open Drain interrupt lines to the processor.
AGND	Power	Analog ground for pot circuitry.
APWR	Power	Analog supply for pot circuitry.
DACGND	Power	Analog ground for audio DAC circuitry.
DACPWR	Power	Analog supply for audio DAC circuitry.

The following table shows the MARY (1203R0) pin assignments.

Mary (1203R0) Pin Assignments

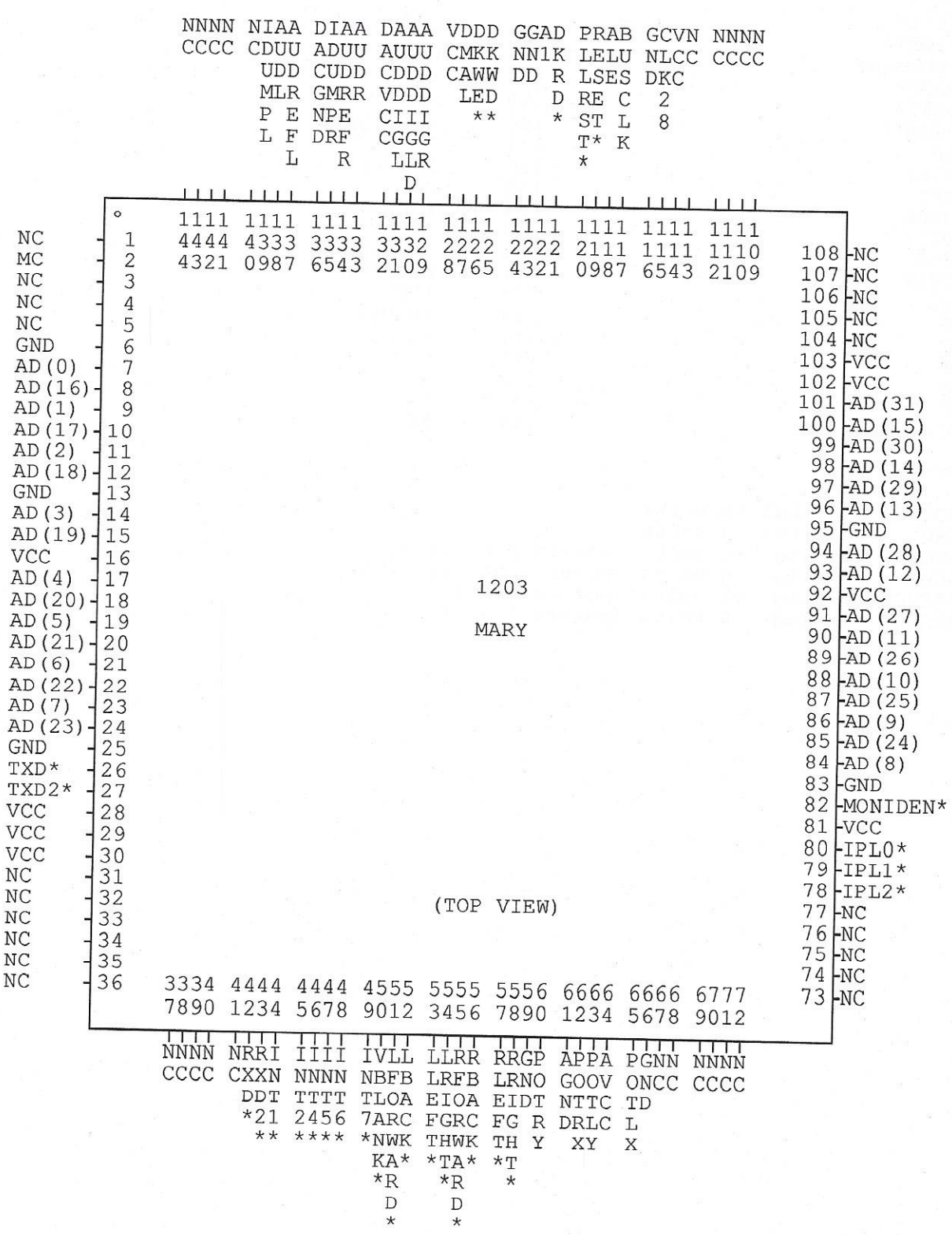
1	NC	73	NC
2	NC	74	NC
3	NC	75	NC
4	NC	76	NC
5	NC	77	NC
6	GND	78	IPL2*
7	AD(0)	79	IPL1*
8	AD(16)	80	IPL0*
9	AD(1)	81	VCC
10	AD(17)	82	MONIDEN*
11	AD(2)	83	GND
12	AD(18)	84	AD(8)
13	GND	85	AD(24)
14	AD(3)	86	AD(9)
15	AD(19)	87	AD(25)
16	VCC	88	AD(10)
17	AD(4)	89	AD(26)
18	AD(20)	90	AD(11)
19	AD(5)	91	AD(27)
20	AD(21)	92	VCC
21	AD(6)	93	AD(12)
22	AD(22)	94	AD(28)
23	AD(7)	95	GND
24	AD(23)	96	AD(13)
25	GND	97	AD(29)
26	TXD*	98	AD(14)
27	TXD2*	99	AD(30)
28	VCC	100	AD(15)
29	VCC	101	AD(31)
30	VCC	102	VCC
31	NC	103	VCC
32	NC	104	NC
33	NC	105	NC
34	NC	106	NC
35	NC	107	NC
36	NC	108	NC
37	NC	109	NC
38	NC	110	NC
39	NC	111	NC
40	NC	112	NC
41	NC	113	NC
42	RXD*	114	VCC
43	RXD2*	115	CLK28
44	INT1*	116	GND
45	INT2*	117	BUSCLK
46	INT4*	118	ALE
47	INT5*	119	RESET*
48	INT6*	120	PLLRST*
49	INT7*	121	DKRD*
50	VBLANK*	122	A1

51	LFORWARD*	123	GND
52	LBACK*	124	GND
53	LLEFT*	125	DKWD*
54	LRIGHT*	126	DKWE*
55	RFORWARD*	127	DMAL
56	RBACK*	128	VCC
57	RLEFT*	129	AUDDIGR
58	RRIGHT*	130	AUDDIGLD
59	GND	131	AUDDIGL
60	POTRY	132	DACVCC
61	AGND	133	AUDR
62	POTRX	134	AUDREFR
63	POTLY	135	IDUMPR
64	AVCC	136	DACGND
65	POTLX	137	AUDREFL
66	GND	138	AUDL
67	NC	139	IDUMPL
68	NC	140	NC
69	NC	141	NC
70	NC	142	NC
71	NC	143	NC
72	NC	144	NC

Power Legend:

VCC	digital	+5 volts	
GND	digital	0 volts	
AVCC	analog	+5 volts	(powers pot circuits)
AGND	analog	0 volts	(powers pot circuits)
DACVCC	analog	+5 volts	(powers DAC's)
DACGND	analog	0 volts	(powers DAC's)

MARY (1203R0) pin diagram.



5 ANDREA.

This chapter describes new features which have been added to the address generator chip. It also discusses major differences in operation between this and older versions of the chip.

5.1 Hard Monitor Control Stops.

The old AMIGA chip set had a set of hardware 'stops' which controlled most aspects of display timing. These include horizontal sync, horizontal blanking, vertical sync, vertical blanking, and others. The second generation AMIGA chips included these hardware stops, but also incorporated programmable registers for this functionality. Using the programmable registers, higher resolution displays (VGA) can be driven. In the new chip set, most of the original hardware stops mentioned above have been eliminated. Display timings must now be programmed when the system is powered up. As a result, no video activity will occur until these registers have been set and the BEAMEN bit of BEAMCON0 is set. While BEAMEN is clear, sync signal generation is disabled and blanking is enabled. Inhibiting video activity in such a way until system software has set up appropriate display characteristics prevents damage to a monitor due to improperly timed sync signals. It should be noted that in order to maintain software compatibility, the display control registers must be programmed with the same values that were hardcoded into the original AMIGA chips. For reasons which will become obvious in the Low Resolution Screen Promotion section, certain guidelines must be followed when programming these registers to drive high resolution monitors.

5.2 Long Line, Short Line Eliminated.

The new horizontal counter precision permits elimination of the long line/short line display attribute (see section 4.1.3). Long line/short line displays were necessary to meet NTSC timing specs. If 227 clocks were always used to time out scan lines, then a horizontal frequency of 15768.9 hz would be obtained. If 228 clocks were always used, then a horizontal frequency of 15699.8 hz would be obtained. By alternating these counts from line to line, the old chip set obtained an average horizontal frequency of 15734.3 hz which is the NTSC spec. By incorporating two extra bits of horizontal counter resolution and counting 910 pixels, the NTSC horizontal frequency is achieved painlessly. This eliminates the need for all of the delay elements presently built into the old chip sets' sync and blanking signals to accommodate short line timing. With one exception, elimination of Long Line Short Line displays will not effect software compatibility. The exception is old coprocessor code which has a wait statement looking for the old horizontal count of 227. This count will not happen in the new system. For compatibility, the LOL bit still exists in the VPOS register. This bit is set to an appropriate state at the beginning of a long field/short field frame, and is toggled every scan line there after.

5.3 Interlaced NTSC And PAL Display Modes.

The new chip set is capable of generating spec PAL and NTSC

displays. To use one of these display modes, either the NTSC or PAL bit in BEAMCON0 is set. In addition to setting one of these bits, the programmable line control registers must be programmed with identical values as existed in the fixed decoders of the old chip set. Failure to reproduce these values exactly may cause misplaced burst and equalization pulses with respect to hsync, etc. (Note that the new chips actually still contain three hardwired decodes.) Failure to set one of the NTSC or PAL bits will disable compatible genlock processing.

The following table shows values used in old systems (based on the old counter precision), and specifies values to use in the new system when programming interlaced and non-interlaced NTSC and PAL displays. (Certain registers exist in the new system which replace old long line/short line circuits of the old system. These registers must be programmed as specified.) These values provide maximum compatibility with software written for the old Amiga systems.

	NTSC		PAL		
Parameter	AGNUS	ANDREA	AGNUS	ANDREA	
BRSTSTART	27	09C	27	09C	(Burst start)
BRSTSTOP	30	0C0	30	0C0	(Burst stop)
EQU1STOP	1C	070	1C	070	
EQU2STOP	8D~	236	8D	236	
HSSTRTX	12	048	12	048	
HSSTOPX	23	08C	23	08C	
HBSTRTX	08	020	08	020	
HBSTOPX	2B~	0AE	2B~	0AE	
HCENTERX	71~	1C6	71~	1C6	
HTOTALX	E3	38D	E3	38D	
SER1STOP		1D2		1D2	
SER2STOP		00C		00C	
CLCNTR		1CC		1CC	
MLSYNC		214		214	
VBSTRT		000		000	
VBSTOP		014		019	
VSSTRT		003		8002	(line 2 and 1/2)
VSSTOP		006		8005	(line 5 and 1/2)
VTOTAL		106	(525 half lines per field)	138	(625 half lines per field)
VEQUSTART		000		000	(hardwire, no reg.)
VEQUSTOP		009	(line 9)	8007	(line 7 and 1/2)

Note: set the NTSC or PAL bit in BEAMCON0.

~ These events actually occurred the half count after that value.

Vertical timings are handled differently in the new chip set when compared to the old. A substantial logic reduction is realizable by counting half lines when generating NTSC or PAL interlaced displays. This new method is contrary to the old method of displaying 262 lines one field and 263 lines the next field. The new chip set displays 262.5 lines each field in NTSC MODE. This means that LOF(long frame/short frame) losses it's meaning in the new chip set. The LOF bit still exists in VPOSR, however it is now more appropriately referred to as 'even field/odd field'. From a software point of view, the new LOF bit behaves the same as it did in the old chip set thus retaining software compatibility. This is because the secondary vertical counter has been designed to count the same way as the

vertical counter did in the old chip set, and the secondary counter is what software and the coprocessor sees. The primary vertical counter is used to count the half lines actually used in sync signal generation.

In NTSC interlaced mode, ANDREA's secondary vertical counter counts 262 lines during short fields and 263 during long fields. Similarly, in PAL interlaced mode, the secondary counter alternates between 312 and 313 lines.

This mechanism is not reflected on ANDREA's sync outputs since they are derived only from the vertical primary counter. The primary counter counts half-lines at all times, under all conditions, allowing generation of true NTSC or PAL timing. In other words, ANDREA will do 262.5 lines on every field (525 lines/frame), when in NTSC mode; in PAL mode, 312.5 lines on every field (625 lines/frame).

In non-interlaced modes, ANDREA's secondary counter will count long fields at all times, regardless of the long-frame bit (LOF) state. Hence, the secondary counter will do 213 lines/frame in NTSC and 313 lines/frame in PAL mode. At the same time, the primary counter is forced to count 263 lines (or 526 half lines) in NTSC; 313 lines (or 626 half lines) in PAL. That is, in non-interlaced modes the primary counter will count up to count VTOTAL+1. This is done to prevent displays of 262.5 or 312.5 lines (the additional half-line is meaningless in a non-interlaced display).

To support the count of half lines, a new bit of precision has been added to the vertical counter, the vhalf bit. When LACE is set, this counter bit is enabled as the least significant vertical counter bit. In this case, HTOTAL and HCENTER are used to toggle the vertical counter. V0 becomes the second least significant counter bit. All comparisons (ex. SPRITE position comparitors) are still made using the most significant bits of the counter v10-v0, so it is not necessary to reprogram comparison registers when interlaced mode is enabled. When LACE is cleared, the new vhalf bit of the vertical counter is disabled and v0 becomes the least significant vertical counter bit.

5.4 GENLOCK Mode.

Genlock mode of the new chip set is designed to operate in the same manner as in the old chip set. This section discusses the genlock mode under different display environments, and especially NTSC and PAL.

As has been discussed in other sections of this document, displays with alternate long and short lines are no longer generated. In order to support old genlock devices, the new horizontal counter must have the capacity to count out two lines before freezing at count "00". The LOL bit has been promoted in the new system to support this requirement; the bit is now a flip flop which toggles at the end of every line. The bit also provides support for old software which tests for long or short lines.

In the description that follows, a "count" refers to the old-chip set count, which does not include two new horizontal counter LSB bits added to the new chip set. Also, LOL indicates "long line" when set, and "short line" when cleared.

5.4.1 Horizontal Counter. -

The horizontal counter obeys the following rules when NTSC mode is enabled (This behavior is identical to the behavior exhibited by the old system):

i. If the external HSY* reset is NOT applied at the end of a "short line", the next line will be a "long line", as normal.

   If no HSY* is applied at the end of a "long line", the horizontal counter will roll-over and is held reset at count "00". The counter will resume counting a short line once the external HSY* pulse is applied.

ii. If the external HSY* reset is applied at the end of a "short line", the following line is forced to be a another "short line" starting at count "00".

iii. If the counter reaches the end of a "long line" when the external HSY* is asserted, the next line is a "short line" and will start at count "01" (as opposed to count "00"). Otherwise, if no reset occurs the counter will roll-over to count "00" and stop.

In all other display modes, including PAL mode, the external HSYNC reset is implemented in a simpler fashion. The counter operates with "short lines" at all times ("long lines" are disabled) and responds as follows:

i. If the external HSY* is not applied at the end of a line, the horizontal counter will roll-over to the beginning of the next line and is held at count "00", until the next HSY* pulse occurs.

ii. If the external HSY* is applied when the counter reaches the end of line, the next line will start at count "01", skipping count "00".

5.4.2 Vertical Counter. -

Both the new and old chip sets use the same rules to control reset of the vertical counter in Genlock mode. (Like the LOL bit used to control horizontal counter reset, the LOF bit is used to give an indication of "long frame/short frame".) The vertical counter operates in the same manner for both NTSC and PAL modes. After ANDREA is configured for Genlock mode and will accept external syncs (ERSY bit in BPLCON0), the vertical counter responds as follows:

INTERLACE MODE (LACE bit is set in BPLCON0):

i. If an external VSY* pulse is applied, the vertical counter will be reset to count "000" (first line) and the "long field" condition is set at the beginning of the next horizontal line, as the horizontal counter goes from count "01" to "02". (This corresponds to a transition from count "07" to "08" in the new chip set horizontal counter resolution.)

ii. If the external VSY* pulse is not asserted during the end of a long field, the vertical counter will roll-over and start a short field sequence.

NON-INTERLACE MODE (LACE bit deasserted):

In Non-interlace mode the vertical counter is set to operate with the value which has been programmed into the LOF bit at all times. Hence, a frame will contain two fields of 263 lines (NTSC mode), or 313 lines (PAL mode) when the LOF bit has been preset by software or set that way by previous hardware activities. (These counts are programmed by system software in the new system.) In this mode, whenever an external VSY* pulse is applied the counter is reset to line count "000" (first line) at the beginning of the next horizontal line.

In a display mode other than NTSC or PAL, the vertical counter will reset at the beginning of horizontal count "00" while the external VSY* pulse is asserted, whether or not interlace mode is enabled. Also, the LOF bit is ignored since in the new chip set both odd and even fields have the same time duration, as determined by the value loaded in the VTOTAL register.

5.4.3 External GENLOCK Signals. -

The horizontal sync pulse, HSYNC should be asserted with exact periodicity during or just before ANDREA's end of line transition as indicated by HTOTAL. The width of the horizontal pulse is not critical, since the horizontal genlock logic is negative edge triggered by HSYNC, however the pulse should be less than one-half a line length in duration. If the horizontal counter is held reset because the chip is in NTSC mode and because of non-occurrence of an HSYNC pulse, the genlock logic will detect and react immediately to the first edge of the next HSYNC pulse in order to release the counter from reset (see horizontal counter reset rules above).

Under any display mode, the width of the vertical sync pulse, VSYNC should be less than one line duration and should be asserted during the beginning of a horizontal line, at which time the vertical genlock logic is triggered and the "first line" is started.

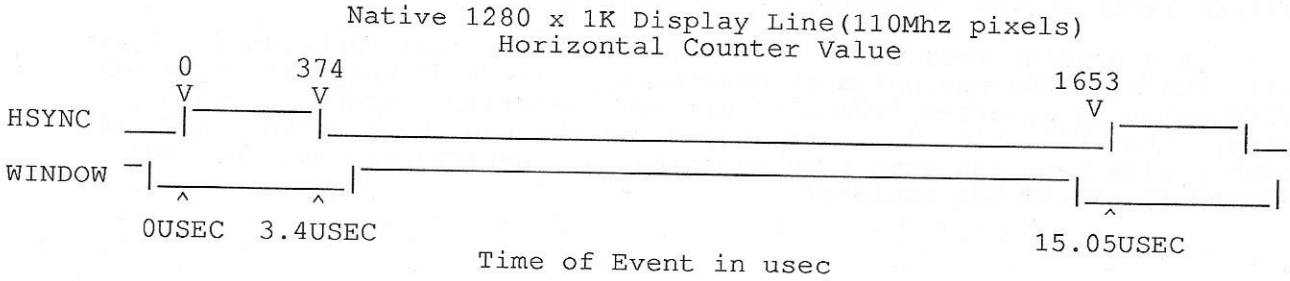
The XCLK signal provided should be twice the frequency of the high resolution pixel for the monitor being driven. This allows a square wave to be generated prior to application to the chips; XCLK is divided by two by the system clock generation circuitry.

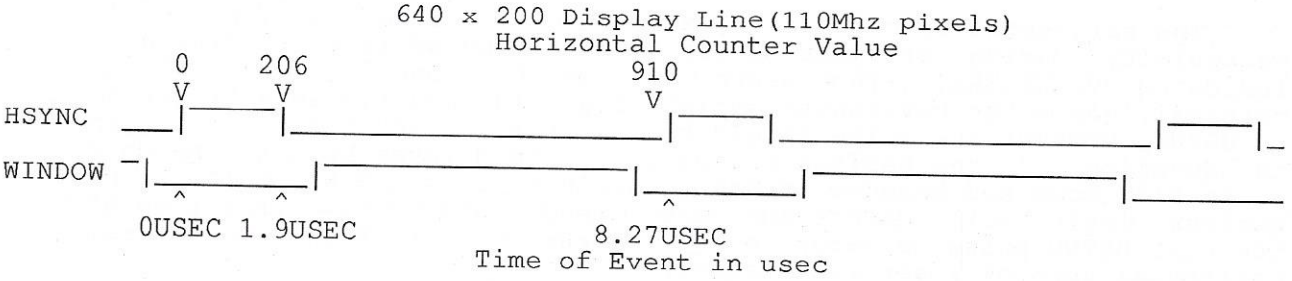
5.5 Low Resolution Screen Promotion.

One goal of the new chip set is to permit the display of low resolution screens on high resolution monitors. This option allows a user to execute software written for a low resolution graphics system without requiring he power down his system in order to plug in another monitor. Furthermore, it is desirable to allow the user to pull down his highres screen so that he can operate on the lowres screen behind. This section describes the screen promotion mode in detail.

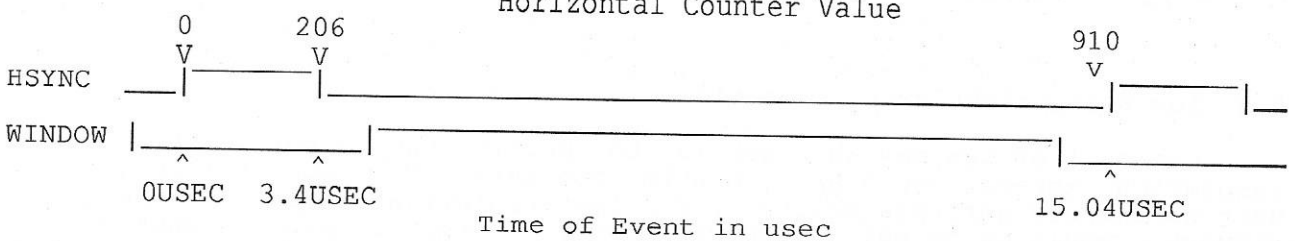
5.5.1 Basic Scheme. -

A monitor of given resolution will always be driven with a constant horizontal frequency, and will always display the same number of lines between vertical syncs, even when a low resolution screen promotion mode is running. Consider the following example. A 1280 x 1K monitor running in native mode has 1654 pixels per line, and 1084 lines per frame. With a pixel frequency of 110Mhz, the horizontal line rate is 66.505Khz, and the frame rate is 61.35hz. An image from a 640 x 200 system is to be displayed on this monitor. In it's native environment, this image contains 910 pixels per line and 262 lines per frame. The following figure shows the waveforms for the horizontal sync signal and display window of the native screen. Following that are waveforms showing what a 640 x 200 line would look like if it were displayed with the native 110Mhz pixel clock.





The figure shows that if one attempted to display a 640 x 200 screen using a 110Mhz native pixel clock, then horizontal sync pulses would be generated at almost twice the needed frequency. The pixel clock is therefore slowed down by such a percentage that the period between successive sync signals remains constant. The following figure shows what the promoted 640 x 200 line looks like when the pixel frequency is reduced from 110 Mhz to 60.5Mhz.



By slowing down the pixel clock, we are able to stretch lowres lines out far enough to cover the entire screen. It is possible to set up all horizontal monitor control parameters (example, horizontal sync (HSSTRT, HSTOP), horizontal blanking (HBSTRT, HBSTOP), and display

window (DIWSTRT, DIWSTOP)) such that each parameter occupies the same percentage of the total display line in any resolution monitor. This has the benefit of permitting the same horizontal counter values to be used to control both the native low resolution display as well as the promoted high resolution display. In the above figures, sync start and stop points occur at the same time point of each horizontal line in both the native display and the promoted display. Careful programming of monitor control and sync signals permit this linear relationship to exist between all monitors Commodore desires to support. This will minimize compatibility problems associated with software which writes directly to RGA registers. A following section shows examples of how to program the graphics control registers so that this relationship can be achieved to drive five different monitors.

The PIXCLKSEL field of the BLCON3 register provides software with the means for adjusting the pixel clock speed. The value programmed into this field is applied to ANDREA's FREQCTRL pins at the beginning of each scan line which in turn causes system clock generation circuitry to switch to the requested frequency setting. Exact frequencies available are system and monitor specific and will be determined by the systems designers as needed. The system clock circuits follow a specific procedure when frequency changes are made to the pixel clock (PCLK) to prevent sporadic glitches or clock slivers. Schematics in Apppendix B show two examples of logic designs which implement such clock generation circuits. One example shows a high end system implementation which can provide all frequencies needed to drive five different monitors. The other example shows a cheaper circuit which could be used in a low end system to drive three different monitors. Both circuit designs presented should be implemented in a gate array or standard cell technology to keep system costs within reason.

If the horizontal adjustment discussed in the preceeding paragraph were the only action taken in promotion mode, then the low resolution screen would be the proper width, but it would be squeezed to the top of the monitor because only 200 lines would be displayed on a 1K screen. Severe vertical distortion results. To overcome the vertical problem, each promoted display line is repeated 1 to 4 times. This will remove most of the xy distortion and provide a readable display.

5.5.2 Hardware Behind Screen Promotion. -

A two bit 'line advance counter' has been incorporated into the new chip set to accomodate the vertical line repeat function described in the last section. The modulo of this counter is programmable to be 0, 1, 2, or 3. When enabled, this counter serves two purposes: it controls how many times a line is displayed, and it is used to increment a new secondary vertical counter. (More on this later.) A set of working registers has also been added to the chip set. Most of the RGA registers used to control display attributes now have a holding register which is written by the processor, and a working register which is loaded from the holding register at the beginning of each display line. The bitplane pointer registers are among those that now incorporate working registers.

In the old chip set, at the end of every line, a modulo is added to each bitplane pointer to advance the pointers to the next line to be displayed. When running in native mode on the new chip set, this same sequence controls advancement of the bitplane pointers. However, when the line advance counter is enabled(meaning the chip set is operating

in promotion mode), a new sequence of events occurs. The line advance counter is incremented every line. If this counter does not go to zero, then the working bit plane pointer register is reloaded with the value in the holding register and the same line is displayed again. If the line advance counter goes to zero, then the bit plane modulo is added to the working bit plane pointer register and the result of this addition is stored back into the working bit plane register as well as back into the holding register. This advances the bit plane pointers to the next display line.

Vertical blanking(VBSTRT, VBSTOP), vertical sync(VSSTRT, VSSTOP), and vertical total(VTOTAL) monitor control registers are never changed from the values needed to control the native mode display. Only the horizontal compare registers are updated when an promoted screen is displayed. An astute reader will now ask "if that's the case how do we make vertically timed events such as sprite placement occur at the same relative position on the screen as they occurred in the native lowres system?" Enter the secondary vertical counter. The old chip set vertical compare registers have been broken up into two groups. The primary group consists of VBSTOP, VBSTRT, VSSTOP, VSSTRT, and VTOTAL, and controls monitor specific timing functions. The secondary group consists of COPINS, DIWSTRT, DIWSTOP, DIWHIGH, SPRxPOS, SPRxCTL, VPOSR, VPOSW, VHPOSR, and VHPOSW. This group controls application program specific vertical time related functions. It is this group of registers which must receive identical line count information regardless of display resolution. Two vertical counters will be implemented to drive these groups. One for the primary registers, and one for the secondary registers. When the system is operating in native mode, both of these counters will contain the same value at all times and they are both incremented at the end of each display line. However, when the chip set is running in promotion mode, the secondary vertical counter will be incremented only when the line advance counter rolls over to zero. The primary vertical counter will continue to be incremented at the end of each display line. When the primary counter is reset to zero at VTOTAL, so are the secondary and the line advance counters. Comparisons made against registers in the secondary group such as copper wait instruction comparisons or SPRITE fetch comparisons, are only made during one physical display line. That is they will occur during the first line of a repeat sequence rather than occuring each time the line is repeated. If required, the Sprite and graphics DMA channels may use all of the time available during each of the repeated lines to complete the necessary DMA cycles without causing system errors. Data fetched as a result of these extended DMA cycles will be displayed during the next 'virtual' (promoted) scan line.

5.5.3 Software Control Of Graphics. -

At system reset all monitor control signals are disabled. It will take explicit action on the part of system software to start the monitor. Kickstart software comes up and does the normal kickstart things. At some point in this process, kickstart will go out and find out what kind of monitor is connected to the system by reading the new MONITORID RGA register. Having determined what resolution monitor is connected to the system, the monitor control registers are programmed. Monitor control registers needing attention are VBSTOP, VBSTRT, VSSTOP, VSSTRT, VTOTAL, BPLCON0-3, DDFSTOP, DDFSTRT, DWISTRT, DWISTOP, HBSTOP, HBSTRT, HCENTER, HSSTOP, HSSTRT, HTOTAL, BRSTSTRT, BRSTSTOP, EQU1STOP, EQU2STOP, SER1STOP, and SER2STOP. (It should be immediately noted that most of these registers did not exist in the A500 and A2000 systems. This is good because it means old software will not know enough to look

for the new registers to change them from the values programmed by the operating system.) Once these registers have been programmed, BEAMCON0 is written. This register programs sync signal polarity, enables the system light pen, and enables NTSC or PAL mode (the NTSC and PAL bits can only be enabled in native mode; they bits cannot be set during NTSC/PAL promotion on higher resolution monitors). BEAMCON0 also contains the BEAMEN bit which is now set to enable sync and graphics, and the BEAMLCK bit which is set to disable future system resets from clearing the BEAMEN bit (See detailed description of these bits in the BEAMCON0 register description of Appendix F). If a monitor other than an NTSC(PAL) type monitor is found, the registers will be initialized for NTSC(PAL) promotion. This initial setting anticipates that the user is going to boot an old program. Given this initial setting, a user could then insert an old A500 game into the machine and have it operate properly. If a new workbench disk is instead booted, the operating system would change the monitor control registers to enter native graphics mode.

At some point after the initial setup has been done, the user may wish to bring up an old style (low resolution) program in a separate AMIGA screen. System software calls would be made to accomplish this. When this new screen is 'on top', moving into promotion mode is simply a matter of reprogramming the monitor control registers. When the user then pulls down the low resolution screen to reveal the high resolution screen behind, on the fly mid-display monitor resolution changes must be made. This is accomplished using the copper to change the display mode. The last line of the first (upper) screen must be blanked to avoid problems with bitplane pointers being over written by ANDREA during normal display processing. The copper should be made to wait for the line which is three display lines before the line where the promoted screen is to start. When the copper wakes up, it is made to set BPU of BPLCON to zero and OBPLMODE to four. This will blank the last line of the current screen. The copper then waits for the beginning of the next display line. When the copper wakes up, the BPLnPT registers, the SPRnPT registers, DDFSTRT, DDFSTOP, BPU0-4 of BPLCON0 and OBPLMODE of BPLCON3 are set to display appropriate data for the promoted screen. (HARDWARE NOTE: MONICA does not want BPU0-4 or OBPLMODE on the line which this change occurs. Instead, she must pipeline this information one additional horizontal line in order to compensate for the line delay thru LINDA.) The copper then waits again for the start of the next display line. When this line occurs, all of the above mentioned registers are transfered to working registers and ANDREA will begin fetching data for the first line of the promoted screen. Note that the bitplane data being fetched is stored in LINDA at this time, and SPRITE data is stored in holding registers in MONICA. At the same time that next screen line data is being fetched, ANDREA is controlling the display of the last line of the previous screen. Because BPU and OBPLMODE had been set for blanking, this line is blank. The copper wakes up for the second time at this point. Now, the following registers are loaded with data appropriate for controlling the promoted screen: BPLCON0-3(except BPU and OBPLMODE), DIWSTRT, DIWSTOP, HBSTOP, HBSTRT, HCENTER, HSSTRT, HSSTOP, and HTOTAL. These registers are transfered to their appropriate working registers at the end of the current line to control the display of the promoted line. PIXCLKSEL of BPLCON3 will have been modified to select the proper pixel frequency for the promotion. When the registers are transferred to their working registers, the system will begin using the new pixel clock and will be in promotion mode. (Note that the copper must have access to 26 bus cycles in order to update all these registers in one line time. In a 1280 x 1K system, this is slightly less than half the total available cycles. If, because of the choosen color depth, there are not enough bus cycles available to update all the registers, then

more than one line must be blanked. It is recommended that system software blank not only the last line of the top screen, but also the first line of the bottom screen as well.) It will probably be necessary to update some of the color registers for use in the promoted screen. The color registers are not double buffered as in the case of the above mentioned registers so it will be necessary to update these during the blanked line and the horizontal blanking period immediately before the promoted screen display starts. If there are not enough cycles during this interval to update all the necessary color registers, then a second and possibly third line will have to be blanked before display can be re-enabled.

Switching back to a native screen from the promoted screen follows the same procedure as described in the preceeding paragraph.

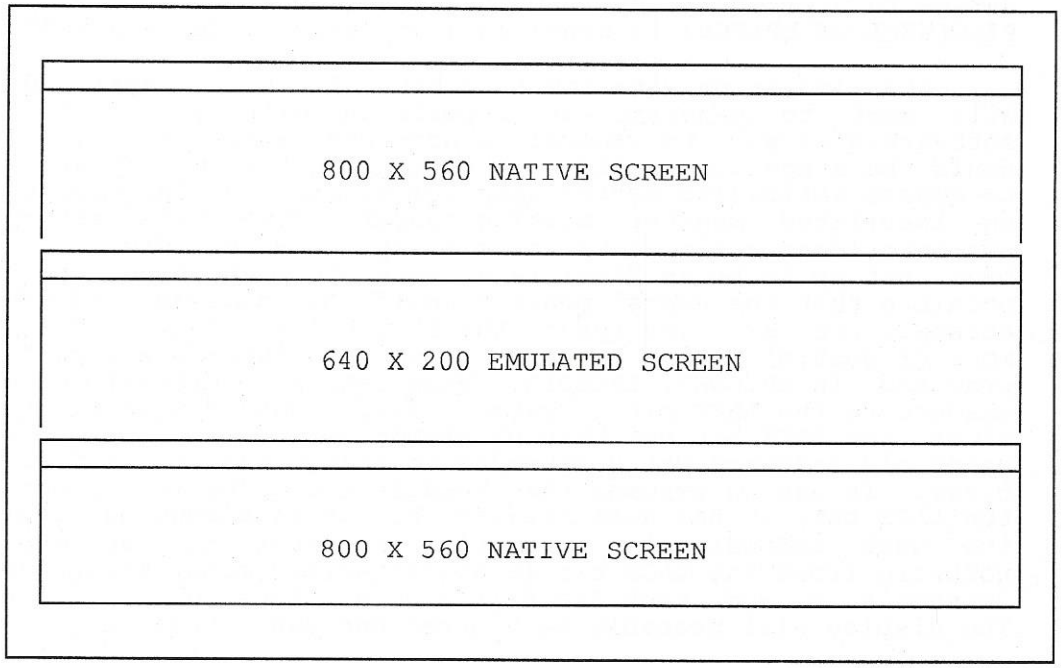
A hardware restriction exists which must be attended to by software. A hardware error will occur if the primary vertical counter reaches VTOTAL and resets to line 0, but the line advance counter does not simultanously roll over to 0. The easiest way for software to prevent this situation from occuring is to program the copper to wait until two lines before the vertical blanking interval, then wake up and reprogram the display to enter native display mode (non-promotion mode). Exit from promotion mode is thereby done off screen before VTOTAL time, and it is guaranteed that the primary counter and line advance counter will roll over to 0 at the same time for all remaining lines of the screen. However it is done, the number of repeated lines per field must sum to VTOTAL in order for emulation mode to operate properly.

There is another reason why it is good practice to switch the display back to native mode at the end of each display field. That is under certain conditions, interlaced display or displays set with LOF=1, and CONSPRT bit of SPRxCTLX (constant sprite) is enabled, multiple sprite DMA accesses will occur (one for each repeated line) thru-out the vertical blanking region. This is also true when CONSPRT is not enabled, but the LOF=1 condition is meet. Multiple sprite position and control DMA cycles will occur during the last line of vertical blanking. To prevent this problem, the display should be in native mode during the entire vertical blanking region.

5.5.4 Multiple Screen Example. -

An 800x560 pixel monitor is currently displaying 3 screens. The backmost screen is a native 800x560 display and appears at the top of the monitor. The middle screen is pulled down about 1/3 of the way and is promoting a 640x200 display. The frontmost screen is a native mode 800x560 display and is pulled down about 2/3 of the way. All three screens are visible; an 800x560 screen at the top of the monitor, a 640x200 screen in the center of the monitor, and another 800x560 screen at the bottom of the monitor.

Primary Vertical Counter	Secondary Vertical Counter
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	7
9	8
10	8
11	9
12	9
13	10
14	11
15	12
16	13
17	14
18	15
19	16
20	17



As can be seen, the secondary counter is 3 counts behind the primary counter at the end of the display field. What this implies is that if, for instance, the copper was to be invoked at physical line 20 to perform some task, then the coppers wait instruction must be modified to wait for line 17(remember that the copper keys off the secondary counter(poof, see COPRSEL of BEAMCON0)) instead of line 20. Futhermore, if the bottom screen is dragged down one more physical line to reveal a little more of the promoted screen, then to be invoked the copper wait instruction must again be modified to wait for line 16. As long as system software accounts for the variable vertical count effects, then screen promotion can be successfully accomplished in a multi screen environment.

5.5.5 Compatibility Issues. -

Not all software written for older machines will run in promotion mode. This section will discuss various classes of old software and point out considerations which would tend to cause problems.

5.5.5.1 Old RGA Register Control Bits. -

For the most part, software will not be interested in changing monitor control registers. Registers which are most likely to be changed are settings in BPLCON0-2, the DDF registers, and the DIW registers. Of concern to this analysis are the bits LACE and ERSY of BPLCON0. Except as noted in other sections of the AAA spec, software can change other bits in these registers without affecting promotion display modes.

For obvious reasons, screen promotion mode will not support genlock devices. A user must have his system configured with a monitor of the same horizontal and vertical frequencies as the genlock device in order to operate properly. As a precaution, in order to prevent damage to user equipment, ANDREA will disable all sync outputs whenever PIXCLKSEL of BPLCON3 is non-zero (not native mode) and ERSY is set.

This leaves us with the LACE bit. In some cases, old software will want to generate an interlaced display. In other cases, old software will want to generate a non-interlaced display. The monitor could be a non-interlaced monitor meaning system software will attempt to coerce interlaced screen data for display, or the monitor could be an interlaced monitor meaning system software will attempt to coerce non-interlaced screen data for display. The new video circuits have been set up to be as flexible as possible with respect to this. It is possible that the users' monitor could be damaged if user software attempts to set (or reset) the LACE bit without also modifying other monitor control bits. For this reason, an interlace lock bit has been provided in BPLCON3, LACEDIS. When set, this bit will prevent further changes to the LACE bit. System software should always set this bit after the LACE bit has been programmed to the appropriate value. This means old software which attempts to change the LACE bit will probably break. It can be assumed that because the software attempted to modify the LACE bit, it has also tweaked the bitplane modulos registers. If the user software is attempting to enter interlace mode but system software froze the LACE bit in non-interlace mode, the screen will be deinterlaced and each field will be displayed one on top of the next. The display will probably be blurred and exhibit jitter.

5.5.5.2 Software Which Takes Over The Machine. -

Some software, especially games, take over the machine when run. This kind of software will only run properly when booted with a system reset (thus giving kickstart a chance to set up the monitor to NTSC(PAL) promotion mode, or when launced from a screen already set to NTSC(PAL) promotion mode. Software which wants to run noninterlaced will most likely run ok. However, one attribute of screen promotion mode may cause problems. Due to the mechanics of promotion mode, the vertical count as seen by the copper (and processor) may achieve counts greater than the NTSC terminal count of 261. (For example, to promote a 640x200 screen on an 800x560 monitor, each display line is set up to repeat two times. A total of 400 of the 560 visible scan lines will be covered. Under these conditions, when the primary counter achieves the terminal count (VTOTAL) of 592, the secondary counter will have reached a count of 296 (or more if the screen is re-programmed to native mode in the vertical blanking region).) It is common practice to program the copper to stop execution until the next vertical field by executing a WAIT instruction where the wait value is a count which is not supposed to occur. If the old software used a value which will occur in the new system, crash. The extra counts seen will vary with the monitor used. For example, an promoted 640x200 display on a 800x560 monitor will see the counter go from 0 to 295. Counts from 0 to 267 will be seen on 1024x768 monitors, and counts from 0 to 270 will be seen on 1280x1024 monitors.

Other problems can occur when old software takes over the machine at boot time. As mentioned in the previous section, if the program attempts to switch to interlace mode, the promotion hardware will deinterlace the image and display the fields successively one directly on top of the other. In the best case the image will be blurred, in

the worst case the image will have jitter and flicker.

5.5.5.3 Software Which Writes Directly To Registers. -

This situation has the same problems as those incurred in the case where old software takes over the machine at boot time.

5.5.5.4 Software Which Uses System Software Calls. -

This kind of software is most likely to run successfully when the chips are in promotion mode. Because system calls are used for all graphics, the operating system can coerce display requests to meet promotion requirements. Noninterlaced screens would be displayed almost directly and interlaced screens would be coerced to be displayed as noninterlaced screens. The operating system is in control of the copper so there are no problems with extra line counts as there are in situations where user software takes over the machine.

5.5.6 Monitor Timing Parameters And Screen Promotion Mode. -

High resolution monitors are typically tolerant of minor deviations from spec sync timings. Because of this it is possible to adjust clock frequencies so that the active display portion of a line on one monitor is the same percentage as the active display portion of a line on a monitor of any other resolution. Designing this kind of relationship into video sync signals allows the horizontal compare register values used to drive one resolution monitor be used to drive any other monitor in promotion mode. The following list outlines critical timing parameters showing one way of achieving this relationship.

(Note: NTSC and PAL sync specs are not as tight as modern highres monitor specs. To make the NTSC horizontal compare values match those of higher res monitors, the NTSC promotion modes assume that active data encompasses 704 pixels. What this really means is that when original resolution hori. compare register data are used to drive an promotion mode, 640 pixel screens will be scrunched towards the center of the screen a total of 32 pixels per side. 640 wide NTSC promotions will cover 90.9% of the horizontal line. Of course this makes promotion of overscan modes trivial.)

NTSC(noninterlaced)

Display Resolution:	640x200
Pixel Frequency:	14.318182 Mhz
Freq avail ICS IC:	14.318182 Mhz
Pixels per line:	910
Hori. Frequency:	15734.3 hz
Lines per Frame:	262
Frame Rate:	60.05 hz

Low Resolution Screen Promotion

(There are no lower resolution screens to promote in NTSC mode!)

VGA

```
Display Resolution: 640x400(or 480)
Pixel Frequency:    28.636364 Mhz
Freq avail ICS IC:  28.636364 Mhz
Pixels per line:    910
Hori. Frequency:    31468.5   hz
Lines per Frame:    525
Frame Rate:         59.94     hz
```

Low Resolution Screen Promotion

```
Screen Promotion:   640x200          Repeat each line 2 times.
Pixel Frequency:    28.636364 Mhz    (All of the screen is filled)
Freq avail ICS IC:  28.636364 Mhz
Pixels per line:    910
Hori. Frequency:    31468.5   hz
Lines per Frame:    525
Frame Rate:         59.94     hz
```

VGA+

```
Display Resolution: 800x560
Pixel Frequency:    36        Mhz
Freq avail ICS IC:  35.999999 Mhz
Pixels per line:    1034
Hori. Frequency:    34816.3   hz
Lines per Frame:    592
Frame Rate:         58.81     hz
```

Low Resolution Screen Promotion

```
Screen Promotion:   640x200          Repeat each line 2 times.
Pixel Frequency:    31.7      Mhz    (upper 71.4% of screen is filled)
Freq avail ICS IC:  31.682785 Mhz
Pixels per line:    910
Hori. Frequency:    34835.2   hz
Lines per Frame:    592
Frame Rate:         58.8      hz

Screen Promotion:   640x400(or 480)  Repeat each line once.
Pixel Frequency:    31.7      Mhz    (upper 71.4% of screen is filled)
Freq avail ICS IC:  31.682785 Mhz
Pixels per line:    910
Hori. Frequency:    34835.2   hz
Lines per Frame:    592
Frame Rate:         58.8      hz
```

1K x 768

```
Display Resolution: 1024x768
Pixel Frequency:    64        Mhz
Freq avail ICS IC:  64.010695 Mhz
Pixels per line:    1324
Hori. Frequency:    48338.4   hz
Lines per Frame:    808
Frame Rate:         59.82     hz
```

Low Resolution Screen Promotion

```
Screen Promotion:   640x200          Repeat each line 3 times.
Pixel Frequency:    44        Mhz    (upper 78.125% of screen is filled)
Freq avail ICS IC:  43.985454 Mhz
Pixels per line:    910
Hori. Frequency:    48351.7   hz
Lines per Frame:    808
Frame Rate:         59.84     hz
```

```
Screen Promotion:   640x400(or 480)  Repeat each line once.
Pixel Frequency:    44          Mhz  (upper 52.1% of screen is filled)
Freq avail ICS IC:  43.985454   Mhz
Pixels per line:    910
Hori. Frequency:    48351.7     hz
Lines per Frame:    808
Frame Rate:         59.84       hz

Screen Promotion:   800x560          Repeat each line once.
Pixel Frequency:    50          Mhz  (upper 72.9% of screen is filled)
Freq avail ICS IC:  49.983471   Mhz
Pixels per line:    1034
Hori. Frequency:    48355.9     hz
Lines per Frame:    808
Frame Rate:         59.85       hz


                                1280 x 1024
Display Resolution: 1280x1024
Pixel Frequency:    110         Mhz
Freq avail ICS IC:  109.99999   Mhz
Pixels per line:    1656
Hori. Frequency:    66425.1     hz
Lines per Frame:    1084
Frame Rate:         61.28       hz
                    Low Resolution Screen Promotion
Screen Promotion:   640x200          Repeat each line 4 times.
Pixel Frequency:    60.45       Mhz  (upper 78.125% of screen is filled)
Freq avail ICS IC:  60.454545   Mhz
Pixels per line:    910
Hori. Frequency:    66428.6     hz
Lines per Frame:    1084
Frame Rate:         61.28       hz

Screen Promotion:   640x400(or 480)  Repeat each line 2 times.
Pixel Frequency:    60.45       Mhz  (upper 78.125% of screen is filled)
Freq avail ICS IC:  60.454545   Mhz
Pixels per line:    910
Hori. Frequency:    66428.6     hz
Lines per Frame:    1084
Frame Rate:         61.28       hz

Screen Promotion:   800x560          Repeat each line once.
Pixel Frequency:    68.7        Mhz  (upper 54.7% of screen is filled)
Freq avail ICS IC:  68.727272   Mhz
Pixels per line:    1034
Hori. Frequency:    66441       hz
Lines per Frame:    1084
Frame Rate:         61.3        hz

Screen Promotion:   1024x768         Repeat each line once.
Pixel Frequency:    87.95       Mhz  (Upper 75% of screen is filled)
Freq avail ICS IC:  87.954545   Mhz
Pixels per line:    1324
Hori. Frequency:    66427.5     hz
Lines per Frame:    1084
Frame Rate:         61.28       hz
```

5.5.7 Graphics Which Cannot Be Promoted To High Resolution Screens. -

In certain configurations, running screen promotion mode puts constraints on available graphics DMA cycles and limits the amount of graphics data which can be fetched for a given scan line. The chips have been designed to use all available bandwidth while fetching display data in screen promotion mode. Fetches for any given scan line will continue as necessary across all repeated scan lines. In some cases, the time which elapses while the previous promoted scan line is displayed is equal to or greater than the time which elapses in the native display mode. In these cases, all graphics modes which can be displayed in native mode can also be displayed on the promoted screen. In other cases, the total time which elapses while the previous promoted scan line is displayed is less than the time which elapses in the native display mode. In these cases, the graphics modes acheivable are not as great as those acheivable on the native screen. This phenomonum is similar to that which exists when comparisons are made between graphics modes possible in a standard Fast Page Mode DRAM system, and graphics modes possible in the same system with VRAM. Note also the discrepancy between graphics modes possible in high end systems and graphics modes possible in low end systems.

The following tables show the maximum graphics modes available for each of the example promotion modes presented in the previous section. Following each table is a comparison with data presented in chapter 3. This comparison indicates the extent to which the promotion has limited the maximum color depth available.

Low End System, Fast Page Mode DRAM

Promoted Graphics Capabilities on 800x560 Monitor

Display Res.	Bitplane	HalfChunky	Chunky	Hybrid	PackLUT	PackHY
640/200	16	YES	YES	YES	YES	YES
704/200	16	YES	YES	YES	YES	YES
640/400	11	YES	YES	NO	YES	YES

(12 bitplanes are possible in native 640x400 monitors)

Low End System, Video DRAM

Promoted Graphics Capabilities on 800x560 Monitor

Display Res.	Bitplane	HalfChunky	Chunky	Hybrid	PackLUT	PackHY
640/200	16	YES	YES	YES	YES	YES
704/200	16	YES	YES	YES	YES	YES
640/400	16	YES	YES	YES	YES	YES

(all graphics promotable)

High End System, Fast Page Mode DRAM

Promoted Graphics Capabilities on 1024x768 Monitor

Display Res.	Bitplane	HalfChunky	Chunky	Hybrid	PackLUT	PackHY
640/200	16	YES	YES	YES	YES	YES
704/200	16	YES	YES	YES	YES	YES
640/400	11	YES	YES	YES	YES	YES
800/560	10	YES	YES	NO	YES	YES

(16 bitplanes are possible in native 640x400 monitors)
(13 bitplanes are possible in native 800x560 monitors)
(Hybrid mode is possible in native 800x560 monitors)

Promoted Graphics Capabilities on 1280x1024 Monitor

Display Res. / Bitplane	HalfChunky	Chunky	Hybrid	PackLUT	PackHY
640/200	16	YES	YES	YES	YES	YES
704/200	16	YES	YES	YES	YES	YES
640/400	16	YES	YES	YES	YES	YES
800/560	7	YES	YES	NO	YES	YES
1024/768	6	YES	NO	NO	YES	NO

(13 bitplanes are possible in native 800x560 monitors)
(Hybrid mode is possible in native 800x560 monitors)
(8 bitplanes are possible in native 1024x768 monitors)
(Hybrid mode is possible in native 1024x768 monitors)
(Chunky mode is possible in native 1024x768 monitors)
(PackedHy mode is possible in native 1024x768 monitors)

High End System, Video DRAM

Promoted Graphics Capabilities on 1024x768 Monitor

Display Res.	Bitplane	HalfChunky	Chunky	Hybrid	PackLUT	PackHY
640/200	16	YES	YES	YES	YES	YES
704/200	16	YES	YES	YES	YES	YES
640/400	14	YES	YES	YES	YES	YES
800/560	13	YES	YES	YES	YES	YES

(16 bitplanes are possible in native 640x400 monitors)
(16 bitplanes are possible in native 800x560 monitors)

Promoted Graphics Capabilities on 1280x1024 Monitor

Display Res. Bitplane	HalfChunky	Chunky	Hybrid	PackLUT	PackHY
640/200	16	YES	YES	YES	YES	YES
704/200	16	YES	YES	YES	YES	YES
640/400	16	YES	YES	YES	YES	YES
800/560	9	YES	YES	YES	YES	YES
1024/768	9	YES	YES	NO	YES	YES

(16 bitplanes are possible in native 800x560 monitors)
(11 bitplanes are possible in native 1024x768 monitors)

5.6 New Blitter Modes.

Several new modes have been added to the blitter. This section discusses these new modes.

5.6.1 Pixel Addressed Blitter. -

The new blitter operates using pixel addressing. To perform old mode blits, the blitter is set up by calculating:

1. First and last word masks.
2. Pointers to the *word* in which the blit is to start (A,B,C,D)
3. The size of the blit (in pixels (lines) for height - *words* for width)
4. The function to perform.
5. The modulus (what to add onto the pointer *after* the last blit cycle to get it to the beginning of the next blit). (A,B,C,D)
6. Two control registers that have various bits to determine what is to happen (including the function from above) and *shift amounts* for both the A and B sources.

For many functions, it is also necessary to calculate a mask plane to be used as one of the sources.

New mode blit operations always works on pixel addresses. In addition, much of the setup work shown above is now automatic. The new blitter calculates the correct first and last mask, determines proper

shift amounts, and determines whether or not to do pre-reads (hence eliminating the problem of having to set up the blitter twice when blitting a narrow rectangle across a destination boundary) before performing the blit.

Definitions:

short - a 16 bit quantity

int - a 32 bit quantity

unsigned - a 32 bit quantity

```
typedef
struct point {
        short x;
        short y;
        }POINT;

typedef
struct size {
        short h;
        short w;
        }SIZE;

typedef
struct bitmap_width {
        short w;
        short dummy;
        }BITMAP_WIDTH;
```

5.6.1.1 Raster-op Blits. -

The new mode blitter needs to be provided with the following information for traditional blitting (see Raster-op and Sort figure):

```
typedef
struct rasterop_operands {
   int              bits_per_pixel;
   unsigned         plane_mask;
   unsigned         b_source_bitmap_address;
   unsigned         a_source_bitmap_address;
   unsigned         destination_bitmap_address;
   BITMAP_WIDTH     b_source_bitmap_width;
   BITMAP_WIDTH     a_source_bitmap_width;
   BITMAP_WIDTH     destination_bitmap_width;
   SIZE             rasterop_size;
   POINT            b_source_origin;
   POINT            a_source_origin;
   POINT            destination_origin;
   int              function;
   }BLITTER;
```

where:

[a,b]_source_bitmap_address is the *bit* address of the corner of a bitmap where a rectangle will be used as a source of the blit.

If only one source is needed, it must be the 'a' source.

[a,b]_source_bitmap_width is the width of the bitmap in pixels

[a,b]_source_origin is the rectangular coordinates of the point in the bitmap of the corner of the rectangle to be blitted. Both x and y are 16 bit positive integers relative to [a,b]_source_bitmap_address.

destination_bitmap_address and destination_bitmap_width is the bit address of the corner of a bitmap where the result rectangle will be found, and the width of the destination bitmap. Note that the destination sometimes is used as the third source.

destination_origin is the rectangular coordinate of the point in the bitmap of the corner of the rectangle of the result of the blit. Both x and y are 16 bit positive integers relative to destination_bitmap_address.

bits_per_pixel indicates the size of the pixel (1, 2, 4, 8, or 16 bits). Pixels must fall on their natual boundary (i.e. 4 bit pixels must always be on 4 bit boundaries; 8 bit pixels must always be on byte boundaries, etc.). This information is actually contained in the function RGA.

rasterop_size contains the height and width (both in pixels) of the rectangle to be operated on.

function is the function to perform - information similar to the current blitter control register - function, direction, etc.

plane_mask is only used when performing operations on chunky pixels and indicates which bits of the pixel the function should be performed on. This permits "plane style" operations to be performed on chunky pixel images. The plane_mask acts as a bit mask: where ones exist, the function specified in the "function" field is performed. Where zeros exist, the original pixel bit value is retained.

Blitting begins when the function register is written. Values written into certain control registers will not be overwritten during the blit operation. This enables related successive blit operations to be started with a minimal amount of reprogramming. Reverse blitting requires that the origin of the rectangles to be blitted specify the lower right hand corner.

5.6.1.2 Line Drawing. -

The new blitter needs the following information when drawing lines (see Line Draw figure):

```
typedef
struct rasterop_operands {
   POINT           clip_a;
   POINT           clip_b;
   int             color;
   int             pattern;
   int             plane_mask;
   unsigned        bitmap_address;
   BITMAP_WIDTH    bitmap_width;
   POINT           end_1;
   POINT           end_2;
```

```
    int             function;
    }BLITTER;
```

where:

bitmap_address is the *bit* address of the corner of a bitmap where the line is to be drawn.

bitmap_width is the width of the bitmap in pixels

clip_[a,b] are the points of a bounding box within the bitmap where the line should (or should not) be drawn.

color is the value to be placed at each point drawn. The color value must be repeated for each possible pixel location within a 32 bit word. For example, if 4 bit chunky pixels are to be written, and the value of the color to be inserted is '0xa', the value of the color register should be 0xaaaaaaaa.

pattern is a bit string containing 1's where line points should be colored and 0's where line points should not be drawn. This must contain 0xffffffff for solid lines.

end_[1,2] contains the rectangular coordinates of the first and last points in the line. Both x and y are 14 bit signed integers (sign-extended to 16 bits) relative to bitmap_address.

The new blitter calculates the deltas and error functions, optionally performs clipping, and draws the line using the pattern register. Line drawing will begin when the function register is written. As with raster-op blits, values written into all registers except the endpoint registers and the pattern register will not be overwritten during the line-drawing operation.

Two commands may be used in new line drawing (Clipping may be enabled on either):

LINE_DRAW - A line is drawn between the two given endpoints. At the of the operation, the end1 register will be overwritten with the end2 values.

LINE_CONT - A line is drawn between the two given endpoints. The first point is skipped (since it should have been written at the end of the previous line). At the end of the operation, the end1 register will be overwritten with the end2 values.

This permits poly-lines (into a single bitmap) to be performed by first setting up all the appropriate parameters and submitting a LINEDRAW command. Then submitting a series of LINECONT commands, each of which specifies only a new end2 point and function.

Figure "Line draw flow" describes the operations performed when drawing a clipped line. This series of operations has been specifically selected to support line drawing in "layers".

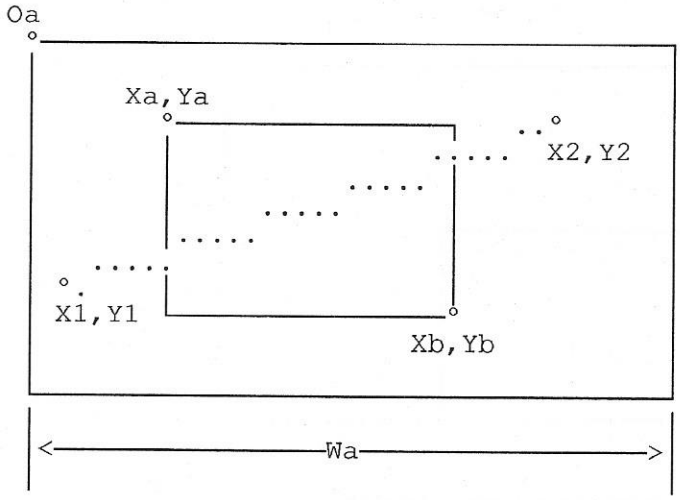
The BRESENHAM line parameters are first calculated as if the entire line is to be drawn. This involves calculating an error term, and two error increments: one to be added each time the error becomes positive (when the dependent variable is to be incremented) and one when the error is negative. Each time the BRESENHAM loop is traversed, the independent variable is incremented. IINC and DINC are also calculated. They represent the quantity (in bit address units) which

must be added to the previous bit address to get to the next bit address of the line. The linebase is also calculated and represents the bit address of end1.

When internal clipping is to occur (this is the mode used in layers), a preclip calculation is performed. This quickly determines whether the line will pass through the clipping region. When it doesn't, the line pattern is adjusted to the position it would have had if the line would have been drawn, and the end2 point coordinates are placed in the end1 location before completing.

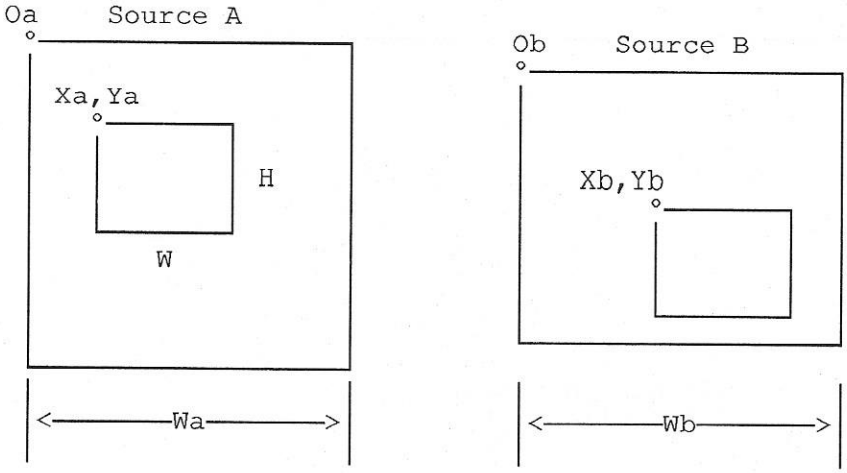
Since the clipped line may be small compared to the overall line length, further processing is performed to minimize the time spent drawing the line. The midpoint of the line is estimated and preclipping is performed on the two resulting line segments. When only one of the line segments appears in the clipped region, the midpoint is recalculated and the procedure is repeated. This continues until both line segments appear within the clipped region.

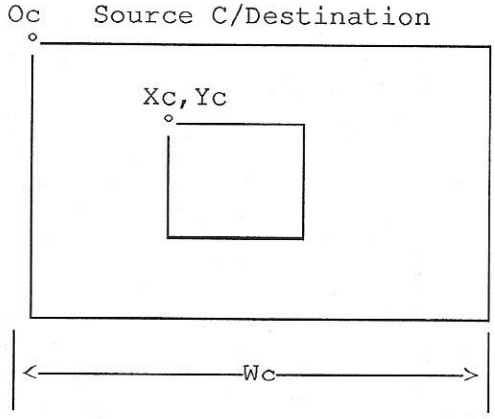
Once the line segment appears in both line sections, drawing is initiated. First, the actual "BRESENHAM point" for the point M, and the error term at M are calculated. This is shown in Figure "Line Draw Error Correction". The line is drawn in two sections: the first begins at M and moves towards end2 until clipping occurs, then the begin point is moved back to M, a point skipped and drawing moves towards end1 until clipping occurs. Of course the line parameters must be negated to move in the opposite direction.



Oa- Source Bitmap Start Address (Bit Address).
Wa- Width of Bitmap (in pixels).
X1,Y1- Pixel Coordinates of Line
X2,Y2 Endpoints.
Xa,Ya- Pixel Coordinates of
Xb,Yb Clipping Rectangle.
BPP- Bits Per Pixel.
PM- Plane Mask.
LP- Line Pattern.
LC- Line Color.

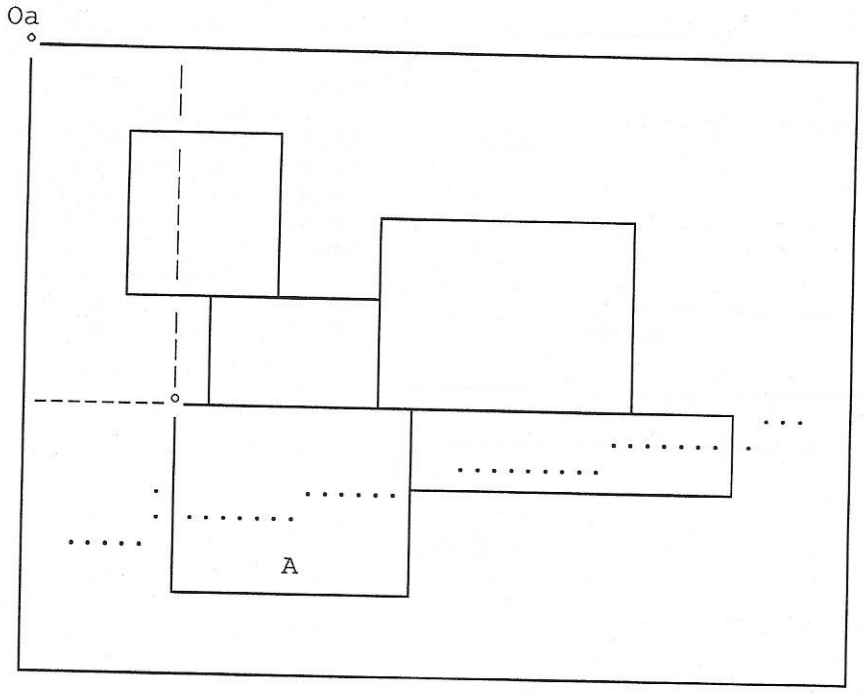
LINE DRAW

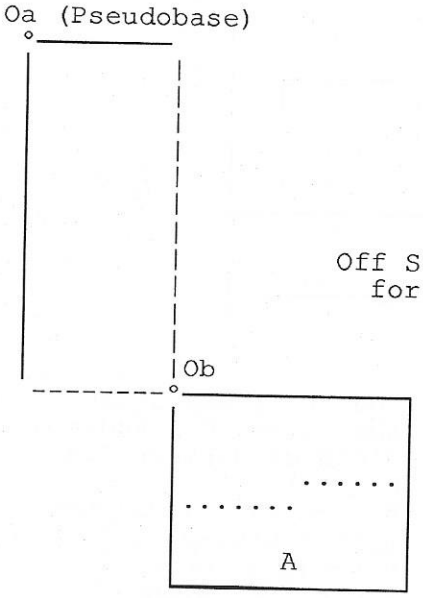




Oa,Ob,-Source Bitmap Start
Oc Addresses (Bit Address).
Wa,Wb,-Width of Bitmaps (in
Wc pixels).
Xa,Ya,-Rectangle Coordinates
Xb,Yb, within Bitmaps relative
Xc,Yc to respective 0.
W,H Width and Height of Raster Op (in Pixels).
BPP Bits Per Pixel.
PM Plane Mask.

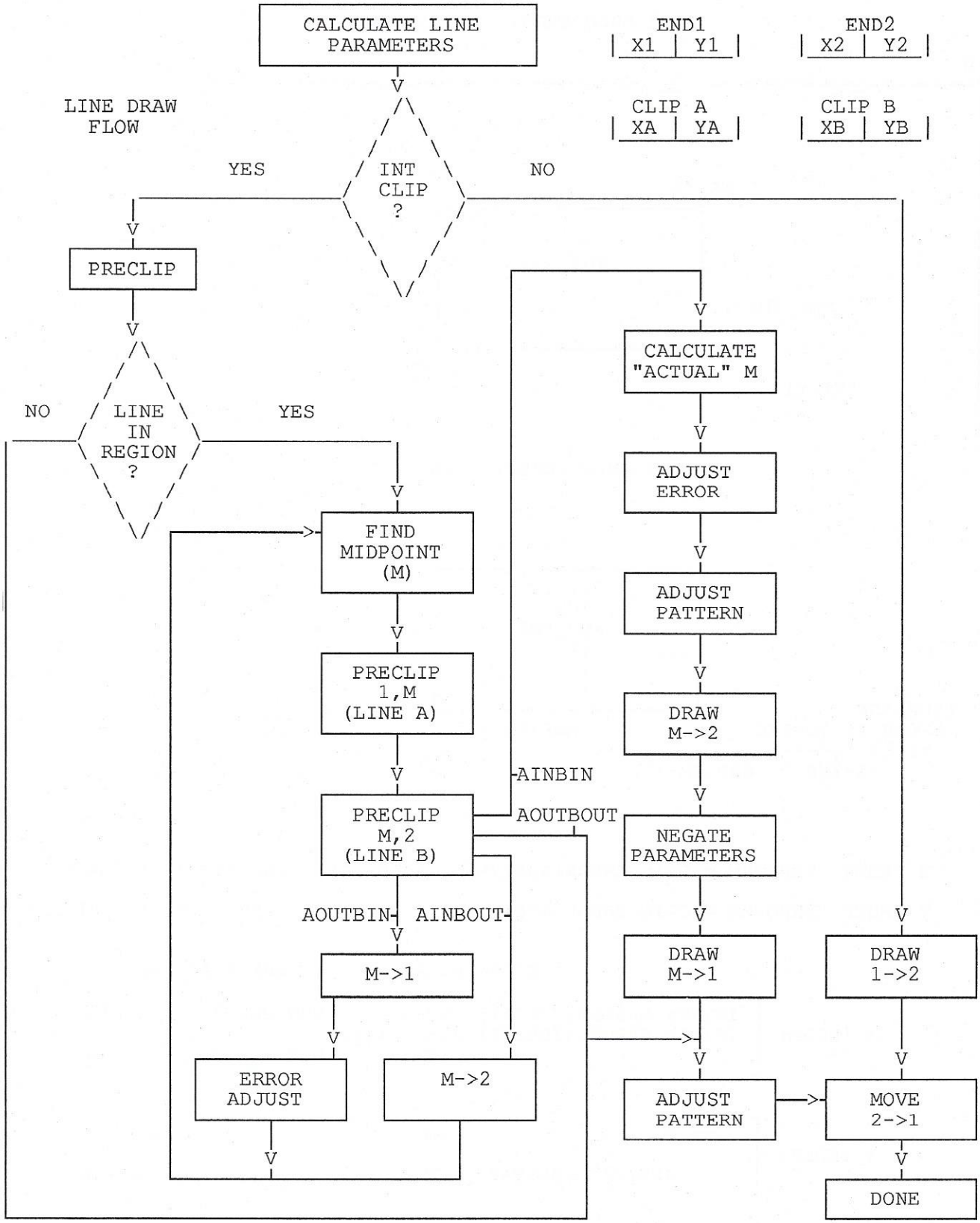
RASTER-OP and SORT



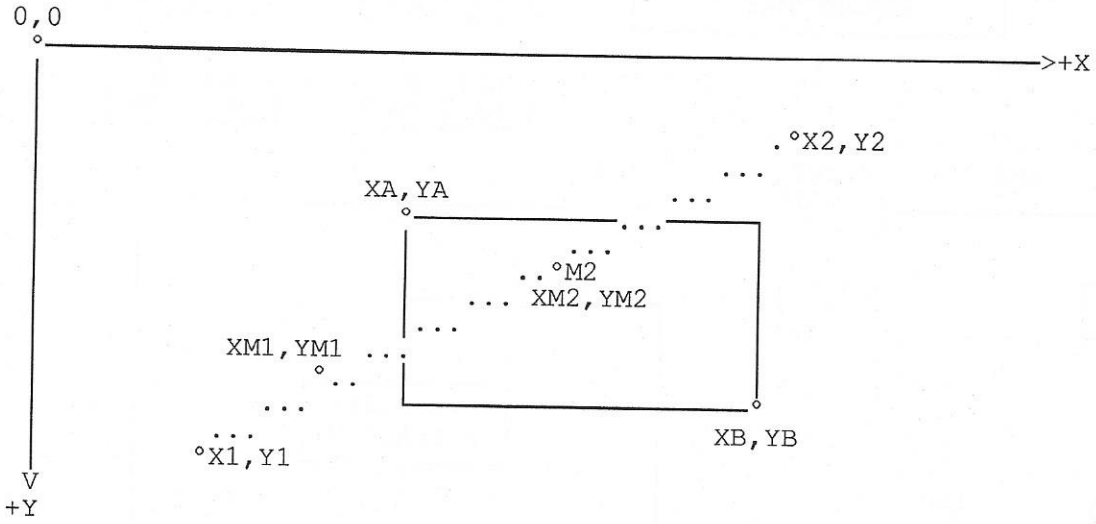


Off Screen Bitmap
for Clip-Rect

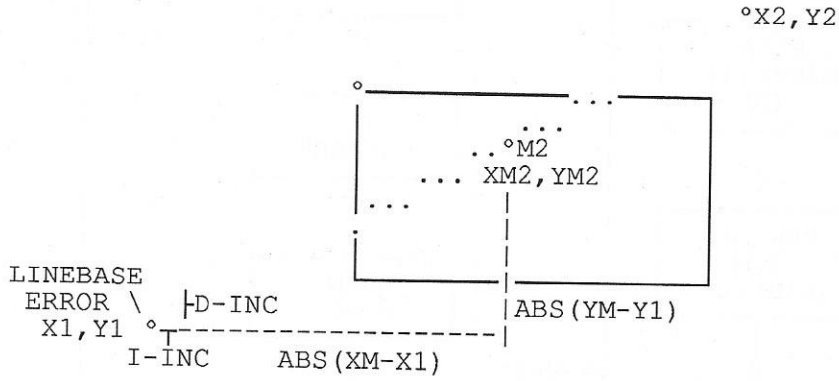
Using the Blitter With Layers


CALCULATE LINE PARAMETERS
END1
X1
Y1
END2
X2
Y2
LINE DRAW FLOW
CLIP A
XA
YA
CLIP B
XB
YB
INT CLIP ?
YES
NO
PRECLIP
LINE IN REGION ?
NO
YES
FIND MIDPOINT (M)
PRECLIP 1,M (LINE A)
PRECLIP M,2 (LINE B)
AINBIN
AOUTBOUT
AOUTBIN
AINBOUT
M->1
ERROR ADJUST
M->2
CALCULATE "ACTUAL" M
ADJUST ERROR
ADJUST PATTERN
DRAW M->2
NEGATE PARAMETERS
DRAW M->1
ADJUST PATTERN
DRAW 1->2
MOVE 2->1
DONE

LINE DRAW ERROR CORRECTION



X INDEPENDENT AXIS



X INDEP ERROR@M = ORIG ERROR+ABS(XM-X1)*(2DELTAY) -ABS(YM-Y1)*(2DELTAX)

Y INDEP ERROR@M = ORIG ERROR-ABS(XM-X1)*(2DELTAY) +ABS(YM-Y1)*(2DELTAX)

		MIDPOINT ADJ	ERROR ADJUST
X INDEP:	IF(NEW ERROR≥2DELTAY)	YM--	NEW ERROR -= 2DELTAX
	IF(NEW ERROR<(2DELTAY-2DELTAX)),	YM++	NEW ERROR += 2DELTAX
Y INDEP:	≥ 2X	XM--	-= 2DELTAY
	< 2DELTAX-2DELTAY	XM++	+= 2DELTAY

5.6.1.3 Using Blitter Sort, Tally, And Arithmetic. -

The blitter needs to be provided with the following information for Bubble Sorting, (see Raster-Op and Sort figure):

```
typedef
struct rasterop_operands {
   unsigned         a_source_bitmap_address;
   unsigned         b_source_bitmap_address;
   BITMAP_WIDTH     a_source_bitmap_width;
   BITMAP_WIDTH     b_source_bitmap_width;
   SIZE             rasterop_size;
   POINT            a_source_origin;
   POINT            b_source_origin;
   int              function;
   }BLITTER;
```

where:

[a,b]_source_bitmap_address is the *bit* address of the corner of a bitmap where a rectangle will be used as a source of the blit.

If only one source is needed, it must be the 'a' source.

[a,b]_source_bitmap_width is the width of the bitmap in pixels

[a,b]_source_origin is the rectangular coordinates of the point in the bitmap of the corner of the rectangle to be blitted. Both x and y are 16 bit positive integers relative to [a,b]_source_bitmap_address.

rasterop_size contains the height and width (both in pixels) of the rectangle to be operated on.

function is the function to perform - information similar to the current blitter control register - function, direction, etc.

bits_per_pixel field of the function indicates the size of the pixel (1, 2, 4, 8, or 16 bits). Pixels must fall on their natural boundary (i.e. 4 bit pixels must always be on 4 bit boundaries; 8 bit pixels must always be on byte boundaries, etc.).

The blitter calculates the correct first and last mask, determines proper shift amounts, etc. as before. Sorting begins when the function register is written and an inplace bubble sort is performed. Values written into registers are not overwritten during the sorting operation.

The Tally operation setup is similar except that the bsource is not used and a destination must be specified.

Chunky mode arithmetic may be performed by setting up the blitter in the traditional sense, but by giving a function code of:

```
ADD2     - Sources A and B are added
ADD2_SAT - Sources A and B are added in saturation mode
ADD3     - Sources A, B, and C (destination) are added
ADD3_SAT - Sources A, B, and C are added in saturation mode
ADD2_AVE - Sources A and B are added then averaged
SUB2     - Source B is subtracted from A
SUB2_SAT - Source B is subtracted from A in saturation mode
```

In all cases, the result is placed in the destination space.

5.6.1.4 Using The Blitter With Layers -

Most new blitter functions have been specifically designed to aid in the manipulations of layers. Consider Figure "Using the blitter with layers". In a windowed environment, the screen is tiled with clip-rects, each of which represents an overlapping window section. As windows are created and moved, the clip-rects are generated and (sometimes) merged.

A particular window is composed of a collection of clip-rects. Some clip-rects may be visible while others are not visible (due to an overlapping window) and are stored in off screen memory. To draw a line in a window requires that the line be drawn thru all the clip-rects which compose that window. Of course some clip-rects will not be drawn into since the line may not touch that portion of the screen, but the algorithm used requires that the line draw be attempted there anyway.

The figure shows a tiled window with a line being drawn through it. Those clip-rects where the line is not to be drawn will be trivially rejected by the preclipping algorithm. Those clip-rects where the line is to be drawn use the algorithm described above.

To aid in blitter operation, when a clip-rect is generated (allocation of off-screen memory, etc.), a pseudo-base should be calculated which represents the point where the base would be if the entire bit map was present (in contiguous form) in this bit map. The width of the original bitmap is also carried forth to the offscreen bitmap.

To draw the line through a series of clip rects involves passing the same parameters to the blitter for each clip-rect except the base (pseudo-base) and clipping coordinates. This may also be sped up by constructing a blitter copper list. (See section 5.7.2.5, 'Using the Copper to Drive the Blitter').

Blitting to clip rects operates in a similar fashion. Only the base, origin and size parameters need to be updated.

5.6.2 Fill Mode On Chunky Types. -

No new fill modes have been added to the chip set. For compatibility, the original fill mode functions are still supported by the chip set. However this mode does not directly support the new chunky pixels.

5.6.3 Sort And Tally Functions. -

Blitter Sort and Tally functions have been added to assist the analysis of chunky graphics images. The sort function may also be useful in supporting merge of clip-rects.

Sort and Tally can be used in data compression algorithms for PACKLUT pixel plane generation to help choose a color look-up table palette. These functions will also prove useful when implementing filtering and noise reduction algoriths by quickly finding the minimum and maximum pixel values of an image.

5.6.3.1 Blitter Sort Function. -

Blitter sorts can be made to operate on any of the chunky graphics data sizes: 2, 4, 8, and 16 bit data. Sort will also operate on 32 bit data. The A source and optionally the B source is used during the sort operation. The A source is set up to point to a chunky plane containing a sort key. A source data defines the final order of the sort lists. The B source can also be enabled. When the B source is enabled, data fetched on the B source will undergo the same manipulations made to the A source, however the B source will not contribute in any way to the sort key. In other words, the same relative 2-bit, nybble, byte, word, or longword swaps made to the A source will also be made to the B source. The destination address register is not preprogrammed. The A source and optionally the B source are the destination planes as well as the source planes. The A and B source bitplane pointer and modulos registers, and blit size register are set up as in standard blitter operations. The B source must be set up to have the same characteristics as the A source. In particular, the B source must begin on the same bit boundary, have the same width and height, and have the same index into the bitmap as the A source. This makes the offsets into both bitmaps the same and insures that the blitter will "run out of pixels" at the same place for both the A and B sources. Once the blit function register has been written, the blitter will be enabled and will traverse thru the planes as many times as necessary to implement a complete bubble sort as defined by the sort key(source A).

It should be obvious at this point how the sort function could be used to sort a plane of Half Chunky pixels. The following example outlines a method of sorting a Hybrid pixel plane.

Sort a plane of hybrid pixels first on the red field, second on the green field, and third on the blue field. The first step of the procedure is to use the blitter copy function to make a copy of the red field. Next, this copy of the red field is accessed as the A source sort key and the B source is pointed at the green field. The blitter sort function is enabled and the green field, as well as this copy of the red field will be sorted as specified by the red field. The sorted copy of the red field is no longer needed. Next, the A source is set to point to the original red field and the B source is set to point to the blue field. The blitter sort is again enabled. At this point, all of the red green and blue fields have been sorted in order of the red field. Next, each group of identical red pixels must be sorted in order of the green field. The blitter tally function is used to determine how many of each identical red color exist. Once the tally has been done, a loop is entered and the green field within each of the identical red pixel groups is sorted. The blue field is made to follow the green field sort during each of these green subfield sorts. Before moving on to the sort of the next green subfield, the blue subfields within each identical green pixel field are sorted. Again, the tally function is used to determine how many pixels of each green color exist.

5.6.3.2 Blitter Tally Function. -

The blitter Tally function is used to determine the number of identical bytes or words which exist in sequence along a plane. It is intended to be used in concert with the blitter sort function to provide statistical analysis of chunky pixel planes. (See examples in preceding section.) Output of the tally function consists of two longwords per identical sequence found. The first longword contains a copy of the data which was being tallied(right justified), and the second longword contains a count of the number of identical bytes or words found. As a example, consider the following sequence of bytes found in memory: 00 00 00 00 01 02 07 07 07 07 07 07 08 08 09 0A. After a tally operation has executed on this plane (byte mode) the destination will contain the following: xxxxxx00 00000004 xxxxxx01 00000001 xxxxxx02 00000001 xxxxxx07 00000006 xxxxxx08 00000002 xxxxxx09 00000001 xxxxxx0A 00000001, where x = don't care.

5.6.4 Chunky Pixel Arithmetic. -

The new blitter is capable of performing addition and subtraction operations on all of the chunky pixel types supported by the chip set (including "Chunky" pixels, 1 ZD, 5 red, 5 green, and 5 blue). Additions can involve two or three operands. Subtraction and averaging can only be used when two sources are enabled.

Three addition modes have been provided:

1. Standard add. Carry out is lost.
2. Saturated add. If carry out is 1, all bits are set.
3. Averaged add. The result of the addition is divided by 2. (The result is right shifted one bit position with the carry out forming the most significant bit. Numbers are assumed to be unsigned.)

Two subtraction modes have been provided:

1. Standard subtract. Borrow out is lost.
2. Saturated subtract. If borrow out is 0, all data bits are cleared.

As with standard blitter copy mode, the A and B sources can be shifted before the addition is made. Additions can be done on any of the supported chunky data lengths, 2, 4, 8, or 16 bit data. In addition to this, a 5 bit data type is possible for use in summing Chunky Pixels. When the 'bitsperpixel' control field indicates 5, the blitter ignores the 16th bit of each word. Remember that a Chunky pixel contains 3 five bit fields, one each for red, green, and blue. It is possible to do single field additions within the longword. To accomplish this, the A source and the destination must point to the same data. The first word mask is programmed to mask the destination field to be modified. When some of the bits of the mask are a zero, the corresponding bits of the destination register are replaced with the original A source bits before the destination data is written.

5.6.5 Multiple Blitter Support. -

The new chip set supports a system which incorporates multiple BLIT chips. A multiple BLIT chip organization allows two to eight blitters process bitplane or hybrid data in parallel, thus yielding a significant increase in blitter performance. The multiple blitter configuration is completely optional and therefore requires substantial system software support. This is because there will be an unknown number of blitters in the system and software must handle different configurations on a case by case basis in order to achieve the most optimal solution in each case. Multiple blitter systems are organized in a 'Blitter per RAM Bank' configuration. When a user wishes to run this type of configuration, he must purchase chip RAM expansion cards which include a BLIT chip. These will plug into the standard chip RAM expansion slots on the motherboard.

The multiple BLIT chip system is a single instruction, multiple data system. Software is responsible for organizing bitplane data in such a way that all external blitters are used efficiently. Typically, one or two bitplanes will be allocated to each physical RAM card. It is a hardware requirement that the starting address of each bitplane have the same relative offset from the start of that cards address space. The BLIT chips in the system are normally programmed simultanously. When a blit operation is started, all of the BLIT chips run synchronously in parallel. When BLIT chip 1 is accessing location M of it's RAM, so are all other BLIT chips in the system. An RGA register has been defined which enables simultanous programming of all BLIT chips, BLITEN. The value in this register is applied to AD bits 24-16 during the RGA portion of the bus cycle. A BLIT chip uses this information along with the RGA address to determine whether or not it is being accessed.

From a software point of view, each BLIT chip in the system looks exactly like the blitter which resides on ANDREA. When external BLIT chips are enabled (non- zero value in BLITEN), ANDREA's blitter is effectively disabled (but still used as explained momentarily). This means that in order to gain a performance advantage, a user must purchase at least two chip RAM expansion cards containing BLIT chips. ANDREA is responsible for allocating bus cycles for each of the many system DMA channels. In order to satisfy blitter cycle requests, the blitter on ANDREA is used to 'shadow' the blitter activity occuring in each of the BLIT chips. The ANDREA blitter makes source and destination DMA requests and these requests are seen by the external BLIT chips as RGA addresses present on the AD bus. Since all blitters are synchronously executing the same instruction on it's piece of data, everything hangs together OK. One point worth mentioning is that the ANDREA blitter is generating DMA requests for the blit operation. As the result of each request, ANDREA either receives source data or writes destination data. This data is garbage. It is prevented from being written to or read from RAM by asserting the special BRAS* signal rather than the normal RAS* signal whenever any of the BLITEN bits are nonzero and the access is being run to satisfy a blitter request. Each BLIT chip which needs to respond to the cycle will recognize the BRAS* signal and operate on the data of it's local RAM access.

5.6.6 Testing The Blitter Microcode ROM. -

Data in the Blitter Microcode ROM can be read directly to facilitate chip testing. This is done by programming the BLTTST and BLTROMD test mode registers.

The following sequence outlines both the external stimulus and internal activity that takes place in this mode of operation.

EXTERNAL	INTERNAL
1) Write 0x7 to BLTTST register (0x480).	
	1a) ROM test address counter is reset to 0 and the test address counter takes over the rom address input bus (MADDR(8:0)).
	1b) Rom location is read and the data latched in the normal output latches.
WAIT at least 32 CLK28 Cycles	
	1c) BLTROMD register is cleared.
	1d) Contents of BLTTST reg modified to 0x3.
	1e) First 27 bits of rom word are shifted out of the data latches through the scan path and into the BLTROMD register (0x40c) with an additional Flag Bit set HI.
2) Read BLTROMD register (0x40c). (Data valid when Flag Bit = 1).	
3) Write 0x7 to BLTTST register (0x480).	
	3a) BLTROMD register is cleared.
	3b) Contents of BLTTST reg modified to 0x3.
WAIT at least 32 CLK28 Cycles	3c) Next 27 bits of rom word are shifted into BLTROMD reg (0x40c) with Flag Bit.
4) Read BLTROMD register (0x40c). (Data valid when Flag Bit = 1).	
5) Write 0x7 to BLTTST register (0x480).	
	5a) BLTROMD register is cleared.
	5b) Contents of BLTTST reg modified to 0x3.
WAIT at least 32 CLK28 Cycles	5c) Last 27 bits of rom word are shifted into BLTROMD reg (0x40c) with Flag Bit.
6) Read BLTROMD register (0x40c). (Data valid when Flag Bit = 1).	
7) Write 0x7 to BLTTST register (0x480).	
	7a) ROM test address counter is incremented.
	7b) Rom location is read and the data latched in the normal output latches.
WAIT at least 32 CLK28 Cycles	7c) BLTROMD register is cleared.
	7d) Contents of BLTTST reg modified to 0x3.
	7e) First 27 bits of rom word are shifted into BLTROMD reg (0x40c) with Flag Bit.

REPEAT SEQUENCE from 2) until all rom data has been read.

The total number of BLTROMD register reads required is three times the total number of ROM words: Readtot is 3 times 444, or 1329 at this time. The total tester time required to dump the ROM is:

Total time required = ((32 + 4)/28Mhz) * 1329 = 1.71ms (Max = 1.98ms)

5.7 COPPER.

The move instruction of the coprocessor has been extended in 32 bit mode to support multiple longword register loads under the control of one move instruction. In addition to this, the coprocessor has been given interrupt processing capabilities.

5.7.1 Multiple Move Instruction. -

The 32 bit version of COPINS has been modified to accommodate multiple move. (Multiple move is not available when 16 bit copper DMA is enabled.) An eight bit field has been defined which indicates how many sequential registers plus 1 are to be written using data found following the move instruction. When this field is 0, one register is written, meaning the move instruction operates as previously used.

5.7.2 Coprocessor Interrupts. -

The COPPER has been redesigned to include interrupt processing capabilities. The primary motivation for this redesign was to allow automatic setup and execution of a series of related blit operations without processor intervention. The coprocessor's blitter interrupt in effect provides a blitter setup DMA channel.

A two level coprocessor stack has been added along with two new jump strobe registers, four new coprocessor address registers, a coprocessor interrupt enable register, and two return from interrupt strobe registers.

5.7.2.1 New Coprocessor Hardware. -

The new jump strobe registers are COPJMP3 and COPJMP4. Like COPJMP1 and COPJMP2, when these registers are written, the coprocessor program counter is written with the value contained in COP3LC or COP4LC respectively. COP3LC and COP4LC differ from COP1LC and COP2LC in that they are pushed onto the copper stack when an interrupt routine is invoked.

COPBLITLC is used to hold the blitter interrupt routine starting address. When a sequence of blitter operations are to be started using the coprocessor blitter interrupt, this register should be written with the starting address of the blitter control copper list.

COPWAIT is a register used to hold the copper program counter pointing to the instruction to be executed when the current wait instruction times out. This register is automatically loaded when a wait instruction is executed. It should never written by the microprocessor.

COP1LC normally contains the starting address of the vertical blanking interrupt routine.

The coprocessor interrupt enable register, COINTRE, is used to enable new coprocessor interrupts. The vertical blanking and wait instruction interrupts are nonmaskable, and therefore cannot be disabled with this register.

The 'return from interrupt' strobe register, COPRET, provides the mechanism for returning from coprocessor interrupt routines. The vertical blanking routine, and wait instruction routines strobe this register after all copper lists for a display field have been executed. The blitter interrupt routine always strobes this register or the COPRETC register (rather than executing a wait instruction) when processing is complete.

COPRETC is similar to the COPRET strobe register in that it provides a 'return from interrupt' mechanism. In addition to this function, the BLIT bit of the COINTRE register is cleared when this register is strobed. COPRETC has been provided to allow the processor to interact with the copper to build blitter restart copper lists synchronously with blitter operation. This will be discussed in a later section.

5.7.2.2 Coprocessor Operation. -

The new Coprocessor operates identically to the way it operated in past revisions of the chip set with the exception that now it is possible to interrupt copper execution so as to execute higher priority interrupt routines. An enabled interrupt is recognized at the end of the current instruction. Once recognized, an interrupt will cause the program counter, the value in COP3LC, the value in COP4LC and the coprocessor state(consisting of the current setting of COPPRI in the DMACONX register, CDANG in the COPCON register, and copper idle status) to be pushed onto a stack. When this has been done, the program counter is loaded with the value of the interrupt routine starting address from the appropriate register, COPPRI (the copper nasty bit) is reset (The copper nasty bit is reset for compatibility reasons. If the interrupt routine wishes to execute in nasty mode, it should set the copper nasty bit.), and CDANG is set. The interrupt routine will then perform the setup operations necessary to satisfy the interrupt condition, and perhaps modify its own interrupt starting address register for the next occurrence of the interrupt. Completion of an interrupt routine is signaled by strobing the COPRET register or by executing a wait instruction. At that point the program counter, status, and jump address registers are restored from the stack. Lower priority interrupt processing will then continue.

Only the vertical blanking interrupt and wait instruction interrupt routines should use the wait instruction. This instruction is used to schedule another wait interrupt, and as such indicates completion of the current interrupt routine. If a wait instruction is executed before a previously scheduled wait instruction has triggered, then the first wait will be lost. If the vertical blanking interrupt routine strobes the COPRET register, then no wait interrupts will occur unless one had been scheduled before the vertical blanking interrupt occurred. If the wait interrupt routine strobes the COPRET register, this signifies all copper processing for this display field is complete(except possible blitter interrupt processing) and no activity will occur until the next vertical blanking interrupt.

5.7.2.3 Supported Interrupts. -

Presently, three coprocessor interrupts have been defined. These are as follows:

Vertical blanking interrupt. The vertical blanking interrupt is the highest priority interrupt. It is nonmaskable, and is invoked with each occurrence of vertical blanking. This interrupt level is normally used to point the bitplane pointers back to the beginning of screen display data.

Wait finished interrupt. Wait interrupts are the second highest priority interrupts and are also nonmaskable. They are scheduled whenever a wait instruction is executed. Once scheduled, a comparitor will watch the line counter to match the compare value specified by the wait instruction. Once the match occurs, a wait interrupt flag is set and the wait interrupt routine will be executed as soon as it becomes the highest priority pending interrupt.

Blitter interrupt. The Blitter interrupt provides a convenient method of restarting the blitter when a series of related blit operations are to be done. This interrupt tends to render the Blitter Finished Flag of the wait instruction useless, but the flag has been retained for compatibility purposes. With the advent of the blitter interrupt, usage of the Blitter Finished Flag is discouraged.

5.7.2.4 Coprocessor Interrupt Levels. -

There are three coprocessor interrupt levels. Level 3 interrupts have priority over level 2 interrupts and will execute to completion before returning to complete a lower priority level 2 or 1 interrupt. Level 2 interrupt routines have priority over level 1 interrupt routines and can be interrupted by level 3 interrupts.

Because the blitter copper list is essentially a sequentially executed program, and because a finite amount of time will elapse between the time a blit operation is started and the time the blitter copper list finishes, it is possible to be executing a Level 1 Blitter Finished interrupt rountine and have a second Blitter Finished interrupt occur before the first routine has completed. When this happens, the second Blitter interrupt will be served as soon as the first completes. (If two blitter finished interrupts occur while in the midst of another blitter interrupt service routine(you're doing something pretty strange!), only one additional blitter interrupt routine will be invoked.) The copper interrupt request bits are implemented as flip flops. These flip flops are set when the corresponding interrupt event occurs, and reset as soon as that interrupt level is entered. The request bit can then be set again while still in the original service routine and the interrupt level will be re-entered after the COPRET register is strobed.

The following list shows relative priority levels of the interrupts.

Level 3 Vertical Blanking Interrupt(Nonmaskable)
Level 2 Wait Finished Interrupt
Level 1 Blitter Finished Interrupt
Idle

5.7.2.5 Using The COPPER To Drive The Blitter. -

Using the blitter interrupt facility of the new coprocessor, it is possible to set up a series of related blit operations which are executed without processor intervention. This capability allows a more efficient system because using the facility eliminates much of the CPU interrupt processing overhead which was incurred in old systems. The copper list needed to control the blit operations can be set up in advance, or they can be generated 'on the fly' in parallel with blitter execution. The following paragraphs describe the procedure necessary to generate blitter setup copper lists on the fly.

The basic idea behind 'on the fly' generation of copper lists is that as soon as the processor has determined appropriate parameters to start up a blitter step, the blitter is started, and the processor moves on to calculate parameters for the next step. In such a situation, it is possible that the blitter could get ahead of the processor and finish one blit step before the processor has parameters ready for the next. In order to assure that the task is completed correctly, a specific procedure must be followed. This procedure will provide the necessary handshaking between the processor, copper, and blitter to guarantee correct operation.

To begin the process, the processor readies a copper list which is made up of a series of move instructions necessary to setup the blitter for the first blit. The following gives an example of such a list. It assumes that the blitter is currently idle.

```
address0          MOVE   BLTAMOD,
                           .
                           .
                      blitter setup instructions
                           .
                           .
addressN          MOVE   GENERALW, (blit operation code)
addressN+1        MOVE   COPBLITLC, next-address0
addressN+2        MOVE   BLTFUNCX, ...
addressN+3        MOVE   COPRET,   ...
next-address0     MOVE   COPRETC,  ...
                  MOVE   COPJMP3LC, (start address of newly allocated
                                      COPPER list memory)
                  MOVE   COPJMP3
```

To start the copper, the processor must write the address of the first copper instruction (address0 in the example)into the COPBLITLC register. Next, the processor enables copper blitter interrupts by setting the BLIT bit in the COINTREW register. Finally, the processor sets the BLITRQ bit of the COINTREW register. This starts the copper if the blitter interrupt is currently the highest priority request. (If it's not, the copper blitter interrupt routine will start as soon as all higher priority interrupts have completed.) Once started, the copper will execute all the blitter setup instructions in the list. In the example, there are seven instructions following the blitter setup intructions. These instructions are special in that they control the copper side of the handshaking between the processor, copper, and blitter. The first special instruction is used to set a 'blit operation code' in the general purpose 'GENERAL' register. This register can be polled by the processor at any time so as to determine which blit is currently being executed, and therefore identify which blits have been completed. The second instruction reprograms the blitter interrupt routine starting address pointer to point to the address of the copper code where COPPER processing is to resume once this blit is complete. Note that the instruction at that address (next-address0) is a COPRETC instruction. The reason for this is that

the copper must be made to stop if the blitter completes before the processor is finished making ready the next blitter setup copper list. The processor will remove this instruction when it is ready as explained later. The third special instruction (MOVE BLTFUNCX,...) is the one which actually starts the blitter. This is located after the 'MOVE COPBLITLC,...' instruction because to assure proper hand-shaking, the blitter cannot be started until after a new starting address has been written into the COPBLITLC register. Finally, a return from interrupt is generated by the COPRET instruction and the copper will go idle until the blit operation has completed. The last two instructions provide a COPPER jump mechanism which allows nonsequential memory locations to be allocated on the fly and used as COPPER lists are built. As the next starting address will most likely not be known when the COPBLITLC address is needed in the current list, this instruction provides a dummy address which is branched to when to current blit operation completes. This will be executed in the event the Blitter completes before the processor has completed the build of the next COPPER list. If that is the case, this temporary 'MOVE COPRETC' instruction will shut down the blitter channel. If the processor has completed the build of the next COPPER list, this instruction will be changed to a dummy 'MOVE NOP' instruction, and the following 'MOVE COPJMP3LC' and 'MOVE COPJMP3' will send the COPPER to the next COPPER list. The actual jump instruction address is created once the next copper list address becomes known.

While these copper instructions are being executed, the processor is off getting the next set of instructions ready. The blitter setup instructions are followed by the same seven special instructions included in the first copper list except of course that the address given in the COPBLITLC instruction should point to the most recent COPRETC instruction. Once all this is ready, the processor links the second copper list to the first by modifing the COPJMP3LC address in the first list to force a jump to the newly built copper list. To complete the link, the processor must follow a handshake procedure.

First, the processor overwrites the first lists' COPRETC instruction by making it a MOVE NO-OP instruction(RGA address 1FC). One of two things could have happened by this point in time. Either the original COPRETC instruction was executed by the copper before the processor overwrote it (because blitter has finished the first task), or the COPRETC has not been executed because the blitter is still running. If the blitter is still running, the processor can go on it's way and generate and append the next copper list to the end of the second, in the same way the second list was generated. If the blitter is not still running, the processor must restart the copper. It is possible to tell whether or not the COPRETC instruction has been executed by looking at the BLIT bit in the COINTRER register. Remember that the COPRETC instruction not only does a return from interrupt, but also disables copper blitter interrupt processing. At this point it is known that the copper must be restarted, but it is possible that between the time that the COPRETC instruction was changed to a MOVE NO-OP instruction and the time that the Blit bit of the COINTREQ register was examined, the second copper list may have executed, started the second blit operation, and that blit operation may have finished and be the one responsible for having reset the BLIT bit currently being examined by the processor (this is possible in saturated systems where processor access is rare). If the second blit is the one responsible, then the processor need not restart the copper. To tell which blit operation actually caused BLIT bit reset, the processor can now, at it's leisure, check the 'GENERAL' register. If the copper does need to be restarted to enable execution of the second list, it should be started following the same procedure used to start

the first copper list.

The number of blit operations which can be programmed this way is limited only by the amount of available RAM. All system activity will proceed normally including the copper's vertical beam counter and wait instruction processing. There is no longer a need to tear copper lists apart to insert new code in order to use the copper to control blitter operations.

5.7.2.6 Coprocessor Compatibility. -

When processing an old copper list, COPRET will never be strobed as it hasn't existed in past revisions of the chip set. The net result is that the coprocessor will always execute at either interrupt level 3 or 2. Each occurrence of vertical blanking will cause the vertical blanking interrupt level to be entered. The current copper program counter will be pushed onto the stack along with COP3LC and COP4LC and the value in COP1LC will be placed into the program counter. When the Vertical blanking routine is finished, a wait instruction will be executed which will schedule a wait interrupt. The copper will then drop into the idle state until the wait condition is met. When this occurs, execution will proceed with the next instruction of the copper list. This sequence will continue until another vertical blanking interrupt occurs. This time when the vertical interrupt routines' final wait instruction is executed, the wait interrupt previously scheduled will be overwritten and the wait interrupt flag will be cleared. The process is then repeated. Compatibility with old copper operation is retained.

5.7.3 Incompatibilities Between New Coprocessor And Old Coprocessor. -

It is important to realize that 100% coprocessor compatibility cannot be acheived. There are two reasons for this. 1)Burst mode fetches take 1.75 times as long to complete as standard bus cycles. This will delay coprocessor instruction execution an indeterminate amount of time to account for the additional .75 cycle times which can now occur as opposed to the old system. 2)High priority DMA requests can now be made at any time during the screen display. This means the coprocessor will incur delays at different points of the display line than were witnessed in the old systems.

Because of these reasons, it will be (almost)impossible to predict exactly when any given copper instruction will execute. This is not to say that the new coprocessor will be inadequate. By knowing what high priority DMA channels are enabled, a user can predict a window of time in which an instruction is guaranteed to have executed. Therefore, he can still effectively use the copper to control display line to display line screen changes, etc.

The following list shows all of the old RGA registers which could possibly be affected by the differences in execution timing between the old and new coprocessor. This list shows only those registers whose update can be considered time critical and those registers that make sense to change using the coprocessor. For example, the audio registers have a high degree of timing flexibility with respect to updates, and therefore havent been considered. As another example it doesnt make sense to use the coprocessor to drive the serial data transmit port. If one did use the coprocessor to drive XMIT, one would

have to worry about starting one transmission late, and the next transmission early thereby causing short stop bit problems.

BEAMCON0	DDFSTRT
BLTxPT	DDFSTOP
BLTxMOD	DIWSTRT
BLTAFWM	DIWSTOP
BLTALWM	DIWHIGH
BLTnDAT	DMACON
BLTnDATX	DSKDAT
BLTDDAT	DSKSYNC
BLTCON0	HBSTOP
BLTCON1	HBSTRT
BLTSIZE	HCENTER
BPLCON0	HHPOSW
BPLCON1	HSSTOP
BPLCON2	HSSTRT
BPLCON3	HTOTAL
BPLxDAT	INTREQ
BPLnPT	INTENA
BPLnMOD	SPRxPOS
CLXCON	SPRxCTL
COLORxx	SPRxDATA
COPJMPn	SPRxDATB
COPnLC	SPRxPT

List of registers which could present coprocessor compatibility problems.

5.7.3.1 Blitter Registers. -

Compatibility problems can be encountered when running old software which uses the coprocessor to control the blitter. There are two cases to consider. Case 1 is when the timings between the coprocessor and blitter execution are so closely matched that the coprocesor is used to make mid blit modifications to the blitter control registers to, for example change the next blit line's modulos registers. Any program which attempts this kind of operation on the new chip set will be broken. It is highly unlikely that existing software would program this type of copper/blitter interaction. Case 2 would be a more orthodox usage of the coprocessor in controlling the blitter. A series of blits which will effect the display are to be done to accomplish an animation. The coprocessor has been set up to control this succession of blit operations in such a way that the blitter is racing 'just in front of' the display beam. Because it cannot be guaranteed that the coprocessor is going to start execution at precisely the xy coordinate its WAIT command was programmed for, any given blit operation can be delayed from starting vs when it would have started in past chip sets. In the extreme worst case, this delay could be as much as 11.48usec, or 82 lowres pixels assuming 6 lowres bitplanes, 8 active sprites, 4 active audio channels, the floppy is being read, and the WAIT xy coordinate was such that the copper was to come 'alive' just at the end of horizontal blanking. In this case there is a chance that the blit will not be finished by the time the beam gets to the data it needs. This chance is small however because in an old system under the same situation, (ie 6 bitplanes, etc.) there are only 89 cycles available to the blitter each scan line, but in the new system there would be 189 cycles available each scan line so the blit would finish much faster. In this case, there is more chance of

software failure because of increased memory bandwidth than because the copper started late.

5.7.3.2 Display Registers. -

The most obvious place to look for problems arising from copper instruction timing related incompatibilities is the display. BEAMCON0, BPLxDAT thru CLXCON, DDFSTRT, DDFSTOP, DIWSTRT, DIWSTOP, DIWHIGH, HBSTOP, HBSTRT, HCENTER, HHPOSW, HSSTOP, HSSTRT, HTOTAL, and SPRxPOS thru SPRxPT listed above control old chip display characteristics. Before examining the role played by the coprocessor instruction execution timing, a review of the new graphics system is necessary. Graphics data is fetched from RAM one line in advance of the line it is to be displayed. Each bitplane line is either burst mode fetched or shifted from VRAM shift registers into a line buffer one plane at a time to buy the increased bandwidth of the access mode. This means that graphics data is pipelined one display line. Similarly, Sprites are also prefetched one line in advance and buffered until the next display line. What this implies is that no midline changes can be made to graphics data or to sprite data. Each sprite can only be used once per display line, although it can still be reprogrammed for reuse on another vertical line. In order to guarantee a stable display, most of the display control registers are double buffered so that they may be changed to affect the next line without affecting the display of the current line. This means that any old software which makes midline horizontal changes to display attributes to change that display line will be broken. The only display control registers listed above which are not doubled buffered are BPLnDAT, and COLORxx.

It is still possible to write to the bitplane data registers (although this action is highly discouraged). Any program which uses the copper to modify any of the BPLnDAT registers at the same time graphics DMA fetches are being done by the chip set will be broken for the following reason. Data is transferred from LINDA to MONICA 32 bits at a time. Old coprocessor instructions which write to these registers will only change 16 bits of the register, and unless the copper write is timed perfectly, the data it deposits will be overwritten with the next copper write to the register. (Before, it was assumed that the first 16 bits would be gone by the time the second write is done.)

It would appear as though the most incompatible part of the new system is the new graphics. This is not because of coprocessor timing incompatibilities but because of characteristics of the new graphics hardware. The majority of old software sets up a stable display and makes no changes to any of the display registers during the entire frame. This software will run OK on the new chip set. A much smaller percentage of the software will set up line to line changes in display attributes. This software will also run OK. The remainder of the software which makes mid-line display changes will be broken.

5.7.3.3 Coprocessor Registers. -

It is possible to have compatibility problems with respect to the COPJMP and COPnLC registers. This can happen when the 680n0 and the coprocessor are running in unison. If the 680n0 has been set up to modify either of these two registers within several microseconds after the copper itself has strobed one of the COPJMP registers to effect a 'GOTO', then there is a chance the program will crash when run on the

new chip set.

5.7.3.4 DMACON Register. -

Its possible that old software uses the coprocessor to enable some of the DMA channels during segments of the display line. The channel most susceptible to problems in light of this action is the blitter DMA channel. If the blitter channel is programmed by the coprocessor to be active for only a few microseconds each display line, just after horizontal blanking, then there is a chance the blitter will run slow or not at all in the new system, depending on the graphics mode selected and the exact interval for which blitter DMA is enabled.

5.7.3.5 Disk Registers. -

Old software which uses the coprocessor to change some of the disk registers may have compatibility problems. If the software used the copper to modify the DSKSYNC register and in unison use the 680n0 to enable disk reads for some elaborate protection scheme, then its likely that program will be broken. In the same respect, it is questionable whether or not this could have been made to work in the old system.

5.7.3.6 Interrupt Request And Enable Registers. -

If the system has been programmed to process time critical interrupts (windows of less than several micro seconds) generated by the coprocesor, then there is a chance of system failure when that software is run on the new system. It is questionable whether or not this could have been made to work in the old system.

6 GRAPHICS SUBSYSTEM.

This chapter covers some of the graphics design issues which have been encountered over the course of the AAA chip set development.

6.1 Pixel Clocks.

In the lowend system, the pixel clock supplied to LINDA and MONICA has the same frequency as the monitor pixels. In the high end system there are dual LINDA and MONICA chips and the pixel clock supplied to LINDA and MONICA is half the frequency of the monitor pixels. In the high end system each LINDA/MONICA pair processes every other pixel in parallel. The even (as specified by the EVEN/ODD input pin) LINDA/MONICA pair extracts and processes even pixels, while the odd LINDA/MONICA pair extracts and processes odd pixels. This maintains a relatively low clock rate in both low and high end systems. This section shall cover the way in which LINDA and MONICA processing is adjusted to preserve a constant software model between low and highend systems.

6.2 Lowend System Graphics.

Lowend systems graphics are processed in much the same way that Denise currently processes graphics. The primary difference between the new chip set and the old chip set is that ANDREA now prefetches graphics data a line in advance and temporarily stores the data in LINDA. (Note: this phenomonum is invisible to systems software. Both the host microprocessor and custom coprocessor have the same graphics data time reference that existed in the old systems. This has been accomplished by pipelining vertical events to both processors by one scan line.) This allows RAM chips to be cycled as fast as possible using either Fast Page mode or VRAM shift cycles. More graphics data can be fetched in this way. All display data sent to LINDA is captured and held until the next display line. When the next display line rolls around (marked by DFTCHWIN), LINDA presents the data to MONICA as described below. MONICA then processes the display data in much the same way DENISE processes her data. Pixels output from MONICA can be either analog or digital RGB. Eight bits of color resolution is maintiained in both cases.

6.3 HighEnd System Graphics.

In the highend system, there is a pair of LINDA/MONICA chips, one pair processes even pixels, while the other pair processes odd pixels. As far as the MONICA chips are concerned, normal display information from LINDA is processed as though MONICA were in a low end system. Each LINDA however is only capturing every other pixel. In this way, each MONICA gets only the data she is concerned with. The aligner circuitry in LINDA performs the function of even or odd pixel extraction (With the exception of packed pixel modes. In these modes, pixel extraction is performed by the OUTMUX circuitry. See the section 7.4, 'Pixel Modes'). The MONICA chips can be set up such that both MONICA chips output their pixel data in parallel. This mode is useful when it is necessary to add some sort of additional post-processing circuitry to the system. Normally, the MONICA chips are set up such that the even MONICA passes it's digital pixel data to the odd MONICA

chip's digital pixel port. In this mode of operation, the odd MONICA multiplexes pixels from both sources, and then converts the serial data stream to analog form to drive the monitor directly.

Terminology: the odd LINDA/MONICA pairs are those chips which process the odd bits as they come from RAM (ie bits 63, 61, 59, ..., 1). Bit 63 is displayed first. The even LINDA/MONICA pairs are those chips which process the even bits as they come from RAM (ie bits 62, 60, 58, ..., 0).

6.4 SPRITES In High End Systems.

There are no RGA addresses which talk to only the EVEN MONICA or only the ODD MONICA of a high end system. Therefore, SPRITE data is loaded into both of the MONICA chips during the same CHIPDATA bus write operation. MONICA's 'EVEN/ODD' input, the LSB of the SPRITE horizontal position compare register, and the enabled SPRITE resolution (HIRES or LORES) determines whether MONICA should process the even or the odd SPRITE pixels written into the SPRDAT registers. (see terminology mentioned in section 6.3, 'HighEnd System Graphics'. Bits 31, 29, ..., 1 are odd pixels and SPRITE bit 31 is displayed first. Bits 30, 28, ..., 0 are even pixels). Of course if MONICA's 'DUAL/SING*' input pin tells her that she is in a lowend system, then she will process all of the SPRITE pixels.

6.5 Monica RGA Read Addresses In The HighEnd System.

In order to retain software compatibility between lowend and highend systems, the fact that there are a pair of LINDA-MONICA chips in the highend system must be invisible to software. As mentioned above, both MONICAs react to all RGA addresses. MONICA has 258 RGA read addresses. These are CLXDAT(Collision data read register), COLORxx(color lookup table registers), and MONICAID. The COLORxx and MONICAID registers present no problem since when these registers are read both MONICA chips will supply the same data and no bus contention will occur. The CLXDAT register presents a different problem. Since each MONICA is processing every other SPRITE pixel, it is possible that collisions can occur in one chip that don't occur in the other. This means that each MONICA wants to drive different data onto the bus when the CLXDAT register is read. The contention problem is solved using 'wired or' logic in four steps corresponding to each half BUSCLK cycle that ALE is low as follows.

first half cycle: nothing occurs (MONICA allows RGA's to get off bus)
second half cycle: MONICA hard drives all databus pins low.
third half cycle: MONICA enables soft latch circuitry on all pins, and hards drives all asserted collision bits high.
fourth half cycle: MONICA duplicates assertions made during half cycle three.

6.6 Horizontal Counters.

When programming the old Amiga chip set, programmers were not aware that there was more than one physical horizontal counter in the system. The software model that was presented indicated that all

horizontal events were based off the same counter. This is not really the case. Their are two horizontal counters in the old chip set, and there will be two in the new system. The first reaction is "so what. The two counters were synchronized in the old system with one of the STREQU, STRVBL, or STRHOR RGA strobes. What's the big deal?". The chips were brought into sync with RGA strobes. Even though these RGA strobes do not exist in the new chip set, their functionality is preserved with the new HRESET signal. The major difficulty is that in the old chip set, at any given clock period, Denise's counter did not contain the same value as Agnus's counter. On top of that, Denise's counter was clocked at twice the frequency as Agnus's counter. A timing diagram in Appendix A depicts the difference which existed in the old chip set counters. These facts present compatibility issues. Old software has been written to accomodate the differences between the two counters when programming such things as the display window region and Sprites.

We have considered ignoring the old difference in counter values and design the new chips such that both counters are always in sync with the same value and frequency. This action would violate one of the design contraints mentioned in chapter 2- 'No Orphaned Software'. It is felt that a significant number of old software packages would break if we chose this alternative.

We have also considered designing the new chips such that both counters are always in sync with the same value and frequency, but containing additional compensation circuitry so that the end responce is the same as that exhibited by the old chip set. This alternative is too costly.

One alternative remains. The new chips will be designed to always be in sync, but contain a count difference that provides the same behavior that existed in the old chip set. It turns out that it is irrelevent that the old Denise and Agnus counters run at different frequencies. Failing to mimic this facet of the old chips will not adversely effect new system behavior.

Monica's counter is synchronized to Andrea's in a software compatible manner as follows. Andrea's HRESET pin is asserted for one PCLK cycle during the PCLK cycle that Andrea's counter contains a value of 8. (The hardwired compatibility line increment decode which occurs at count 7 of every line is used to generate HRESET). Monica accepts HRESET from Andrea and then counts an additional fourteen PCLK cycles. The next PCLK cycle after that, Monica presets her horizontal counter to a value of 4. The end behavior is that Monica's counter achieves counts which Andrea never see's, and Monica never see's counts 0, 1, 2, or 3. When Andrea's counter is 23 (17 hex), Monica's counter is 4. Except for the increased resolution (which has no detrimental effects), this is the same behavior exhibited by the old chip set.

(Editors note: The preceeding discussion explains the ideal situation to use to synchronize ANDREA's and MONICA's horizontal counters. When the circuits were implemented, it was realized that if this behavior were implemented, then it would be necessary for MONICA to perform horizontal comparisons on odd PCLK half cycle boundaries in dual systems, and on even PCLK half cycle boundaries in single systems. To make the chip logic design manageable, the final chip design implemented the following: Monica accepts HRESET from Andrea and then counts an additional thirteen PCLK cycles. The next PCLK cycle after that, Monica presets her horizontal counter to a value of 4. When Andrea's counter is 22 (16 hex), Monica's counter is 4.)

6.7 Graphics Programming.

The AAA graphics circuits have been designed to be compatible with pre-ECS Amiga chips. When the system monitor is an NTSC or PAL type monitor, the circuits can be programmed with the same graphics control register constants used to program pre-ECS chips.

6.7.1 Programming The Display Control Registers. -

The first parameter which should be established when setting up a display is the length of each scan line. The HTOTAL register is programmed with a number such that the inverse of this number times the pixel clock cycle time is equal to the display's spec'ed horizontal frequency. The HCENTER register can be programmed once HTOTAL is known. This register is set to half HTOTAL.

Horizonatal sync should be established next. For proper operation, it is rcommended that horizontal sync be made to start at pixel line position 0. For compatibility with old software, horizontal sync can be programmed to occur later in the scan line as defined by older Amiga chip sets. To avoid hardware error, horizontal sync should never be programmed to occur outside of this range. The monitor specification should be consulted to determine the proper duration of horizontal sync.

After horizontal sync has been defined, the horizontal blanking region is defined. The position and duration of this region relative to horizontal sync is dependent on the monitor being driven. Typically, horizontal blanking occurs before horizontal sync starts, and ends after horizontal sync is deasserted. Horizontal blanking causes the display hardware to output the color black within horizontal blanking intervals.

With horizontal sync and horizontal blanking defined, the display window can be defined. This window (controlled by the DIWSTRT and DIWSTOP registers) defines the number of pixels to be displayed each scan line, and where on the scan line (relative to horizontal sync) the pixels are to be displayed (normally centered within two sucessive horizontal sync pulses). When the display beam is outside of the horizontal blanking region but not yet within the display window, the display hardware outputs the system background color (LUT register 0). To the display hardware, the DIW registers define a window in time during which the monitor color guns are enabled to drive display data other than blanking (black) or background color (LUT register 0).

When an NTCS or PAL monitor is being programmed for compatibility with existing software, certain other registers must also be programmed. Section 5.3, 'Interlaced NTSC and PAL Display Modes', contains a table defining other register values which need to be programmed in order to replicate the hard-wired graphics compare values which existed in pre-ECS chips.

The vertical timing control registers are set up in a manner analogous to the procedure described above for the horizontal control registers. Special precautions should be taken when programming the VBSTART and VBSTOP registers. The VBSTOP register must be programmed with a value other than zero, and should be programmed to occur before the start of display window. VBSTRT should be programmed to occur after the end of the display window, and at line zero (it's hardwired location in old Amiga chips) or just before line zero at the end the

last display field. The reason for these restrictions is related to Sprite DMA fetches when the Sprite(s) are enabled to display anywhere (when the CONSPRT bit of any of the SPRxCTRL registers is set). On short fields (LOF==0), and when a CONSPRT bit is set, any Sprite DMA which occurs during the vertical blanking region will have a full horizontal line time to be serviced. On long fields between lines zero and VBSTOP, any Sprite DMA request which is made will not occur until the middle of the scan line. Therefore, in this region of the display, Sprite DMA has only half the normal amount of time in which to be fetched before an overrun condition occurs. As long as display data is not actively being fetching (normally the case during vertical blanking), there should not be a problem allocating enough cycles to make all the fetches. In the extreme worst case, (when eight 128 bit sprites are enabled and 'displayed' during the same line of vertical blanking), all fetches will take less than 6 usec.

6.7.2 Programming DDFSTRT And DDFSTOP. -

For compatibility purposes, the DDFSTRT and DDFSTOP registers have been retained in the new chip set. An additional bit of precision has been added to these registers in the new chip set. Aside from this extra precision, when in bitplane mode, and when the number of bitplanes is less than 9, (when in the compatible display mode) the DDFSTRT, DDFSTOP, DIWSTRT, and DIWSTOP registers are programmed with exactly the same data which was used to program past Amiga chips. (See chapter 3 of the Amiga Hardware manual for a detailed discussion of how to set these and the DIWSTRT and DIWSTOP registers for a display compatible with old software.)

To the display hardware, the DDFSTRT and DDFSTOP registers define a window in time during which display data is accessed from the Line Data Buffer IC (LINDA) and sent to the Monitor Control IC (MONICA). They are programmed relative to the display window (DIW) registers and relative to the type of pixel to be displayed. DDFSTRT is set equal to DIWSTRT minus an appropriate pipeline delay value (modulo HTOTAL) based upon the requested display mode. Section 6.7.4, 'Programming the DDF and DIW Registers in Chunky Pixel Modes', gives a table which identifies the proper delay value to use when calculating DDFSTRT.

The DDFSTRT and DDFSTOP registers do not physically contain a bit for the least significant bit H00, therefore it is only possible to define the fetch window to within two highres pixels. The least significant bit is always forced low by hardware. Once DDFSTRT has been established, DDFSTOP is determined by adding the number of pixels to be displayed each line to the value of DDFSTRT and subtracting 32. (Note that if the hardware has been placed in low resolution pixel mode, then the value used in the number of pixels to be displayed field of this equation should be multiplied by two. I.E. this calculation is done using highres pixel precision.) There are two restrictions to keep in mind when programming DDFSTRT and DDFSTOP (these restrictions do not apply to the DIW registers): 1) The number of bits fetched from the LINDA IC each scan line must be an integral number of 16. 2) The display data fetch window cannot be narrower than 128 bits (bits not pixels), low res or high res. This means all display windows as they are stored in memory must be at least 128 bits wide.

6.7.3 High End System Graphics Programming. -

Most of the graphics control registers are programmed with identical data to achieve identical displays when targeted at identical monitors, irrespective of whether the system contains single or dual graphics chips. The exceptions to this rule are the DDFSTRT and DDFSTOP registers when the display mode is bitplane. Due to extensive graphics circuit pipelines, values programmed into these registers in dual graphics chip systems cannot be the same as the data programmed in single graphics chip systems or pre-ECS systems. When displaying bitplane graphics in a high end system, software must subtract the constant 48 from the DDFSTRT and DDFSTOP values which were used to program past Amiga chips or AAA single graphics chip (low end) systems. System software can determine whether the system is low-end or high-end by polling the chip identification field of the VPOSR register. It is important to realize that the constant given above is in terms of the new resolution provided by the AAA DDFSTRT and DDFSTOP registers; it is in terms of high resolution pixels. (Note, the H00 bit positions (the LSB) in DDFSTRT and DDFSTOP registers are forced low by the hardware).

Examination of the new monitor control registers (ex HSSTRT) indicates that most monitor signals can be programmed to within one high resolution pixel. This is true in single graphics chip systems. It is not true in dual graphics chip systems. If the monitor control registers are programmed to start or stop a signal on an odd (the H00 (or LSB) bit = 1) pixel boundary, the resulting timing will be truncated down to an even pixel boundary in dual graphics chip systems. For this reason, it is recommended that software always program monitor control signals to start and stop on even high resolution pixel boundaries.

6.7.4 Programming The DDF And DIW Registers In Chunky Pixel Modes. -

When one attempts to program the AAA chips to display of any of the new AAA chunky pixel modes, no compatibility issues need to be considered. This is because chunky pixel modes have never existed before in any Amiga systems. This fact permitted the chips to be designed to support a new DDF programming model when chunky pixels are displayed. The new model is much simpler than that which must be used to set up bitplane displays: if the pixel type is not equal to bitplane, then
DDFSTRT should be set to
DDFSTRT = DIWSTRT - 40,
and DDFSTOP should be set to
DDFSTOP = DDFSTRT + (# of pixels to display) - 32.
All variables and constants are in terms of high resolution pixels. For example, to display a high resolution 640 x 200 NTSC screen in the old system, the horizontal field of the DIWSTRT register is programmed with 0x81 (hex) and the DDFSTRT register is programmed with 0x3C (hex). (Both given in terms of the resolution provided by the pre-ECS chips.) In terms of the new resolution available to software, the difference between these values for DIWSTRT and DDFSTRT is
1000 0001 0 (binary DIWSTRT) - 11 1100 00 (binary DDFSTRT) =
^ this bit (H00) is forced low by hardware, the next LSB, H01 is in bit position 15 of DDFSTRT.
0x102 (hex DIWSTRT) - 0xF0 (hex DDFSTRT) = 0x12 = 18 high res pixels.
This difference is the pixel difference apparent to software, and not the actual hardware pixel difference (see section 6.6, 'Horizontal

Counters').

The following table defines how DDFSTRT and DDFSTOP are programmed relative to DIWSTRT for all available display modes. As in previous Amiga chip sets, DIWSTRT is programmed relative to Hsync so as to center the screen.

Pixel Mode	DDFSTRT	DDFSTOP
Bitplane, < 9 planes, highres, single	DIWSTRT - 18	DDFSTRT + LL - 32
Bitplane, < 9 planes, lowres, single	DIWSTRT - 34	DDFSTRT + (LL*2) - 32
Bitplane, < 9 planes, highres, dual	DIWSTRT - 66	DDFSTRT + LL - 32
Bitplane, < 9 planes, lowres, dual	DIWSTRT - 82	DDFSTRT + (LL*2) - 32
Bitplane, > 8 planes, highres, single	DIWSTRT - 34	DDFSTRT + LL - 32
Bitplane, > 8 planes, lowres, single	DIWSTRT - 66	DDFSTRT + (LL*2) - 32
Bitplane, > 8 planes, highres, dual	DIWSTRT - 114	DDFSTRT + LL - 32
Bitplane, > 8 planes, lowres, dual	DIWSTRT - 178	DDFSTRT + (LL*2) - 32
All other highres graphics*	DIWSTRT - 40	DDFSTRT + LL - 32
All other lowres graphics*	DIWSTRT - 40	DDFSTRT + (LL*2) - 32

LL- Length of each Line to be displayed in pixels.
* All other includes variations in system (dual or single), and variation in pixel mode (Hybrid, PACKLUT, PACKHY, Chunky, HalfChunky, 4-bit HalfChunky, and 2-bit HalfChunky).

6.7.5 Programming High Resolution Monitors; -

Monitor control constants which existed in the pre-ECS chips were (apparently) arbitrarily chosen. For example, in NTSC, HBSTRT occured at the 50th pixel position of the scan line, HSSTRT occured at the 72nd pixel position, and the last display pixel occured at the 898th pixel position. When using the new chips to drive monitors other than NTSC or PAL, it is recommended that this methodology change such that Hsync is programmed to occur at pixel position 0. When this is done all other control signals must be skewed to match the Hsync skew.

Hardware restrictions exist in the graphics chips which govern Hsync signal placement within the scan line. When the values given in section 5.3, 'Interlaced NTSC and PAL Display Modes', are used to set up a graphics display, the sync signals are positioned as far to the right on the scan line as possible without causing hardware errors in any of the display modes. Moving the Hsync and Hblanking signals further to the right would result in the line buffer chip receiving it's display control data too early and possibly cause the end of the display lines to be treated improperly. (The line buffer chip always receives it's control data starting at scan line pixel position 11. Fifteen bits of information are transferred at a 28Mhz rate. In an NTSC system, the transfer is complete by pixel position 26. In a high-end 1280 x 1024 system, the transfer is complete by pixel position 41.) There is a similar restriction in the other direction. Errors may occur if the start of Hsync were moved so far to the left on the scan line that it crosses scan line boundaries and occurs at the end of scan lines. In order to prevent problems of this type, the monitor control signals should be set so that the scan line is blanked during the interval that control data is sent to the line buffer chip. In addition, software must guarantee that DDFSTRT does not occur until this transfer is complete.

6.7.6 Graphics Data Addressing Calculations Made By Hardware. -

The following equations summerize how the chips process the bitplane pointer registers, and how the chips determine the number of DRAM accesses to make when fetching display data.

DDFSTOP and DDFSTRT in the following equations are scaled to highres pixels (i.e. H00 is appended and forced to 0).

$$BPP * \frac{DDFSTOP - DDFSTRT + 32}{16} = \begin{cases} NWORDS, & HIRES = 1 \\ NWORDS * 2, & HIRES = 0 \end{cases}$$

where BPP - Bits Per Pixel
- 1- bitplane mode
- 2- Two Bit Half Chunky mode
- 4- Four Bit Half Chunky mode
- 8- Half Chunky mode
- 8- Hybrid pixel mode
- 8- Packed LUT mode
- 16- Chunky mode
- 16- Packed Hybrid mode

NWORDS - Number of 16 bit words to fetch
HIRES - HIRES bit of BPLCON0 register; high resolution

NWORDS is right shifted once if graphics mode low resolution. Use BPLnPT as the start address for each graphics fetch. Because these are byte addresses but the system data bus is 32 bits wide, bit 1 and possibly bit 2 (dual systems) will be forced low when driven to system RAM. (Also note the alignment restrictions mentioned in chapter 7 for each of the pixel types). At the end of graphics fetches,

```
BPLnPT = BPLnPT + BPLmMOD + (NWORDS * 2)
```

The following algorithm is used to determine the number of RAM accesses (or shift cycles in the case of VRAM) to make. (Use BPLnPT before above calculation is done.)

```
if (single) then
    NSHIFT = 1;
    EXTRA = NWORDS(bit0) | BPLnPT(bit1);
end;
if (dual) then
    NSHIFT = 2;
    EXTRA = NWORDS(bit0) | NWORDS(bit1) | BPLnPT(bit1) | BPLnPT(bit2);
end;

NACCESSES = (NWORDS / (2**NSHIFT)) + EXTRA
```

where:
- NSHIFT - an exponent used to scale NWORDS to match the size of the data bus in the system.
- single - indicates the chips are in a low end system (DUAL_SING pin = low)
- dual - indicates the chips are in a high end system (DUAL_SING pin = high)
- EXTRA - Set to indicate the video data is not an integral number of data bus width accesses, and therefore one addition cycle must be run.
- NACCESSES- Number of CAS accesses or shift clock cycles to run.

7 LINDA.

Graphics LINe DAta buffer chip.

7.1 Purpose Of LINDA.

LINDA has been conceived and specified for the sole purpose of providing high performance video graphics. Graphics resolution of the original AMIGA system was severly limited by the amount of graphics data which could be fetched from system memory. The new system is capable of generating much higher graphics resolution than the old system. A lowend configuration using fast page mode RAM can generate displays of up to 800 x 560 x 9 bitplanes. Using VRAM, the same configuration can generate 800 x 560 x 13 bitplane displays, or it can display HYBRID pixels at 800 x 560 to provide 24 bit per pixel, true color displays. A highend configuration can do more. Using Fast Page Mode RAM it can put up 1280 x 1024 x 5 bitplane displays. When VRAM is used, 8 bitplanes can be displayed at 1280 x 1024 and HYBRID pixels can be displayed on 1024 x 768 monitors.

7.2 Basic Functionality.

LINDA is one of three chips which make up the Advanced Amiga graphics sub-system. The line buffer function which LINDA provides permits the use of advanced DRAM access cycles (such as Fast Page Mode cycles or shift register VRAM cycles) to fetch graphics data from between four and eight times faster than would be possible without the special cycles. With the advanced cycles, more graphics data can be accessed in the same period of time, so that more data can be displayed to acheive higher resolution, or more color, or both. In addition to the improvement in graphics resolution, the line buffer permits a high degree of compatibility with software written for the A1000. Such compatibility could not be retained if other techniques such as use of VRAM or dedicated frame buffer storage were employed.

In addition to the line buffer function mentioned above, LINDA provides other functionality.

An aligner is included in LINDA's input circuitry to support the word alignment restriction which existed in the A1000.

LINDA decodes packed pixels (a new display mode) to provide animation capabilities.

The direction of data flow thru LINDA can be reversed so as to support real time, low cost framegrabber applications.

LINDA is a slave to one of the other chips in the graphics sub-system, ANDREA. ANDREA tells LINDA which mode to process in, when to latch input data, and when to fetch data for transfer to the next chip in the graphics sub-system, MONICA. There are two independent line buffer RAMs in LINDA. At any given time, ANDREA is controlling the input of graphics data into one of the RAM buffers, and at the same time controlling the output of graphics data from the other RAM buffer. At the beginning of the next scan line, ANDREA will command LINDA to switch RAM banks, and then fill the bank which was just accessed for data to send to MONICA with data for the next scan line. At the same time, ANDREA will command LINDA when to begin sending data to MONICA

from the other RAM bank. This process continues until the vertical blank interval at which time ANDREA commands LINDA to send nothing to MONICA. At the beginning of the next screen, the whole process is started over.

LINDA is completely dependent on ANDREA for signals as to when to begin processing graphics data, and when to latch input data. Once started, LINDA will maintain all the necessary states and conditions to continue processing data correctly and will do so until ANDREA informs her that it's time to stop.

LINDA has been designed to accept each of the pixel types described in section 7.4, 'Pixel Modes', and process them properly. LINDA is responsible for buffering graphics data, in some cases manipulating graphics data, then accessing the buffered data on the next monitor scan line in longword (32 bits) fetches for presentation to MONICA.

7.3 Framegrabber Mode.

LINDA contains all the buffer circuitry necessary to support Framegrabber mode. See section 4.10, 'Framegrabber Interface', for a complete description of this mode.

7.4 Pixel Modes.

This section gives a detailed description of the pixel modes which have been incorporated in the custom chip set, and in particular how LINDA processes data for each of the display modes.

7.4.1 Bitplane Mode. -

The chipset supports from 1 to 16 bitplanes. Up to 8 bitplanes can be used to address a 256 x 25 bit color lookup table. MONICA supports the original even and odd playfields. When dual playfield mode is enabled, MONICA priority logic sends either the even playfield or the odd playfield to the LUT. When BPU of BPLCON is greater than eight, when dual playfield mode is disabled, and when HAM mode is disabled, only bitplanes 1 thru 8 are used for display generation. When BPU of BPLCON is greater than ten, when dual playfield mode is disabled, and when HAM mode is enabled, only bitplanes 1 thru 10 are used for display generation.

ANDREA generates all necessary addresses and control signals to control the fetch of bitplane data from RAM. ANDREA specifies when valid graphics data is present on the GDATA bus for capture by LINDA. One scan line of bitplane data is fetched in it's entirety before moving on to the next. SHFTCLK provides the clock used by LINDA as she is capturing and processing GDATA. ANDREA asserts SCACT each SHFTCLK cycle that the VDATA bus contains valid data LINDA is to process. ANDREA asserts NXTPLANE (see timing diagram in Appendix A) to indicate to LINDA that all of the current bitplane has been fetched and the next GDATA is a member of the next bitplane. LINDA writes aligned input data into sequential locations of RAM, skipping to the next course segment boundary (each course segment contains 20 64 bit double long-words; enough storage to hold the biggest bitplane LINDA will

process) with each NXTPLANE pulse.

Plane data is transferred from LINDA to MONICA one 32 bit longword of plane data per transfer. PCLK provides the clock necessary to drive all of LINDA's circuitry associated with this transfer activity. ANDREA asserts DFTCHWIN as an indicator of when LINDA is to begin sending bitplane data to MONICA. Data is accessed from the line buffer using a 'fetch one, skip nineteen' addressing scheme. LINDA asserts CAPEN for each PCLK cycle she is driving valid data onto the GDATA bus and transferring to MONICA. See timing diagrams of Appendix A. MONICA latches this data with the next rising edge of PCLK in the BPLnDAT to BPL1DAT parallel to serial conversion registers. Transfers are always done 32 bits at a time starting with the highest numbered active plane (indicated by BPU) and ending with a transfer of bitplane 1 data. LINDA is responsible for keeping track of which bitplane data has been sent, and which bitplane data remains to be sent. ANDREA is responsible for sequentially loading into LINDA, a scan line in advance of when the data is to be sent to MONICA, each of the bitplanes beginning with the highest numbered bitplane. LINDA is responsible for timing out a certain number of PCLK cycles from the rising edge of DFTCHWIN before the transfers begin. The number of PCLK cycles which must elapse is dependent on the number of active bitplanes, BPU (see timing diagrams of Appendix A). MONICA is responsible for keeping track of which BPLDAT register is to be loaded next. She does this using BPU of BPLCON. After HRESET, the first register loaded is indicated by BPU. A copy of BPU is decremented each time another register is loaded. This decrementor always points to the next register to be loaded. When the decrementor reaches a value of one, it is reloaded with BPU. The display continues in this way until the end of the display line.

LINDA's input and output circuitry operate independently and asychronously to each other. The input and output circuitry also operates concurrently. In other words, as new data is being loaded into one of LINDA's line buffer RAM's, the data which was loaded into LINDA's other line buffer during the previous scan line is being read out and transferred to MONICA. At the beginning of each scan line as indicated by DFTCHWIN, the line buffer RAM banks are switched, so that the next scan line can be loaded into the buffer which was just sent to MONICA, and the buffer which was just filled can be sent to MONICA.

To support old software, bitplane data can be word aligned in memory. LINDA's aligner circuitry is used to realign misaligned bitplane data and remove unnecessary prefix data before it is stored in one of the line buffer RAM's.

7.4.2 HALF CHUNKY Pixels. -

Half Chunky pixels are 8 bit pixels stored linearly and used to address MONICA's LUT. Half Chunky pixels are transferred from LINDA to MONICA four at a time. The most significant byte is displayed first. See timing diagrams in Appendix A. Alignment of Half Chunky pixels in memory is restricted to double long-word alignment (64bits). This simplifies LINDA's aligner circuitry and does not present compatibility problems since Half Chunky pixels are a feature new to this chip set.

All of the interactions between ANDREA, LINDA, and MONICA for processing Half Chunky pixels are essentially the same as those described above in bitplane mode. (This is also the case for Four-bit Half Chunky, Two-bit Half Chunky, and Chunky pixels.) The differences

are as follows. 1) LINDA writes and then reads line buffer data using a sequential addressing scheme. 2) ANDREA will only assert NXTPLANE once per scan line. This is because all of the necessary video data can be burst fetched (in the case of Fast Page DRAM, or shifted into LINDA in the case of VRAM) as the result of one RAM RAS-CAS fetch cycle. (There is a case where a RAM page boundary is crossed during the fetch sequence which requires two RAS-CAS fetch cycles. ANDREA accomodates the break in data transfer which may occur as a result of the second RAS-CAS cycle by deasserting SCACT for those cycles where no data is to be latched and processed by LINDA.) 3) The delay which LINDA must process between the time DFTCHWIN is asserted and the time LINDA drives the first four pixels onto the GDATA bus is different for half chunky pixels than the delay needed to transfer bitplane pixels. See timing diagrams of Appendix A.

7.4.3 Four-Bit HALF CHUNKY Pixels. -

Four-Bit Half Chunky pixels are similar to Half Chunky pixels. They are 4-bit pixels stored linearly and used to address MONICA's LUT. Four-Bit Half Chunky pixels are transferred from LINDA to MONICA eight at a time. The most significant nybble is displayed first. Alignment of 4-Bit Half Chunky pixels in memory is restricted to double long-word alignment (64bits).

7.4.4 Two-Bit HALF CHUNKY Pixels. -

Two-Bit Half Chunky pixels are 2 bit pixels stored linearly and used to address MONICA's LUT. They are transferred from LINDA to MONICA sixteen at a time. The most significant two bits are displayed first. Alignment of Two-Bit Half Chunky pixels in memory is restricted to long-word alignment (32bits). This display mode uses long-word alignment as opposed to the double long-word alignment used in the other half-chunky modes in order to support scrolling with four scroll control bits.

7.4.5 CHUNKY Pixels. -

Chunky pixels are 16 bit linear pixels. Unlike the half chunky pixel types, these pixels bypass MONICA's LUT and drive the monitor color guns directly. Chunky pixels contain 5 bits of red info (bits 14-10), 5 bits of green info (bits 9-5), 5 bits of blue info (bits 4-0), and 1 ZD bit (bit 15) for genlock capabilities. Chunky pixels are transferred from LINDA to MONICA two at a time. Each color field is scaled to the most significant portion of MONICA's color data paths as this is done. Alignment of Chunky pixels in memory is restricted to double long-words (64bits).

7.4.6 HYBRID Pixels. -

As the name implies, these pixels are a cross between chunky pixels and bitplane pixels. They allow the display of 24 bit pixels. The Hybrid approach to 24 bit pixels overcomes the packing/unpacking problem associated with true 24 bit chunky pixels. Hybrid pixels are stored as three 'half chunky' planes, each plane describing one of the

colors red, green, and blue. LINDA is responsible for extracting colors from each of the three color longwords and assembling them into 24 bit pixels (the most significant bytes are extracted and sent to MONICA first). The pixels are in turn sent to MONICA, one at a time, one every pixel clock cycle when the display mode is highres, and one every other pixel clock cycle when the display mode is lowres as specified by the CAPEN output control line. See timing diagram in Appendix A. The least significant 24 bits of the VDATA bus contains the pixel data. Bits 23-16 contain the red data, bits 15-8 contain the green data, and bits 7-0 contain the blue data. Bits 31-24 should be ignored.

ANDREA asserts NXTPLANE three times as each of the three color planes are fetched from memory for storage in LINDA. The red color plane is fetched first, followed by the green then blue color planes. LINDA stores each of the color planes using a 'write one location, skip two' addressing scheme. This approach simplifies subsequent pixel assembly, by alternating sequential double long-word buffer locations with red, green, and blue data. When LINDA sends a scan line of data to MONICA, she sequentially accesses three double long-words and temporarily stores this red, green, blue data in holding registers within the OUTMUX circuitry. From these registers, LINDA assembles eight 24 bit pixels and sends them to MONICA along with the usual qualifying CAPEN signal.

7.4.7 Packed Pixels. -

Packed pixels form a compressed screen image which when decompressed and displayed provide a high degree of color resolution, but at a fraction of the storage requirements. Packed pixel data is compressed as follows. The screen is segmented into 4 x 4 pixel regions. Two colors are assigned to each of these regions along with a 16 bit bitmap which define which of the two colors to display. When a '1' pixel is encountered, one of the pixel colors, color 1 is displayed, and when a '0' pixel is encountered, color 0 is selected and displayed. The following diagram depicts the mapping and associated pixel storage.

```
      1    2    3
     0123|0123|0123| ... region A1 is stored A1Color0|A1Color1|0123456789abcdef
     4567|4567|4567| ... region A2 is stored A2Color0|A2Color1|0123456789abcdef
A    89ab|89ab|89ab| ... region A3 is stored A3Color0|A3Color1|0123456789abcdef
     cdef|cdef|cdef| ...

     0123|0123|0123| ...                    .
     4567|4567|4567| ...                    .
B    89ab|89ab|89ab| ...                    .
     cdef|cdef|cdef| ...

     0123|0123|0123| ...
     4567|4567|4567| ...
C    89ab|89ab|89ab| ...
     cdef|cdef|cdef| ...

       .   .   .   .
       .^ Regions of ^.
       . the display  .
       .   .   .   .
```

Decompression is done in real time by the LINDA chip. LINDA sends

standard Half Chunky pixels or standard Hybrid pixels to MONICA after decompression. LINDA writes then reads packed pixel line buffer data using a sequential addressing scheme.

The display data fetch circuit places constraints as to the exact position on the screen where packed pixel lines can begin. The rule is simple: packed pixel lines (processed in units of four display lines in single line repeat mode, units of eight in two line repeat mode, etc) can start on the first active display line as defined by the DIWSTRT register, or every fourth (eighth, etc) physical display line there-after. Attempting to start a packed pixel display on any other scan line will result in unpredictable display output, and extranous graphics overrun interrupts. This of course implies that the vertical scroll of packed pixel displays is course; these display modes can only be vertically scrolled in groups of four (eight, etc) scan lines.

7.4.7.1 PACKLUT Pixels. -

PACKLUT pixels are decompressed by the LINDA chip to form standard Half Chunky pixels. Each of the screen region colors are 8 bit values used to address an entry in MONICA's Color Lookup table. 32 bits are required to fully specify each screen region giving a data compression ratio of 4 relative to standard Half Chunky pixels. LINDA performs the data decompression in the OUTMUX circuitry. (All PACKLUT data is stored in a line buffer, even in the high end system, and even/odd pixel extraction (in the case of the high end system) is done as the data is decompressed.) Because of the organization of packed pixels (each region contains information for 4 sequential scan lines), each buffer of data is accessed four times. This means ANDREA only needs to send data to LINDA one out of every four scan lines. (ANDREA can also take advantage of the fact that there is more time available and send packed pixel data to LINDA over the course of more than one scan line. This permits the generation of displays which would otherwise be impossible to produce due to the fact that there would not be enough RAM bandwidth available for all data to be accessed in one scan line time.) Each time a buffer is accessed by LINDA's output control circuitry, a different nybble of the bitmap portion of the packed pixel is used to determine proper decoding. In the case of PACKLUT pixels, four Half Chunky pixels are decoded simultanously with each transfer to MONICA.

Alignment of PACKLUT pixels in memory is restricted to double long-word alignment (64bits).

7.4.7.2 PACKHY Pixels. -

PACKHY pixels are decompressed to form standard Hybrid pixels. The two colors defining each screen region are 24 bit values used to drive the display color guns directly. In this display mode, 64 bits of data are required to specify each region giving a data compression ratio of 6 relative to standard Hybrid pixels. The chips process PACKHY data in much the same manner that PACKLUT data is processed. The differences are that twice as much data must be fetched and stored in LINDA, and pixel data is sent to MONICA in the same manner HYBRID pixel data is sent.

Alignment of PACKHY pixels is restricted in memory to double long-word alignment (64 bits).

8 MONICA.

MONItor Control Array.

8.1 Sync On Green.

Many monitors manufactured for the IBM and IBM compatible machines are set up to extract a composite sync signal from the analog Green input. As one of the design objectives of the AAA chip set is to support as many monitors as possible, MONICA has been designed to drive composite sync onto the analog green output when programmed to do so. When set high, SOGEN of BPLCON2 enables this function.

8.2 Color Table Offset Registers.

New registers, BPLOFF and BPLOFFR, have been defined to permit fast color context switches when the chips are in bitplane or Half Chunky mode. This register can also be used to generate interesting color cycling effects. As the name implies, the register holds base addresses for the two playfields (as well as the even and odd Sprites). The registers can be written via either BPLCON4, or BPLOFFN. When the offset registers are accessed using BPLOFFN, the value stored in the register is added to all bitplane or Half Chunky addresses applied to the LUT. The adders are modulo 256 and therefore will roll over if the sum is greater than 255. When the offset registers are accessed using BPLCON4, the value stored in the register is XORed with the addresses applied to the LUT. Data written to the offset registers using BPLCON4 writes to the upper four bits of the registers while clearing the lower four bits.

In addition to this register a bit has been added to BPLCON3: BASEN, Color Base Enable. When BASEN is set, the value present in EBPLOFF and OBPLOFF of BPLOFF are applied to either the adder or the XOR depending on whether BPLOFFN or BPLCON4 was accessed last. When this bit is cleared, EBLPOFF and OBPLOFF are disabled by driving zeros to the input of the adders and 0x10 is applied to the adders during Sprite LUT accesses to disable ESPRM and OSPRM.

8.3 Overlay Plane.

The AD bus path to BPLDAT registers has been remain intact in MONICA. This datapath allows the even playfield to be used as an overlay playfield when a CHUNKY or PACKED display mode has been enabled in the odd playfield, and when graphics fetches are made from VRAM. Burst mode transfers are used to write overlay playfields into MONICA. ANDREA uses the BPL2PT and the BPL2MOD register to address overlay plane graphics data. MONICA buffers all 128 bits of this data in BPLOVRDAT. The OVRLAYEN bit in DMACONX is used to enable overlay plane display (MONICA) and overlay plane DMA (ANDREA). The same timings shown in Appendix A for RAM Burst Access are used when the overlay playfield is written into MONICA.

Overlay graphics are limited when compared to normal graphics modes. No hardware horizontal scroll is available for overlay playfields. (Note: overlay plane pixels will be processed at the beginning of each scan line at the same point in time which the

adjacent chunky pixel data is started. The even playfield scroll value is still accessed and used to control the exact starting point relative to the adjacent chunky pixel playfield. The overlay playfield is not scrollable because the even playfield scroll value does not contain enough precision to control a scroll across the entire 128 bits of pixel information which is burst fetched into the chip. The overlay plane can be "scrolled" up to 15 bit positions.) No address pointer modulus can be applied to overlay planes at the end of display lines. Each display line must contain an integral number of four longword pixel data plus 1. (If the physical display line is not an integral multiple of 128 pixels (bits), the extra data which is fetched will be ignored by the display hardware.) Plus 1 is added to the number of 128 bit pixel groups fetched due to an additional hardware limitation. Each display line must contain an extra 128 pixels (bits) of 'padding' at the end of each scan line. This padding is never displayed, but is always fetched by the graphics system. For example if a 640 pixel overlay plane is desired, the image in memory must contain 768 bits per scan line.

8.4 HAM Mode.

The new chip set supports HAM mode exactly as in the old chip set. When HAM of BPLCON0 is set, then bitplanes 1 thru BPU are accessed to generate the display. BPU is set to 6 to enable a HAM mode which is compatible with old Amiga chips. In this case planes 5 and 6 supply the HAM control bits and planes 1 thru 4 provide an address to the LUT, or are used to replace one of the LUT color outputs as driven by the HAM control bits. When the LUT is addressed, only the bottom 16 LUT registers are available. When one of the LUT colors is replaced by planes 1 thru 4, both most significant and least significant color nybbles are replaced with the specified HAM replacement color.

HAM mode has been extended to take advantage of the additional bitplanes the AAA chip set provides. When HAM mode is enabled, but BPU has not been set to 6, up to ten bitplanes can be used. In this mode, bitplanes 3 thru BPU are used to address the LUT, and bitplanes 1 and 2 are used to provide the HAM control information. When the BPU is set to 10, all 256 LUT registers are accessible. When a LUT color modification is done, all 8 bits of the chosen LUT color are replaced with plane 3 thru 10 data. If BPU is set to some number greater than 6 but less than 10, then the most significant address bits are forced low. This permits HAM modes requiring lower data overhead, and provides compatibility with the AA 8 bit HAM mode.

Following are the HAM commands used to control HAM modes. Note that as each pixel is processed, the HAM commands appear on the HAM control bus(note they are identical to the HAM commands which exist internal to the old DENISE chip):

00 - use the bitplane address specified by the other bitplanes to access the LUT and replace all 3 display colors resulting data
01 - replace the blue display color with the bitplane LUT address
10 - replace the red display color with the bitplane LUT address
11 - replace the green display color with the bitplane LUT address

8.5 High End System HAM.

In the low end system, MONICA processes HAM mode in the same manner as the old DENISE chip. When a modify pixel is encountered, MONICA inserts the pixel into the appropriate red, green, or blue color field just before output. In the high end system, a difficulty arises in the HAM logic which requires further discussion.

In the high end system, each MONICA is presented every other pixel for processing. The nature of HAM demands that both of the MONICA chips know what the last displayed pixel was in order to have the ability to modify it. To provide this ability, each MONICA outputs the raw bitplane data used in HAM mode for input to the remote MONICA (HAMC0-1, and BLUTA0-7 are output from each MONICA and input to the other) so that each MONICA can determine what the last bitplane pixel colors were. HAM is further complicated by the fact that SPRITE display may inhibit any given bitplane LUT fetch, however the bitplane HAM logic must continue to access and modify bitplane colors while SPRITE are being displayed. To accomodate HAM mode in high end systems, the Color Lookup table has been triple-ported. One port is dedicated to the processor for relatively low speed on-the-fly updates. This port is a bi-directional read/write port. The second port is a high speed read-only port. This port is used by all the display modes (including HAM mode) which access the LUT for an indirect source of pixel color data. The third port is another high speed read-only port used to make all SPRITE accesses. Off-loading SPRITE LUT accesses onto the third port has allowed the AA HAM processing algorithm to be extended so as to provide full HAM functionality in the high end system.

8.6 Extra Half Brite Mode.

Extra Half Brite mode has not been extended in the new chip set. It has been retained as originally defined to support old software. As in the old chip set, Extra Half Brite cannot be explicitly enabled. Whenever BPU is set to 6, when dual playfield is disabled, and when HAM is not set, Extra Half Brite Mode is enabled. This setting can of course be overriden by setting KILLEHB, kill extra half brite.

8.7 Horizontal Scrolling.

This section describes how each of the graphics modes are scrolled.

8.7.1 Bitplane Mode. -

Bitplane data is scrolled in the same manner as in the old chips. Program a value of 1 to 16 into the playfield scroll registers to scroll low-res pixels, and program a value of 1 to 8 to scroll high-res pixels on low-res pixel boundaries (high-res pixels can only be scrolled in pairs).

8.7.2 HALF CHUNKY And PACKLUT Pixels. -

Scrolling for Half Chunky and Packed LUT pixels is controlled by programming a value of 0 to 3 into the scroll control register when in low-res mode. The scroll control register is programmed with 0 to 1 when the display mode is high-res. Alignment of Half Chunky pixels in memory is restricted to double long-word alignment (64bits). Because of this it is necessary to reprogram DDFSTRT and DDFSTOP with respect to DIWSTRT in order to smoothly scroll 0 to 7 pixels. (The glitch occurs across the longword (pixels 3 to 4) boundary.)

8.7.3 Four-Bit HALF CHUNKY Pixels. -

Scrolling for 4-Bit Half Chunky pixels is controlled by programming a scroll value of 0 to 7 in low-res mode, or by programming 0 to 3 in high-res mode. Alignment of 4-Bit Half Chunky pixels in memory is restricted to double long-word alignment (64bits). Because of this it will be necessary to reprogram DDFSTRT and DDFSTOP with respect to DIWSTRT in order to smoothly scroll 0 to 15 pixels. (The glitch occurs across the longword (pixels 7 to 8) boundary.)

8.7.4 Two-Bit HALF CHUNKY Pixels. -

Scrolling for 2-Bit Half Chunky pixels is controlled by programming 0 to 15 into the scroll control register in low-res mode, or by programming 0 to 7 in high-res mode. Alignment of 2-Bit Half Chunky pixels in memory is restricted to long-word alignment (64bits). Because of this it will be necessary to reprogram DDFSTRT and DDFSTOP with respect to DIWSTRT in order to smoothly scroll 0 to 31 pixels. (The glitch occurs across the longword (pixels 15 to 16) boundary.)

8.7.5 CHUNKY Pixels. -

Scrolling for Chunky pixels is controlled by programming a value of 0 to 1 into the scroll control register in low-res mode. Alignment of Chunky pixels in memory is restricted to double long-word alignment (64bits). Because of this it will be necessary to reprogram DDFSTRT and DDFSTOP with respect to DIWSTRT in order to smoothly scroll 0 to 3 pixels. (The glitch occurs across the longword (pixels 1 to 2) boundary.)

High-res chunky pixels cannot be scrolled using the PF1H scroll value in BPLCON1. (The corresponding high-res pixel scroll across low-res pixel steps is accomplished by reprogramming the DDFSTRT and DDFSTOP registers.)

8.7.6 HYBRID And PACKHY Pixels. -

Hybrid and Packed Hybrid pixels cannot be scrolled by programming the scroll control register. In these pixel modes, scrolling is accomplished by reprogramming DDFSTRT and DDFSTOP with respect to DIWSTRT.

8.8 Sprites.

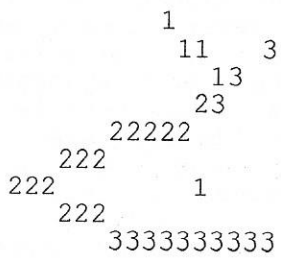
Extended Sprite registers have been added to the chip set to permit the display of 32 bit sprites. Several new modes have also been incorporated into extended sprite operation. The following paragraphs discuss the new sprite modes. Note that whenever any of the old SPRxCTL registers are written, the bits which enable the new modes defined below are reset so as to provide software compatibility with the old chip set. These modes can only be enabled (or reenabled) by writing (or rewriting) the extended sprite registers.

In the old chip set, Sprite resolution was always lowres. In other words, in lowres mode, each sprite pixel was 1 bitplane pixel wide. In highres mode, each sprite pixel was 2 bitplane pixels wide. A FAST bit has been added to the extended sprite control registers which allows this to be overriden. When this bit is a zero, the sprite is displayed in lowres (half the pixel clock speed) as in the old chipset. When this bit is a one, the sprite is displayed in highres, or at the pixel clock speed (this works in all systems and combinations of lowres screens/highres screens except when in a dual system and lowres pixel mode is set).

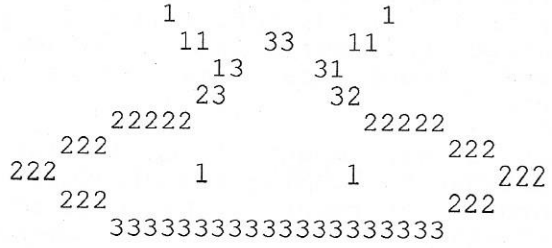
In the old chip set, it was impossible to display sprites outside of the display window defined by DIWSTRT and DIWSTOP. No sprites could appear in the screen border. An extended sprite control bit has been defined, CONSPRT, which will override this shortcoming. When this bit is set, sprites are enabled for display anywhere except blanking intervals. MONICA uses the signal present on the CBLK input pin to determine where the blanking region is. It is possible to have SPRITE collisions generated within the blanking regions. From a hardware point of view, MONICA processes SPRITE data at all times when CONSPRT is set. This information is simply blanked at the DAC outputs when the CBLK input indicates a blanking region. When CONSPRT is cleared, sprites can only be displayed within the region defined by DIWSTRT and DIWSTOP.

A new sprite repeat function has been incorporated into the new chip set. The control field HSRPT in SPRxCTLX defines how many times a sprite is to be repeated each horizontal line. This mode allows the generation of wide repetitive sprites. For example, it could be used to draw horizontal lines for cursor generation. A final use of this mode is generation of double wide symetrical sprites. The following paragraph describes this futher.

The new SMIRROR bit allows sprites to be mirrored about a vertical axis each time the sprite is redisplayed under control of HSRPT. The axis is seen to be located just beyond sprite data bit 0. SMIRROR allows the display of wide symetrical sprites. An example of its use is as follows. It is desired to generate a 64 bit wide space ship. One method of doing this is to use two sprite channels and display them so they always touch each other. But this means using two sprite channels and it means that two sets of registers must be manipulated. Another way to accomplish the task is to design a space ship which is left to right side symetrical. The left side of the ship can be coded as a sprite bit map as follows:



Then the HSRPT field is set to 1 (to repeat the sprite twice) and the SMIRROR bit is set. What will appear on the screen is the following double wide sprite:



Pretty crude, but you get the point. This mode is realized in hardware by using a circulating sprite shift register and reversing shift direction each time the sprite is repeated.

8.9 Burst Mode Sprites.

Two additional control bits have been provided in the SPRxPOSX registers, BURST64, and BURST128. When one of these control bits is set, the DMA controller executes burst mode bus cycles to satisfy SPRITE data requests. This permits the display of 64 or 128 bit wide SPRITEs. Certain limitations exist regarding the use of burst mode SPRITE fetches.

The Burst64 control bit in the extended Sprite control register enables 64 bit wide Sprites, and causes two transfer burst cycles. 64 bit Sprite lists must be aligned to begin at even two longword addresses. The Sprite DMA hardware has been designed to execute two complete two transfer burst cycles when Burst64 is set. One burst transfer is used to fill the SPRxDATX A data register, and the other is used to fill the SPRxDATX B data register. The DMA hardware was simplified by requiring two such burst transfers always be made whenever the sprite list is accessed, and the Sprite DMA channel is in Burst64 mode. To this end, the Sprite position and Sprite control information which is found following the Sprite data portion of the list must be repeated two times (ie position, position, control, control). This not only simplifies the DMA logic, but also provides a means for padding Sprite lists with data to prevent page crossing problems stemming from Sprite data alignment problems.

The Burst128 control bit in the extended Sprite control register enables 128 bit Sprites, and causes four transfer burst cycles. 128 bit Sprite lists must be aligned to begin at even four longword addresses. In a manner similar to the method described above for

padding Sprite lists with data to prevent page crossing problems, the Sprite position and Sprite control information which follows the data portion of the Sprite list must be repeated four times (ie position, position, position, position, control, control, control, control).

It is possible to generate Sprite lists containing a mix of 32 bit Sprites, 64 bit Sprites, and 128 bit Sprites, however certain alignment restrictions apply when such mixing is done. The length of the position and control sections of a Sprite list are not the only data which can affect Sprite list alignment. Sprite data itself can affect the alignment of the list. For example, if a Sprite list were set up to display Sprite 0 twice, with the first Sprite definition in this list set up to be a 32 bit Sprite 2 lines high, and the second Sprite definition set up to be a 128 bit Sprite, then the Sprite list must be situated in memory to begin at an address such that address bit A1 is 0, and address bit A2 is 1. This will guarantee that no page crossing problems will be encountered during the processing of the 128 bit Sprite because when it comes time to fetch and display the 128 bit SPRITE, the SPRITE address pointer will have been advanced by 12, and both the A2 and A1 address bits will be set to 0. The user should also be aware that Sprite control information is retained from display field to display field. Therefore the Sprite padding restriction established by the last Sprite in a list applies to the position and control data for the first Sprite in the list. This of course can be over-ridden by programming the copper or processor to modify the Sprite control registers at vertical blanking time before the lists are first accessed for the first time in the new display field.

8.10 Establishing MONICA's Outputs.

MONICA's outputs can be configured in several different ways depending on the system and system needs. MONICA's digital and pixel data output buses can be selectively enabled or disabled. A table at the end of this section defines all of the options available for each of the system configurations which have been anticipated.

8.10.1 Selecting The Analog Or Digital Color Outputs. -

In dual system configurations, the PIXMUX pin is used to set the chip to enable the internal color DAC's. The PIXMUX pin is tied low to establish this mode of operation. During reset, MONICA tristates then samples this line. If it is low for more that four clock cycles, MONICA sets herself to local DAC mode, and multiplexed operations will take place if the chip is in a dual system configuration.

To set the chip to external DAC mode, PIXMUX is pulled high with a 10k ohm resistor. The chip will sense that the pin is not low, and after reset is over, both odd and even chips will be set to output their digital RGB data as quickly as possible. PIXMUX becomes an output at this time as well, and outputs an appropriately delayed clock, 'sftclk' to be used by external circuits to latch the digital pixel data. The output skew of this clock has been matched to the digital pixel data skew so that the first cycle of a lores two cycle sequence is present when PIXMUX is high. An external pixel multiplexer and D->A can then be implemented with circuitry similar to what is inside the chip for the necessary postprocessing operations. In addition to enabling chip digital color data, PIXMUX high at reset time causes SOGEN/ZDCLKEN to be output on the chip's ZD pins. The ANDREA

chip provides the CSYNC and CBLK to external circuits.

Single Monica systems also sense the PIXMUX line. As in the dual system case, PIXMUX must be tied low to enable the local D->A. Normally the digital RGB outputs are zeroed in this mode to avoid FCC, power, and power supply glitch problemss. If desired or necessary, the DIGON bit(0) in the MONTEST register can be used to enable all of the digital color bits in single system configurations. (Note that DR(7), DG(7), DB(7), and DB(0) are always enabled to drive.) The DIGON bit has no effect in dual system configurations.

In single Monica systems set to external (as described above for dual systems), digital color data is output as soon as possible. SOGEN is presented to the ZDX pin for use by external D->A circuitry. GAUD is still sent on ZD as expected during blank, but it's start is delayed four cycles to avoid pipelining problems in the external ZD circuit. Note that SOGEN is presented to the ZDX pin in the internal single mode as well as external single mode.

8.10.2 Genlock Device Transparency Control Bits, ZD And ZDX. -

In internal dual mode, ZDX is used to transfer ZD data from the even chip to the odd chip. The odd chip ZD pin in turn presents the final (multiplexed) ZD data, changing on half PCLK (sftclk) boundaries. The nominally unused ZD pin on the even chip presents GAUD in this configuration. GAUD is also presented on the ZD pin during blanking under normal operating conditions. It is presented to the ZDX pin in dual external systems during blanking as well, but with a four cycle delay to avoid pipeline delay problems in the external multiplex circuits (note this is the same technique used in the single mode). The ZDCLKEN pin has no effect on the ZDX pin. It causes an inverted version of PCLK to be output to the ZD pin, which overrides most other controls. The delay of this output matches the delay of the RGB signals, and thus the rising edge of this clock can be used to reclock the digital RGB data. The clock presented in DUAL systems may be ineffective as the pixel data is a half PCLK cycle long. In this case, use a Phase Lock Loop circuit to generate a 2x PCLK signal. In the dual external system, SOGEN and ZDCLKEN on the ZD pin are not overridden by the ZDCLKEN pin. This allows the external multiplexer to use the same PLL technique to make a ZDCLK signal if desired.

MONICA Output Control Table:

dual	local	even	digital outputs zeroed	dtoa zeroed	RGB delay (cycles)	ZD pin	ZDX pin	PIXMUX****
0	0	0	no	1 (off)	out(1)	ZD**	SOGEN	out(10k high)
0	0	1	no	1 (off)	out(1)	ZD**	SOGEN	out(10k high)
0	1	0	(~DIGON or BLANK)*	BLANK	out(2)	ZD**	SOGEN	in (tie low)
0	1	1	(~DIGON or BLANK)*	BLANK	out(2)	ZD**	SOGEN	in (tie low)
1	0	0	no	1 (off)	out(1)	SOGEN	oddZD+	out(10k high)
1	0	1	no	1 (off)	out(1)	ZDCLKEN	evenZD+	out(10k high)
1	1	0	no***	BLANK	in	ZD**	in	in (tie low)
1	1	1	no	1 (off)	out(1)	GAUD	evenZD	in (tie low)

Notes:
BLANK= inverted CBLK one cycle delayed
- RGB(delay) -- delay is cycle from plocalout
+ GAUD is switched onto this pin four cycles after blank starts, and is switched off just as blanking ends, as in single ext mode for ZD pin below.
* normally blanked on BLANK if on for test, otherwise off completely.
** "normal" ZD signal, with GAUD on blanking. Normally, GAUD is switched onto the pin immediately when blanking starts, and goes off immediately when it ends. In external single systems however, GAUD is switched onto the pin four cycles after blanking starts and goes off immediately when blanking ends. This avoids pipelining problems for a circuit sampling GAUD on ZD after an external circuit has delayed the signal a few cycles. If ZDCLKEN is true, this overrides all other settings for this pin in the ** circumstances. In this case an inverted version of PCLK comes out whose skew delay matches the digital RGB skew delay. Use the positive edge of this clock to capture RGB digital data.
*** don't care, as bits are used as inputs here
**** signal output is sftclk delayed.

8.11 Zero Detect (Genlock Transparency) Operation;

The original Amiga ZD functions as well as several new ZD modes are provided for use when the chip ZD pin is not being used for other system functions. (See previous section.) For compatibility, when none of the new ZD modes are enabled, the ZD pin is asserted if all the addresses to the colortable are zero (both Sprite and Playfield). In Hybrid mode, ZD is activated if all 24 bits are zero. In Chunky mode, bit 15 defines ZD unless a sprite has priority over the CHUNKY data. In this case ZD will NOT be active. Only a zero sprite will activate ZD. New ZD modes are available for use in bitplane mode. Among the new modes: one of the enabled bitplanes can be selected to activate the ZD pin on a plane by plane basis; and a new bit position has been added to each LUT register to allow individual color to be singled out for ZD. The following describes register bits available for controlling each of the available ZD modes.

BPLCON2 (0x104)
bit 15-12 ZDBPSEL(3:0)
bitplane mode:
0-15 select bitplane 1-16 to generate ZD
other modes:
0-7 selects bit 0-7 of pixel
8 selects overlay plane

bit 11 ZDBPEN
Enables the selected bitplane to assert ZD. ZD will be asserted independent of any active Sprite priority. This bit does not disable other ZD modes from also asserting ZD. ZD asserted due to this mode is or'ed with other ZD assertions except as noted below.

bit 10 ZDCTEN
enables the color table MSB (transparent bit) to assert ZD high rather than the default generation of ZD (the condition when ZDCTEN is low). (Any ZD assertions resulting from this mode are or'ed with ZDBP.)

BPLCON3 (0x106)

bit 5 BRDBLK
sets the border (outside display window) to be black. Doesn't affect ZD. can be disabled if BPLCON0(0x100), bit 0 ENBPLCN3 is low.

bit 4 BRDNTRANS
border not transparent. Sets ZD to zero outside of display window. Can be disabled if BPLCON0(0x100), bit 0 ENBPLCN3 is low. Overrides all other settings above if enabled by enbplcn3.

9 MARY.

More Audio Registers Y.

This chapter describes new disk and audio features which have been added to the custom chip set. It also covers some differences of operation between MARY and the old PAULA chip.

9.1 Disk Controller.

The AAA disk controller has been designed for flexibility. It can be used to read/write a variety of serial data streams from a variety of media and or hardware. Tempered with the design goal of flexibility was the desire to achieve a low cost controller. In order to accomodate both of these goals simulatanously, it was necessary to sacrifice some of the hardware automation normally associated with disk controllers, and require that device software assist with some of the sector header formatting operations. By choosing the proper balance between software support and hardware, a controller has been implemented which provides competitive access times, minimum software overhead, minimum chip real-estate, and maximum flexibility.

Two and four megabyte floppy disk drive support has been added to the custom chip set. The floppy disk circuits are capable of encoding and decoding MFM format to provide IBM compatibility, and new modes have been added to support RLL(2,7) formats and audio CD ROM. New and extended RGA addresses have been defined to control these new modes and higher density floppy drives. A new DMA channel has been defined which allows a high degree of flexibility with respect to disk media formats. Under software control, it is possible to support many new sector formats without the need for hardware modifications. Old disk control registers have been left intact to preserve compatibility.

New floppy features include:

1) data separator up to 20x faster (100ns window) with variable parameters.
2) data separator can have different (async) clock from system clock.
3) decode/encode of MFM, Biphase Mark, and RLL(2,7) codes.
4) sector based read and write with CRC calc and generation. data dumped/read directly from/to user's buffer.
5) continuous read/write of CD/DAT/FM broadcast digital format.
6) specialized formats with data rates up to ~5Mbit/sec.

9.1.1 Comparison Of Disk Controller Capabilities: Mary Vs Paula. -

	Mary	Paula
raw bit width	88-9000ns*	2000,4000ns
max raw bit rate	11.4M bit/sec*	0.5 Mbit/sec
encoding methods:		
raw	yes	yes
GCR	yes	yes
MFM	yes	no (done in software)
RLL(2,7)	yes	no
Biphase-Mark	yes	no
sync	32,16,8 bit	16 bit

modes:		
track	yes	yes
sector	yes	no
CD digital	yes	no
trackplus	yes	no
hardware CRC	yes (sector,trackplus)	no
async PLL clock	yes	no
input types	pulse,NRZ	pulse

*- May need increasingly good data to work below 140ns, 7Mbits/sec. The maximum DMA rate is <= 9.9Mbits/sec, so only encoded modes will work above this rate (CD and trackplus modes- need 1/2 raw dma rate).

9.1.2 Theory Of Operation. -

At the heart of the disk controller is a simple state machine. On read, the state machine takes a 1 bit wide stream of data and converts it into a 32 bit wide dma stream. On write, the state machine performs the reverse operation. The disk controller also contains hardware used to deal with various formats the data stream may take on.

The serial data stream undergoes several layers of formatting prior to actual transmission to a storage media or receiving device. At the first layer of formatting, clock and sync information is added to the data stream. Clock information is necessary to specify exactly where in the data stream each data bit is latched. Sync information signifies where word boundarys in the data stream begin. Sync information is normally inserted into the data stream at block boundaries, while clock information is interspersed regularly thru-out the data bits. Both sync and clock information is required in the data stream to insure reliable recovery of information. Special coding techniques must be used to combine the clock and sync information with the data bits so that when the data is retrieved, sync information can be readily recognized in the data stream, and so that clock information can be easily separated from the actual data. Several standard coding techniques have been established in the industry which satisfy these requirements. A distinguishing feature of these codes is that there are a minimum and maximum number of zero bits between any two successive '1' bits. (A '1' bit is signified by either a pulse (disk drives) or an edge (CD format).) In MFM and Biphase-Mark formats one data bit is coded to form two raw bits (one clock and one data bit). In one type of GCR code, 4 data bits are coded to form 5 raw bits. Another type of GCR code converts 6 data bits into 8 raw bits. Other more sophisticated Run Length Limited (ex, RLL(2,7)) codes encode groups of data bits in such a way as to guarantee a certain minumum spacing between two successive one bits such that the data stream transmit rate can be increased beyond that which can be done using (for example) the MFM coding scheme. This permits more data to be stored on a media of a given density specification. GCR, FM, MFM, RLL(2,7), and Biphase-Mark are all RLL codes. In many descriptions of these and other RLL codes, there is reference to clock bits and data bits. In the earlier codes, (FM, Biphase-Mark) every other bit is a clock bit, while the others are data bits. In MFM however, some clock bits are missing from the A1* sync mark so that these can be recognized within the bit stream without ambiguity. In RLL(2,7) and GCR the data bits are encoded such that there is no clear distinction as to what is clock and what is data.

9.1.3 Disk Controller Coding Formats. -

The AAA disk controller contains all the hardware necessary to support each of the data coding schemes discussed in the preceeding paragraphs. This section presents a detailed description of each of these schemes. The reader is referred to appendix F for a description of each of the disk control registers. Bits 0 an 1 of ADKCONX ('CODE') are programmed to set the desired code.

- CODE=0 (raw). Raw bits go directly to/from memory. This is the same as the Paula disk controller. All encoding/decoding is expected to be done in software. GCR also uses this code.

- CODE=1 (MFM).

MFM Encoding Chart

data	raw	
1	01	
(0)0	00	
(1)0	10	if a 1 precedes the zero,the 1st bit is set

sync mark (raw): 0x4489

This code requires a medium amount of transmit bandwidth, and a medium amount of data separation accuracy.

- CODE=2 (RLL(2,7)).

RLL(2,7) Encoding Chart

data	Ex. hex data	raw	hex	Disk Controller Decode States (read)	Disk Controller Encode states (write)
10..	A..	1000	8..	0->1->0	0->1->0
11..	F..	0100	4..	0->2->0	0->1->0
000..	000..	100100	924..	0->3->1->0	0->2->1->0
010..	492..	001000	208..	0->1->1->0	0->0->1->0
011..	6DB..	000100	104..	0->0->2->0	0->0->1->0
0010..	2..	00001000	08..	0->1->0->1->0	0->2->0->1->0
0011..	3..	00100100	24..	0->1->0->2->0	0->2->0->1->0

sync mark (raw): 0x4048 (2 in a row good idea- 0x40484048)

This code requires about 50% less bandwidth for the same data as MFM, but needs better data separation and disk noise margin than MFM.

The algorithm used to encode RLL(2,7) is as follows: look at the first two bits in the data stream. If they match the 10,11 patterns, output the four raw bits as specified in the preceeding chart. If the first two bits are not '10' ot '11', consider the first three bits. Check to see if these bits match any of the three bit patterns given in the chart. If there is a match, output the corresponding six bits of raw data. Finally, if there has not yet been a match, determine which four bit pattern given in the chart matches the first four bits in the data stream, and then output the corresponding eight raw bits. Once the first pattern is encoded, move the pointer past the bits just encoded, and start again.

Section 9.1.11, 'C Routine Which Encodes/Decodes RLL(2,7)', lists a C program which implements this algorithm.

o CODE=3 (Biphase Mark).

Biphase Mark Encoding Chart

data	raw
1	11
0	01

sync marks(raw): 0x9C (B-block start)
(replaces left chan every 192)
0x93 (M-left channel)
0x96 (W-right channel)

This code is primarily used to drive CD-ROM devices. It requires a higher (ie, 2 times) transmit bandwidth than MFM for the same data, but has smaller demands on the data separator than MFM.

Reference the following articles for more information:
"Philips-Sony Digital Audio Interface", Elektor Electronics, June 1988

"AES Recommended Practice for Digital Audio Engineering- Serial Transmission Format for Linearly Represented Digital Audio Data", AES3-1985, ANSI S4.40-1985

"The Art of Digital Audio", John Watkinson, Focal Press, Stoneham, MA. ISBM 0-240-51270-7

9.1.4 Interfacing The Disk Controller To Software Device Drivers. -

A second layer of formatting is usually applied to data being written to storage media. This layer of formatting is done to provide relatively quick random access to small blocks of data at some physical location on the media. Data formats at this level define media specific addresses. Some formats also define a means to enable bit error detection, while others define a means to enable both error detection and correction. Many high level data formats have been defined in the industry. All of these formats cater directly to the media being used and to the hardware being used to access the media. To a smaller degree, these formats are also tailored to the needs of the systems in which they are used.

It would be a significant benefit to Commodore if our systems could be made to interface to as wide a spectrum of serial bit stream storage devices as possible. However, providing direct hardware support for all desired media in this catagory would render our systems too costly to fit the budget of the majority of our customer base. Yet many serial bit stream data media have many common attributes. When examined at all levels, one discovers that the most significant differences between each of these media lay in the high level formatting which takes place. These facts forces one consider the possibility of requiring a certain degree of software formatting support to significantly reduce the total system hardware overhead necessary to drive any particular storage media, yet retain a competitive performance position in the personal computer market.

Hardware cost alone is not the only reason software formatting support should be considered. New media and new media formats are introduced into the industry regularly. By providing the minimum hardware necessary to perform serial bit stream data operations at a competitive performance level and requiring system software device drivers to assist the hardware with high level formatting tasks, there is a high probability that our systems can be made to support future media and formats without requiring hardware changes. These trade-offs have been made in the AAA disk controller circuits. In an area equivilent to that which would be required to implement a Western Digital floppy disk controller circuit, a controller has been implemented which, with a small amount of system software support, can not only emulate all the functions of a standard Western Digital floppy disk controller used in IBM PC systems, but can also emulate the floppy disk controller being used in Commodore's A500 Amiga product. Other media which can be driven by the AAA disk controller will be covered in subsequent sections of this document.

9.1.5 Possible Applications For The Disk Controller. -

This section describes some of the media and formats evaluated over the course of the disk controller design. These are listed in table format. The items mentioned in the table give an indication of the broad range of media which might be supported with this disk controller circuit, right out to the edges of the envelope.

	Mary	Paula
0.5 Mbyte floppy		
<0.25 Mbit/sec raw,		
<0.125 Mbits/sec data		
IBM 360K format	yes (sector or track)	yes (full track write)
Apple II ?	yes	yes
1.0 M floppy		
<0.5 Mbit/sec raw,		
<0.25 Mbits/sec data		
Amiga format	yes	yes
IBM 720K format	yes (sector or track)	yes (full track write)
MAC format	most likely	1st half of disk or so
CBM GCR	yes	no
2.0 M floppy		
<1.0 Mbit/sec raw,		
<0.5 Mbits/sec data		
IBM 1.44M	yes	no
IBM 1.2 M	yes	no
MAC ?	most likely	no
4.0 M floppy		
<2 Mbit/sec raw,		
<1 Mbits/sec data		
IBM 2.88M	yes	no
new Amiga RLL(2,7)		
track format		
(theoretical)		
4.0 M	likely	no
5.2 M	maybe	no
vary clk rate		
track to track		
6.2 M	maybe	no

30 M floppy (RLL(2,7)) <6 Mbits/sec raw <3 Mbits/sec encoded standard format (if developed) 21.6 M	likely (async clock for write)	no
Amiga track format (theoretical) 20 M	most likely	no
26 M	maybe	no
CD* digital format <5.6 Mbit/sec raw, <2.8 Mbits/sec data	yes (write may need async clock)	no
DAT* digital format <6.1 Mbit/sec raw, <3.1 Mbits/sec data	probably (write needs async clock)	
digital broadcast* <4 Mbit/sec raw, <2 Mbits/sec data	probably (write needs async clock)	no
ST-506 hard drive <10 Mbit/sec raw < 5 Mbits/sec data @ 9.52/4.76 Mbits/s	maybe	no
@ 10/5 Mbits/s	maybe (write needs async clock)	no
Ethernet <10 Mbit/sec data	slight possibility using PLLRST and much external support	no

* CD,DAT, and digital broadcast are the same format, just different speeds.

9.1.6 Disk Controller Modes. -

This section describes extensions made to the Paula floppy disk controller circuits to enable high performance support of media previously alien to the Amiga product line. See the ADKCONX register description in appendix F for the register bits programmed to set the desired disk mode.

- Track Mode. Track mode (mode=0) is designed to be used with the old Amiga format. Track mode is also used to format disks for use in sector mode. As with the Paula floppy disk controller, track mode requires that data be converted to/from raw bit format by the coprocessor or host processor.

- Sector Mode. Sector mode (mode=1) has been designed especially for the Western Digital (IBM) format, but can also be used in a wide range of other sector formats. It can operate on from one to N sectors per track and perform CRC generation/error checking. Sector mode requires some raw data in memory. This data is a copy of the sector header which addresses the sector data to be

processed. The disk controller uses this data to match against the raw data stream from the disk drive. Once the proper sector header has been found, the disk controller will begin reading/writing unencoded sector data from/to memory, performing programmed coding on the data (MFM, RLL(2,7), etc) as it is processed from/to the media serial bit stream.

- CD Mode. CD mode (mode=2) is designed for CD, DAT, and digital broadcast formats. It transfers decoded data to memory, and encodes data from memory as it is output. CD mode generates an interrupt every sector to allow double buffering of audio data.

- Trackplus Mode. Trackplus mode (mode=3) is designed as a limited format mode for high speed transfers which require reduced dma bandwidth. It transfers only encoded/decoded data from/to memory. Sync marks are automatically inserted into/recognized from the raw data stream, and CRC's are generated/checked. Track mode requires an image of the unencoded track data exist in memory when writes are done, and leaves a decoded image of the track data in memory when writes are done. This image includes a header section, sync data, gap data, CRC data (this data is left blank when writes are done as the controller will calculate the proper CRC and insert it into the data stream), and the actual sector data.

9.1.7 Programming The Disk Controller; Getting Started. -

This section outlines examples of some of the media formats supported by the disk controller circuits. It also explains hardware bit settings needed to operate the disk controller in the specified modes. None of the examples presented have been tested in the real world. As of this writing, the disk circuits exist in schematic form only! The examples establish a starting point for software device driver development.

9.1.7.1 Sector Mode Example, IBM Format. -

9.1.7.1.1 IBM Format. -

This format has been taken from the Western Digital WD 1772-02 spec.

60 bytes 4E	beginning of disk only	
12 bytes 00	start of sector	96usec
3 bytes A1*	sync mark- raw bits 0x4489	24usec
1 byte FE	address mark, start addr. header	8usec
1 byte track #		8usec
1 byte side #		8usec
1 byte sector #		8usec
1 byte sector length		8usec
2 bytes address header crc		16usec
(set CRC to FFFF at beginning of first A1*)	end of address header	
22 bytes 4E	gap between address/data	176usec
12 bytes 00		96usec
3 bytes A1*		24usec

1 byte FB	data mark end of data header	8usec
256 bytes data	the real thing	2048usec
2 bytes CRC	CRC for data and data mark (CRC starts at first A1*) end of data section	16usec
24 bytes 4E	gap between sectors end of sector	192usec

repeat sector section for number of sectors on the track.
(times given are for 2.88 Mbyte floppy)

Note: the A1* sync mark violates standard MFM encoding in that certain clock bits are intentionally 'missing'. This allows the sync mark to be recognized without ambiguity.

The data section is somewhat variable. For example, a 1.44M disk, may consist of 36 256-byte sectors per track, two sides, eighty tracks per side, or it may consist of 18 512-byte sectors per track, two sides, eighty tracks per side.

9.1.7.1.2 Formatting. -

As disk formatting is only done occasionally and system performance degradation is therefore neglishable, no hardware support has been provided to assist with sector mode formatting. Device driver software is responsible for creating a 'blank' track of data for use in formatting the disk. This 'blank' data track must include all sector header information and appropriate CRC's. Device driver software must also encode this data into a raw bit stream. Use track mode to write the resulting raw blank track data to the disk.

9.1.7.1.3 Reading. -

Use sector mode with code=1 (MFM). While moving the head to the track desired, encode the address header into raw bits. Begin the encode at the third A1*. This should be followed by the encoded data header. Point DSKHDRPTX at the resulting raw bits.

Example (raw version of these):

1 byte A1*	(sync mark- raw bits 0x4489)	
1 byte FE	address mark	start of address header
1 byte	track #	
1 byte	side #	
1 byte	sector #	
1 byte	sector length	
2 bytes	crc for address header, starting at first A1*	end of address header
1 byte A1*		
1 byte FB	data mark	start of data header

Point DSKPTX to the address the sector data would like to go.

```
Set DSKCRCI reg to 0xE295  (the CRC for A1* A1* A1* FB-- data mark)
Set DSKGAP to sectcnt=1, gaplen= 22 + 12 + 3 + 1 = 38 encoded bytes
Set ADKCONX as desired     (fasta=0,fastb=1 for 2M floppy)
                           (set syncen1,0=2 for 32 bit sync)
                           (set mode=1, code=1)
Set DMACONX   0x80000210   disk dma on
Set DSKSYNCX  0x44894489
Set DSKLENN   len1= 8   (raw 16 bit chunks of address header to match)
              len2= 2   (raw 16 bit chunks of data header to match)
              len3= 128 (16 bit chunks of data stored)  256 byte sectors
              dmaen=1
              write=0
```

Write DSKLENN a second time. This starts the controller. Wait for the DSKBLK interrupt; have a global software timeout in place in case something goes wrong (for example, a hard bit error could occur which prevents the hardware from matching the sector header information). The interval timed should allow for two or three revolutions of the disk. After the disk operation complete interrupt is taken, read the DSKBYTRX register to check for a possible CRC error. The data will now be in the buffer starting at DSKPTX.

To read more than one sector from the same track, calculate the raw data describing each sector's address and data headers, and append these bits to the first sector's address and data header. Set the sectcnt in DSKGAP to the number of sectors desired instead of 1. All sectors will be read in the sequence specified, barring no hard sector header errors occur.

9.1.7.1.4 Writing. -

Use the same setup as defined above for the read operation except for the following variations:

* ADKCONX- set the precomp to an appropriate value and set MFMPREC
* point DSKPTX to the place the data should come from.
* DSKGAP- gaplen= 22 - 1= 21 (minus 1 byte is for controller delay)
* after the address header raw bits, add this instead of 4 raw bytes of data header:

```
 12 bytes of 00  -> 24 raw bytes  AAAA AAAA ... AAAA AAAA
  3 bytes of A1* -> 6  raw bytes  4489 4489 4489
  1 byte  of FB  -> 2 raw bytes-  5545
                    4 raw bytes-  0000 FF00 crc- hardware fills
                                  in 1st 16 bits and ends after
                                  bit 23 (matches Western Digital
                                  disk controller WD 1772-02)
```

* on DSKLENN, len2= (24+6+2)/2= 16
  also set the write bit.

Again, for more sectors append more raw bits of data onto the first sector and increase sectcnt.

9.1.7.1.5 Read To Write Time. -

The following figures specify hardware turnaround time for end of sync read data to start of write.

```
(gap*16) raw bits (mode 1 only) +
12 1/2 raw bits +
(6 C14 ticks + 4 CPLL ticks) rounded up to next integral # of bits +
12 CPLL ticks +
3  C14 ticks

ex: (4M disk @ standard speed, fasta=fastb=1)
  12 1/2 bits (bit times)
   1     bit  (rounded up ticks)
   .96   bits (CPLL and C14 ticks)

  14.46  bits
```

CRC calculations start at the raw->encode boundary in mode 1 on both read and write. They start after the sync mark in mode 3 on read and write.

9.1.7.2 RLL Sector Format Example. -

This example is very similar to the IBM sector mode example given above. The disk format is the same as that given in the IBM example. The only distinction is that the IBM example uses MFM coding, while this example uses RLL(2,7) coding. The following registers are programmed differently:

* Set the phase locked loop (data separator) to new values. See section 9.1.8.6, 'Exanple: Establishing PLL Parameters For RLL(2,7)'.
* DSKCRCI should be fine at 0xFFFF. It will also work at 0xE295, but this no longer has the same meaning as it did in IBM formats because sync mark is different.
* In ADKCONX, set CODE=2 instead of 1.
* If applying a pre-compensation on writes, set RLLPREC instead of MFMPREC.
* Use 0x4048 instead of 0x4489 for the sync mark. It is recommended that 0xFE is used as the address mark and 0xFB is used as the data mark. Whatever state the RLL state machine is when it encounters these marks, it will be at zero when it leaves them.

It is likely that the sync and gap lengths can be considerably shortened without effecting reliable operation if "custom" disk formats are used. The PLL tends to lock much quicker than the time provided by the twenty-four byte gap of the Westrn Digital format.

9.1.7.3 CD Format Example. -

```
* Set the PLL to the appropriate speed (see dskpll.doc)
* ADKCONX  - syncen1,0 = 3  (8 bit sync)
           - fasta,fastb =1
           - bothedges=1    (NRZ i/o format)
           - lsbfirst=1
           - incsync =1
           - mode =  2
           - code= 3 (Biphase Mark)
* DMACONX  - disk on- 0x80000210
* DSKSYNCX - 0x9C (read)
* DSKSYNCX - 0x9C36 (write)
```

```
* DSKGAP   - sectcnt= 4, or whatever
* DSKPTX   - point at first sector data to send/receive:
* each data:
        31 - parity (all but sync samples)
        30 - channel status
        29 - text,etc (sub channel code)
        28 - validity
      27-8 - 20 bits of data (27 is MSB)
       7-4 - aux data
       3-0 - sync
             read: (B->5 M,W->9)
             write: 0 for B, 1 for M, 2 for W
pattern of data identified by sync marks: B,W, 191*(M,W)
* DSKHDRPTX- point to second sector data
* DSKLENN  - len3= 2*192 = 384
           - read or write as appropriate
           - DMAon
```

An DSKBLK interrupt will be generated at the end of every sector. At this point the DSKHDRPTX can be rewritten to point at the next sector, and the interrupt reset. If it is desired to go on for more than the maximum capacity of DSKGAP (63 sectors), the sector counter can be rewritten at this time as well.

9.1.7.4 Trackplus Example. -

```
* Set the PLL to appropriate speed, say 3 ticks
* ADKCONX  -  shortpulse=1  one tick width output, for fast stuff
             all precomp bits low
          -  syncen1,0 = 2 (32 bit sync) for read, 0 for write
          -  fasta=fastb=1
          -  lsbfirst=0
          -  incsync=0
          -  bothedges=0
          -  mode= 3
          -  code= 1 (MFM)
* DMACONX  -  disk dma on
* DSKCRCI  -  0xE295
* DSKGAPX  -  sectcnt=number of sectors per track, 4 for this example.
* DSKSYNCX -  0x44894489 for write
          -  0x44895545 for read
* DSKHDRPTX-
    write:    8 bytes of '00' (64 ones to sync up PLL- less maybe ok)
              4 bytes of '000000FB' sync hardware fills in first three
                bytes
              4 bytes of '0000FFFF' crc hardware fills in first two
                bytes
              (this 8,4,4 pattern repeated for each sector on track)
    read:     DSKHDRPTX is unused.
* DSKPTX   -  write: normal data, sectors appended to each other
          -  read:  normal data, sectors appended to each other
* DSKLENN  -  len1= 4 (for write. on read len1 is unused) length of
                gap in words
          -  len2= 2 (for write. on read len1 is unused) length of
                sync mark in words
          -  len3= 512  specifies 1k byte sectors
```

This example generates the following 4 sector disk track (note that the CRC is automatically generated/checked by the disk controller):

```
    8 bytes '00'
    4 bytes 'A1* A1* A1* FB'
 1024 bytes  data...
    2 bytes  CRC
    2 bytes  'FFFF'
       .
       .        8,4,1024,2,2 pattern repeated for each sector
       .
```

After a read operation, check DSKBYTRX to make sure that there were no CRC errors.

9.1.8 Setting The Disk Phase Locked Loop Parameters. -

This section contains a description of the whys and wherefores of setting the disk controller phase locked loop (PLL) parameters. PLL parameters are set by writing the DSKPLLF and DSKPLLP registers. The main parameters that govern loop operation are:

fmax	0-0x7ff	dskpllf(0x224),	bits	27-16	max frequency
fmin	0-0x7ff	dskpllf(0x224),	bits	11-0	min frequency
writeper	0-0x07f	dskpllf(0x224),	bits	31-28, 15-12	period of write pulse
corrpos	0-0x7ff	dskpllp(0x220),	bits	27-16	plus phase gain
corrneg	0-0x7ff	dskpllp(0x220),	bits	11-0	minus phase gain
fgentle	0-3	dskpllp(0x220),	bits	30,29	freq gain
varypll	0,1	dskpllp(0x220),	bit	28	allow variable parameters
tstreset	0,1	dskpllp(0x220),	bit	31	load min freq-> freq
fasta	0,1	adkcon,x(0x09E)	bit	8	2* speed
fastb	0,1	akdconx(0x28C)	bit	7	4* speed

See appendix F, REGBITS, for other details of the DSKPLLF and DSKPLLP register bits.

9.1.8.1 Asynchronous Clocking. -

The disk controller should work with the C28 clock set to values other than 28.63636 Mhz. If this is changed, the default settings may no longer work with a standard disk, and variable settings for the PLL will have to be used for reading/writing standard 1M,2M,4M floppies.

9.1.8.2 FASTA, FASTB Bits. -

FASTB only effects the clock, and has no effect on loop stability. FASTA doubles the speed of the loop internally, and halves the phase feedback to keep the loop stability constant. Watch out: f the frequency number is too large with FASTA low, the phase feedback from one correction can run into the phase feedback of another correction. Phase feedback can be a maximum of 8 CPLL ticks long.

Many common drives can be handled using only FASTA, FASTB:

	FASTA	FASTB
1M MFM floppy	0	1
2M MFM floppy	1	0
4M MFM floppy	1	1

9.1.8.3 Default Settings. -

dskpllf 0x32848210
dskpllp 0x03fc007c

If FASTA=1 (common usage)

```
    fmax= 0x284  644  +60        +10%       lock in freq max
~center=         584  8192/584= 14.01 CPLL ticks
    fmin= 0x210  528  -60        -10%       lock in freq min
writeper= 0x038   56   56/4=     14 CPLL ticks
 corrpos= 0x3fc 1020  +436       phase gain
 corrneg= 0x07c  124  -460
 fgentle= 0                  freq counts by 4 every correction
```

If FASTA=0, the freq and writeper number of ticks are simply doubled.

9.1.8.4 Parameter Effects On Loop Settling And Stability. -

9.1.8.4.1 Writeper. -

This parameter has little affect on loop stability or settling. When writing to the disk, the write bit time counter is used. This counter resets the PLL sum to zero every time the counter times out. Read data is disabled from affecting the loop. This may affect initial conditions, but will have little effect on parameter settings.

9.1.8.4.2 Fmax And Fmin: - FMAX and FMIN should normally be set to the same plus and minus percentage from the desired nominal (center) frequency. If they are too narrow, the variations in the source or system clock may pull the data out of the lock range. If they are too wide, the lock in time may be too long, or the wrong freq can be locked into.

The approximate settling time (number of '1' bits in) is:

(fmax-fmin)/(3* 4/(2^fgentle)) = 9.7 bits (standard settings)

'3' is average corrections/bit
'4' is freq change for one correction at fgentle=0.
It goes down by a factor of two with each increase of one in fgentle.

This number is AFTER the pulse is locked into one window. Normally this happens within one or two bits as long as the range is reasonably narrow and there's only one or two zeros between the ones that sync the phase locked loop.

Using standard settings, full range settling will occur within (644-528)/(3*4) = 9.7 '1' bits.

Another consideration of fmax, fmin is that they should be well inside the phase gain numbers. The phase gain should be around twice as wide as the frequency range, or maybe 1.5 times if stretching it. The problem here is that when the loop is settling to one end of the range and the phase gain is too small, the + and - phase corrections are not symmetrical as they are when locked in the center.

Since data pulses do not come every bit window, if the setting of fmax, fmin is too wide, the loop may lock into the wrong multiple of the pulse separation. Take the example of the freq range from 8 to 14 CPLL ticks, and the sync pulses come in every 3 windows of 12 ticks each. The loop may lock at 9 ticks, corresponding to 4 * 9 tick windows, or at 12 ticks, corresponding to 3 * 12 tick windows.

9.1.8.4.3 Corrpos And Corrneg. -

The larger corrpos and corrneg are, the faster the PLL locks onto one window, allowing the frequency settling to proceed apace. However, a large phase gain can keep the frequency from settling to exactly the correct value, although the error is generally small. The smaller the phase gain, the more forgiving the loop is of incorrect placement of any one pulse (less noise sensitive). The gain, like fmax and fmin, should be placed symmetrically around the center frequency. If the phase gain is set too small, the loop will not settle into one frequency, but oscillate up and down, making it difficult to receive data correctly. The value of approx +-450 or so in the standard setup is a conservatively stable number for that setup. The phase gain can be reduced about 1.41 times for each increase of one in fgentle to maintain about the same stability. The phase gain can also be reduced a bit if the center frequency is higher, and should be increased for a lower center frequency.

9.1.8.4.4 Fgentle. -

FGENTLE values of 0, 1, and 2 give 4, 2, and 1 counts respectively of the frequency per correction. Programming a '3' into fgentle is the same as programming a '2'. The higher numbers of frequency gentle allow less noise sensitivity while maintaining loop stability.

9.1.8.4.5 Bit Density And Loop Settling. -

Most disk formats specify '1' bits as close together as the format allows in the gaps to cause the controller phase locked loop to sync up quickly In MFM, the raw bit pattern is 101010..., and in RLL(2,7), the raw bit pattern is 100100100... The more widely spaced the '1' bits are, the longer the PLL takes to settle. Simply speaking, it takes a certain number of '1' bits to cause the number of corrections necessary to lock in the loop. If the '1' bits are spaced twice as far apart, it takes the PLL twice as long to lock.

There is another effect which needs to be checked when new PLL settings are designed. As the '1' bits in the data stream move further and further apart, the loop becomes less and less stable. If the PLL

settings are not designed properly for a given disk format and density, it is possible to loose the loop lock when the maximum permissible number of '0' bits between '1' bits are encountered in the data stream. Any new set of PLL parameters should be rigorously tested prior to production release. The minimum '1' bit, maximum '0' bit pattern which can occur using the RLL(2,7) code is 10000000100000001. The minimum '1' bit, maximum '0' bit pattern which can occur using the MFM code is 100010001.

9.1.8.5 Simulation Of New PLL Parameters. -

Section 9.1.12, 'Code Permitting Software Simulation of PLL Settings' contains a program which enables the developer to run a functional simulation of the phase locked loop. This program generates a graphical output which depicts loop settling and stability attributes based on a set of PLL parameter settings. The settings the program operates on are identical to those which can be programmed in the disk controller circuit. Along with the parameter settings, the program inputs a pulsed input bit stream which represents read data which could be presented to the actual disk controller circuit. When properly set up, this input data file contains a wide range of bit patterns mimicing the actual disk data bit stream. This data should contain many permutations of legal bit combinations, variable number of zeros between ones in the bit patterns, and instances of long streams of maximum zeros between one bits. The program allows the developer to experiment with the various parameters and establish a quick and stable set of parameters for any media which the disk controller will interface. As output, the program draws a graph on a time line of the PLL frequency and phase throughout the test. From this graph, it can be easily determined how quickly the controller can lock onto the bit stream, and whether or not any data sensitivity problems exist. Once an acceptable set of parameters has been established, the developer can try the settings on the actual hardware.

The program has a text output mode. This mode is enabled by setting 'dographics' to 0. The fmin and fmax varibles set the range for graphics drawing, and the dskpllf and dskpllp registers are set with hex numbers. These are located near the beginning of the code. The program runs under Borland Turbo C on an IBM PC or compatible with VGA monitor. The graphics routines are simple. They could easily be modified to run on other machines or with other compilers.

```
sdskpllg.c -- shell that runs it all, does graphics
sumfreq.c  -- unmixes the sum/freq pipeline
dskpll.c   -- the circuit
```

9.1.8.6 Example: Establishing PLL Parameters For RLL(2,7). -

Following is an excerpt from an engineering notebook which documents one procedure used to establish new PLL parameter settings.

RLL(2,7) setting for standard 1,2,4M drives.

bit windows are 2/3 the size of MFM ones for the same drive:

14/(2/3) = 9.33 CPLL ticks 9.5 is closest available (-1.0% speed)

writeper= 9.5 * 4 = 38 (0x26)

center freq= 8192/9.5 = 862, 0x35E

Let's keep the range narrower, as genlock no longer moves the system clock:

+-6 % 913 (0x392), 813 (0x32D) fmax,min

Let's set fgentle to 1, as RLL is more sensitive to phase error than MFM, so we'll keep corrections more gentle.

450/1.41= 319 1181 (0x49D) 543 (0x21F) corrpos,corrneg

checking phase corr > 2* freq range (1181-543)/(913-813)= 6.38 (ok)

full scale range settling (913-813)/(3* (4/2))= 16.7 '1's

We'll set the tstreset bit high, so the freq will start at the min and show the settling better, avoiding the wait for the PLL to get into the freq range which would happen otherwise.

Leave readsum low- reading the DSKPLLR (0x230) register will then read the current freq of the loop, giving some hint of whether the loop is settled. readsum high is used for chip testing.

varypll, of course, is high

Tallying things together:

(0x224)DSKPLLF 0x2392632D (written first)
(0x220)DSKPLLP 0xB49D021F

To run the program with these settings, the following was varied
ticksperloop = 9.5
loopsperbit= 3 (to start, 8 later)
mem[1]-> dskpllf settings
mem[2]-> dskpllp settings
fmin, fmax 862+-150 (the 862 stuck in)

Running the program showed very quick and stable settling with loopsperbit=3, with settling in around 9 ones after getting into the freq range. (~ 1/2 full scale slew for settling)

At loopsperbit=8, it took longer to settle, and there was some ringing (freq overshoot, then down again), but it settled out after a one big overshoot, then a couple more small ones. This should be acceptable stability. To show how the phase gain affect this, it could be increased for more stability, decreased for less stability.

The integer tweak on the end of the n= equation was also explored, changing it to -2,-3 and +0,1,2. This gives an idea of how the loop will settle into slightly different frequencies. It is easy to get the freq out of range by moving it too much.

9.1.8.7 PLL Settings For Other Formats. -

This section defines PLL parameter settings used to drive media other than floppy disk drives. These parameter sets have been established and checked using the program described in the previous section. As of this writting, no field checks have been made on any of the settings described in this section.

Media Specific Information

format	ticks (clock cycles)	speed	error	rate	
CD	5	44.74khz	1.5% over	44.1 khz	(one sample each channel)
DAT	4.5	49.72khz	3.6% over	48 khz	"
digital radio	7	31.96khz	0.1% under	32 khz	"
ST-506	3	4.77Mhz	4.6% under	5 Mhz	one data bit

9.1.8.7.1 CD,DAT, Digital Radio. -

The Biphase Mark code is highly self clocking and has only one zero between ones maximum in the normal code. One of the sync marks, however, leaves a possibility of 3 zeros in a row, the same as MFM. Between 0 and 3 zeros in a row can exist in all these codes. CD, DAT, and digital radio all use the same code. However, they all run at different speeds.

CD (5 ticks)

writeper- 5*4= 20 (0x14)

center freq- 8192/5 = 1638
+- 5% -- 1720 (0x6B8), 1560 (0x618) fmax,fmin

phase gain +-450 2088 (0x828), 1188 (0x4A4) corrpos,corrneg
phase gain/freq range check (2088-1188)/(1720-1560) = 5.65 (ok)
full range settling (1720-1560)/(3*4)= 13.3 '1's

dskpllf 0x16b84618
dskpllp 0x982804a4

Testing with a '10001000' bit pattern (loopsperbit=4) showed settling to 1/2 scale in 7 or 8 bits with somewhat overdamped settling (no ringing, gradual approach to the final frequency value). If noise reduction is important, the phase gain could be reduced somewhat. However, CD data will probably come out of buffered and crystal controlled sources. This is probably not too important; leave it as it is.

DAT (4.5 ticks)

writeper 4.5*4 = 19, 0x13

center freq 8192/4.5 = 1820 +-7% 1947 (0x79B) 1700 (0x6A5)
extra range so a normal speed DAT (-3.6%) can be read ok.

phase gain +-450 2270 (0x8DE) 1370 (0x55A)
phase gain/freq range check 3.6 (ok)
full range settling 20.5 bits

dskpllf 0x179B36A5
dskpllp 0x98DE055A

Digital Radio (7 ticks)

+-5% freq, +-300 phase correction

dskpllf 0x14CCC45B
dskpllp 0x95BE0366

The gentler phase gain specified for the digital radio may be better for the more variable bits coming off the air. This has been checked stable using a '10001000' bit pattern against the simulator of section 9.1.12, 'Code Permitting Software Simulation of PLL Settings'.

9.1.8.7.2 ST-506 MFM Hard Drive. -

Hard drive bit streams push the disk controller circuits to the limit in terms of bit speed (3 ticks). At this speed, time discretization noise of the bits is large, so corrections should be as gentle as possible. The +-450 standard phase gain gives more than enough loop stability. The gain has been reduced as much as is reasonable while watching for good stability as well as keeping the phase gain numbers around 1.5 times the frequency range.

1) +-3% freq, +-170 phase center freq 2730, fgentle=1
phase/freq ratio 2.09
settling time (2813-2651)/(4*2) = 20.2 bits
Originally designed for fgentle=2, it is stable at fgentle=1, and settles twice as fast. If even gentler feedback than this is needed, use +-2% freq +-80 phase gain, similiar to 2) below.

dskpllf 0x0afdca5b
dskpllp 0xbb540a00

2) center freq +4.6%, to read a normal ST-506 disk
+-2% freq +-80 phase gain, center freq 2856, fgentle=2
phase/freq ratio 1.41
settling time (2913-2800)/4 = 28.2 bits

```
dskpllf 0x0b61caf0
dskpllp 0xdb780ad8
```

This rings a bit, but is stable. It is still stable with even lower phase gain (+-60), but the phase/freq ratio is the limiting factor.

9.1.9 Disk Controller Data Rate Limitations. -

In addition to potential PLL stability problems at high data receive rates, system DMA latency impacts maximum acheivable bit rates. The following figures derive the maximum disk bit rate as a function of system DMA latency.

Link latency 1.43us max (MARY to ANDREA serial DMA link, 9-20 BUSCLK cycles)

Andrea dma latency
min 4 ticks .28us (cycle runs immediately)
max disk 2.23us (bitplane fetch) (in high end system-> .28us)
.58 (overlay)
.28 (refresh)
.28 (int transfer cycle)
3.63us total

total disk dma latency including effect of fifo:

(link + andrea + (andrea-bitplane))/2 = 3.23us

dma data transfer up to 32/3.23 bit/us = 9.9 Mbits/sec (poof)

data sep speed 2.5 ticks of 28M- 11.4 Mbits/sec
(this is raw bits, but it may get a bit dodgy with 2 1s in a row at > 5.7 Mbits/sec or so. Window margin may be a bit dodgy near the high end. Set the feedback to very gentle and try it !)

theoretical speed using PLLRST; 2 ticks of 28M - 14.3 Mbits/sec

raw transfer rate is 2* encoded rate

Ethernet	10	Mbits/sec
hard disk encoded	5	
CD encoded	2.8	
4 Mbyte drives raw	2	
1 M floppy raw	0.5	(max of Paula controller)

9.1.10 C Routine Which Generates CRC16. -

```
/* CRC generator */

main()
```

```
{
int set,pcrc,crc,i,j,pinsr,insr,poutsr,outsr,di,dout,y,jmax;
int pusecrc,usecrc;
int mem[]={0xa1a1,0xa1fb,0x0000,0xffff};
insr=0;
set=1;
crc= 0xffff;
jmax=2;
for (j=0;j<=(jmax+1);j++)
 {
 for (i=0;i<16;i++)
  {
  if (i==0) pinsr= mem[j];
  else pinsr= insr << 1;
  di= (insr&0x8000)!=0;
  dout= (crc&0x8000)!=0;
  pusecrc= (j>jmax);
  if(usecrc) di=dout;
  y= (crc==0);
  if (!set)
   {
   pcrc = crc << 1;
   pcrc= pcrc + (di^dout);        /* bit  0 xor */
   pcrc= pcrc ^ (32*(di^dout));    /* bit  5 xor */
   pcrc= pcrc ^ (4096*(di^dout));  /* bit 12 xor */
   }
  else
   {
   pcrc= 0xffff;
   }
  poutsr= (outsr<<1) + di;
  printf("%04x %04x|",pcrc,poutsr);
  /*if (i%8==0) printf("\n");*/
  /* ffs setting */
  crc= pcrc;
  insr= pinsr;
  outsr=poutsr;
  usecrc=pusecrc;
  set=0;
  } /* end of i loop */
 }/* end of j loop */
printf("%x end\n",(outsr<<1)+((crc&0x8000)!=0));
}
```

9.1.11 C Routine Which Encodes/Decodes RLL(2,7). -

```
/* RLL 2,7 encode, decode */

#include <stdio.h>

main()
 {
 int j,bit,outsr,poutsr,outsr2,poutsr2,outsr3,poutsr3;
 int clk,prlls1,prlls0,firstbit,secbit,rlls,raw,poutsr2d,outsr2d,out;
 int prawd,prawsr,rawd,rawdd,rawsr,wdld,shft,pclk;
 int plasthalf,lasthalf,plasthalfbit,lasthalfbit,pcnt,cnt;
 int pshiftcnt,shiftcnt,equal,bits,psbit1,psbit0,pstate,state;
 int pinsr,insr,pinsr2,insr2,pinclk,inclk,data,pinsr3,insr3,pinsr4,insr4;
 int mem[17]={0x0000,0xffff,0x8880,0x4048,
```

```
        0xaaaa,0xffff,0x0000,0x0000,0x0000,0x4924,0x9249,0x2492,
        0x6db6,0xdb6d,0xb6db,0x2222,0x3333};
    int check[8],k,l;
    unsigned int i;
    k=l=0;
    j=0;
    outsr3=0;
    raw=1;
    shiftcnt=0;
    lasthalf=1;
    lasthalfbit=1;
    rawsr=0xffff;
    srand(7);
    for(i=0;j<32767;i++)
     {
     shft= (i%14==0) && lasthalfbit;
     plasthalfbit= ( (rawsr&0x18)==0x18)
       || (i%14==0 && !lasthalfbit)
       || (i%14!=0 && lasthalfbit);
     if (shft) pshiftcnt= shiftcnt - 1;
     wdld= shft && (shiftcnt==0);

     if (wdld)
      {
      raw= (j<=3);
      if (j==4) k=0;
      if (j<=16) poutsr= mem[j];
      else poutsr= rand();
      /*printf("(%x)",poutsr);*/
      check[k]= poutsr;
      j++;
      k= (k+1) % 8;
      pshiftcnt=15;
      }
     else if (shft & !wdld) poutsr=(outsr << 1);
     if (shft)
      {
      poutsr2=((outsr2 << 1)+((outsr&0x8000)!=0))&0xf;
      prawsr= (rawsr << 1) + raw;
      }
     pclk= shft;
     prlls1= ((outsr2&0xC)==0 && clk && rlls==0) || (!clk && rlls==2);
     prlls0= ((outsr2&0x8)==8 && clk && rlls==0)
          || ((outsr2&0xC)==0 && clk && rlls==2) || (!clk && rlls==1);
     firstbit= ((outsr2&0xE)==0 && rlls==0)
          || ((outsr2&0xE)==6 && rlls==2)
          || ((outsr2&0xC)==8 && rlls==0);
     secbit= ((outsr2&0xC)==0 && rlls==2)
          || ((outsr2&0xC)==0xC && rlls==0);
     if (clk) poutsr3= firstbit + 2*secbit;
     else if (i%14==1 && !clk) poutsr3= outsr3 >> 1;
     poutsr2d= (outsr2 >> 3);
     prawd= (rawsr & 0x10)==0x10;
     out= (rawd && outsr2d) || (!rawd && outsr3%2);
    /* if (i%14==3) printf("%x%x",rawd,out);
     if (i%(14*8)==0) printf(" ");*/

     /* receive stuff */
     equal= insr2==0x4048 /* && (insr4&0x00ff)==0x80 */;
     if (i%14==0)
      {
      pinsr= (insr << 1) + out;
```

```
        pinsr2= (insr2 << 1) + ((insr & 0x0002)!=0);
        pinsr4= (insr4 << 1) + ((insr2 & 0x8000)!=0);
        pinclk= !inclk && !equal;
        if (inclk)
         {
         bits= 4*(0x3 & insr2) + (0x3 & insr);
         psbit1= ((bits==9 || bits==4) && state==0 && inclk)
              || ((state==2 || state==3) && !inclk);
         psbit0= ((((bits==9 || bits==8 || bits==2 || bits==0) && state==0)
              || (bits==4 && state==3))&& inclk)
         || (bits==8 && state==1 && inclk)
         || ((state==1 || state==3) && !inclk);
         pstate= 2*psbit1 + psbit0;
         data= (bits==8 && state==0)
         || (bits==4 && state==0) || (state==2)
         || (bits==8 && state==1);
         pinsr3= (insr3 << 1) + data;
         pcnt= cnt+1;
         }
        if(equal)
         {
         printf("sync ");
         l=0;
         }
        if ( inclk && pcnt%16==0)
         {
         if (j%500==0 || (check[l]!=pinsr3))
           printf("%6d %04x %04x \n",j,check[l],pinsr3);
         l= (l+1) % 8;
         }
        }
       if (equal)
        {
        pcnt=0;
        pstate=0;
        }
       /* ff settings */
       rlls= (!prawd)*(2*prlls1 + prlls0);
       outsr= poutsr;
       outsr2= poutsr2;
       outsr3= poutsr3;
       rawdd=rawd;
       outsr2d=poutsr2d;
       rawd=prawd;
       rawsr=prawsr;
       clk= pclk;
       lasthalf=plasthalf;
       lasthalfbit=plasthalfbit;
       shiftcnt=pshiftcnt;
       insr=pinsr;
       insr2=pinsr2;
       insr3=pinsr3;
       insr4=pinsr4;
       inclk=pinclk;
       state=pstate;
       cnt=pcnt;
       }
      printf("end\n");
      }
```

9.1.12 Code Permitting Software Simulation Of PLL Settings. -

See section 9.1.8.5, 'Simulation of New PLL Parameters', for a written description of this code.

9.1.12.1 SDSKPLLG.C. -

```
/* shell to run dskpll.c */
/*   9jan90  gjk */
/*  10jan90  graphic version gjk */

#include <stdio.h>
#include "dskpll.c"
#include "sumfreq.c"
#include "graphics.h"

main()
 {
 int i,j;
 float ticksperloop=9.5;
 int loopsperbit=8;
 int n=4*ticksperloop*loopsperbit-0;
 float realticksperloop= (float)n/(4*loopsperbit);
 long int mem[7]={0x00000000,0x2392632d,0xb49d021f,0xffffffff,0x7fffffff
  ,0x00008280,0x7fffffff};
 int rgain[7]={0x000,0x224,0x220,0x230,0x290,0x230,0x230};
 int cpllr,cpllrd,cpllf,cpllfd,c14r,c14rd,c14f,c14fd;

 /* read write variables */
 int /*c14r,*/rst,alei,strobe,rga,dsken;
 long int dbw,sum;
 long int dbr,dbdrivers,phasereg,freqreg;
 int tstreset,varypll;

 /* loop variables */
 int /*cpllr,*/rstf,sepdisf,fastaf,PLLRST,pulse,early;
 /*long int phasereg,freqreg;*/
 int dkrckf,dkrdatf,longbit,freq;
 /*long int *sum;*/
 int ppulse,pearly,pulsed;
 /* int tstreset,varypll */

 /* printing variables */
 char str[100];
 int realfreq;
 unsigned int realsum,realsumh;

 /* graphics variables */
 int gdrive=9;
 int gmode=2;
 int fmin=862-150;
 int fmax=862+150;
 int foff=0;
 int fscale=0;
 int xscale=1;
 int dographics=1;
 int oldfreq=0;
 int oldsum=0;
 int k=0;
 int oldk=0;
```

```
if (dographics)
 {
 initgraph(&gdrive,&gmode,"\\tc");
 fscale= 480/(fmin-fmax);
 foff= 480-fscale*fmin;
 moveto(0,foff+fscale*(0x248));
 }

cpllrd=0;
c14rd=0;
rst=1;
alei=1;
strobe=0;
rga=0;
dbw=0;
sum=0;
dbr=0;
rstf=1;
dbdrivers=0;
phasereg=0;
freqreg=0;
tstreset=0;
varypll=0;
c14fd=0;
cpllfd=0;
pulsed=0;
pearly=0;


early=0;
pulse=0;
fastaf=1;
sepdisf=0;
dsken=0;
PLLRST=0;

/* main loop */
for(i=0,j=0;i<=16000;i++)
 {

 /* system clocks (end of cycle activity) */
 /*  do logic before FFs on these clocks */

 c14r= (i&7)==0;
 c14f= (i&7)==4;
 cpllr= (i&3)==0;
 cpllf= (i&3)==2;

 /* circuitry */


/* loop itself clocked on CPLLR  */
if(cpllr || cpllrd)
dskpll(cpllr,rstf,sepdisf,fastaf,PLLRST,pulse,early /* inputs */
     ,phasereg,freqreg,tstreset,varypll      /* inputs */
     ,&dkrckf,&dkrdatf,&longbit,&freq,&sum);/* outputs */

/* processor read/write section clocked on C14R & CPLLF & on sum changes */
/* cpllrd is to clock sum in-- run AFTER the pll section */
if (c14r || c14rd || c14f || c14fd || cpllrd || cpllf || cpllfd)
dskpllrw(c14r,c14rd,c14f,c14fd,cpllf,cpllfd
```

```
      ,rst,alei,strobe,rga,dbw,freq,sum,dsken /* ins */
      ,&dbr,&dbdrivers,&phasereg,&freqreg,&tstreset,&varypll); /* outputs */

 /* clocked on CPLLR */
 if (cpllr || cpllrd)
 sumfreq(cpllr,freq,sum     /* inputs */
       ,&realfreq,&realsum,&realsumh);  /* outputs */

  /* print statements go here */
  /*
  if ((i&3)==1 && !dographics) printf("%x %3x %8lx %8lx %lx %8lx %8lx %8lx\n"
        ,strobe,rga,dbw,dbr
        ,dbdrivers,phasereg,freqreg,sum);
 */
  if ((i&3)==1 && !dographics) printf("%d,%x%x %3x %4x %4x %x%x%x %x%x%x%x"
      ,i,pulse,early,realfreq,realsum,realsumh
      ,dkrckf,dkrdatf,longbit ,tstreset,varypll,strobe,cpllr);

  if ((i&3)==1 && !dographics) printf(" %8lx",dbr);
  if ((i&3)==1 && !dographics) printf("\n");

  if ((i%20)==1 && dographics)
   {
   k++;
   moveto(xscale*k,foff+fscale*oldfreq);
   lineto(xscale*(k+1),foff+fscale*realfreq);
   oldfreq=realfreq;
   }
  if (pulsed && (i&3)==1)
   {
   if (early) realsum=realsumh;
   if (!fastaf) realsum=realsum/2;
    else realsum= realsum & 0x1fff;
   if (!dographics) printf("(%x)%x",realfreq,oldsum/64);
   else
    {
    moveto(xscale*oldk,480-oldsum/64);
    lineto(xscale*(k+1),480-realsum/64);
    oldsum=realsum;
    oldk=k+1;
    }
   }

 /* system clocks (beginning of cycle activity) */
 /* set FFs & change outputs to ext stuff on these clocks */
 /* also do any logic after internal FFs */
 /* do nothing that has logic dependency on inputs */

 c14rd= c14r;
 c14fd= c14f;
 cpllrd= cpllr;
 cpllfd= cpllf;

 /* system signals */

 /* n set above */
 pulsed=pulse;
 ppulse=(i%n<=3);
 pearly= ((i%n)==0 && ((i%4==2)||(i%4==1))) || (pearly && !(i%n==0));
 if (cpllr)
  {
  pulse=ppulse;
```

```
    early=pearly;
    }
  rst= i<63;
  rstf=rst;
  alei= (i&63)<32;
  strobe= (i&63)<8;
   if ((i&63)==32) rga= rgain[j];
   if ((i&63)==63)
    {
    dbw= mem[j];
    if (j<6) j++;
    }
  }
 sprintf(str,"freq=%d,phase=%x,t/l=%5g,tfreq=%5g\0",realfreq,oldsum/64
      ,realticksperloop,(float)2048*(4/realticksperloop));
 if (dographics)
  {
  moveto(0,10);
  outtext(str);
  }
 else
  printf("%s\n",str);
 }
```

9.1.12.2 SUMFREQ.C. -

```
/* calc for real freq & sum (undo pipeline) for disk pll*/
/* gjk 10jan90 */

/* clocked on CPLLR */

sumfreq(cpllr,freq,sum      /* inputs */
        ,realfreq,realsum,realsumh)   /* outputs */
 int freq,*realfreq;
 long int sum;
 unsigned int *realsum,*realsumh;
 {
 static int pfreq1,freq1,pfreq2,freq2,pfreq3,freq3;
 static long int psum1,sum1,psum2,sum2,psum3,sum3,psum4,sum4;

 if (cpllr)
  {
  pfreq1=freq;
  pfreq2=freq1;
  pfreq3=freq2;
  psum1=sum;
  psum2=sum1;
  psum3=sum2;
  psum4=sum3;
  }
 if (!cpllr) /* (cpllrd) */
  {
  freq1=pfreq1;
  freq2=pfreq2;
  freq3=pfreq3;
  sum1=psum1;
  sum2=psum2;
  sum3=psum3;
  sum4=psum4;
```

```
  *realfreq= (freq1 & 0xf00)|(freq2 & 0xf0)|(freq3 & 0xf);
  *realsum=(sum1 & 0xf000)|(sum2 & 0xf00)|(sum3 & 0xf0)|(sum4 & 0xf);
  *realsumh= (sum1>>8 & 0xf000)|(sum2>>8 & 0xf00);
  }
 }
```

9.1.12.3 DSKPLL.C. -

```
/* Disk Phase Locked Loop for Mary */
/*   start 20dec89  glenn keller */
/*   initial entry complete 8jan90 gjk */
/*   19jan90 gjk c14r->c14f for varypll,tstreset interfaces */
/*   23feb90 gjk mods for freq read, ext PLL reset */
/*   30may90 gjk mods to match timing checked schematic */
/*    4jun90 gjk reset mods for to match schematic (rstmore) */
/*    5jun90 gjk reset mods for to match schematic (zero lower add4 bits)*/
/*       also reset of dkrdatf, non writecnt hangup change */
/*   13jun90 th,gjk delay mph,pph to match schematics */
/*   19jun90 gjk extra pulse for tstresetd to make it work at slow cpll */

#include <stdio.h>
extern FILE *fp2;

/* processor read/write section clocked on C14R, C14F, and CPLLF */
/* also run on CPLLRD to get sum in right-- run AFTER pll section */

dskpllrw(c14r,c14rd,c14f,c14fd,cpllf,cpllfd
        ,rst,alei,strobe,rga,dbw,freq,sum,dsken /* inputs */
        ,dbr,dbdrivers,phasereg,freqreg,tstreset,varypll)  /* outputs */

 int c14r,c14rd,c14f,c14fd,cpllf,cpllfd,rst,alei,strobe,rga,dsken,freq;
 long int dbw,sum;
 long int *dbr,*dbdrivers,*phasereg,*freqreg;
 int *tstreset,*varypll;
 {
 static int dskpllfl,dskpllpl,olddrive;
 static int ptstreset1,tstreset1,ptstreset,ptstreset1d,tstreset1d;
 static int pvarypll1,varypll1,pvarypll;
 int readsum;


 dskpllfl=((rga==0x224)&& !alei) || (dskpllfl && alei);
 dskpllpl=((rga==0x220)&& !alei) || (dskpllpl && alei);

 if (strobe && dskpllfl) *freqreg=dbw;
 if (strobe && dskpllpl) *phasereg=dbw;
 if (rst || dsken)
  {
  *phasereg= *phasereg & 0xefffefff; /* zero varypll and readsum */
  }
 readsum= (*phasereg & 0x1000)!=0;
 if (!alei && rga==0x230)  /* dskpllr */
  {
  *dbr=0;
  *dbr= freq;
  if (readsum) *dbr= sum;
  if (!olddrive) (*dbdrivers)++;
  olddrive=1;
  }
```

```
 else
  {
  if (olddrive) (*dbdrivers)--;
  olddrive=0;
  }

 if (c14r) ptstreset1d= strobe && dskpllpl && (dbw & 0x80000000)!=0;
 if (c14rd) tstreset1d= ptstreset1d;
 if (c14f)
  {
  ptstreset1=strobe && dskpllpl && (dbw & 0x80000000)!=0 || tstreset1d;
  pvarypll1= (*phasereg & 0x10000000)!=0;
  }
 if (c14fd)
  {
  tstreset1=ptstreset1;
  varypll1=pvarypll1;
  }
 if (cpllf)
  {
  ptstreset=tstreset1;
  pvarypll=varypll1;
  }
 if (cpllfd)
  {
  *tstreset=ptstreset;
  *varypll=pvarypll;
  }
 }

/* loop itself clocked on CPLLR  */

dskpll(cpllr,rstf,sepdisf,fastaf,PLLRST,pulse,early /* inputs */
      ,phasereg,freqreg,tstreset,varypll        /* inputs */
      ,dkrckf,dkrdatf,longbit,freq,sum,rstmore)      /* outputs */

 int cpllr,rstf,sepdisf,fastaf,PLLRST,pulse,early;
 long int phasereg,freqreg;
 int tstreset,varypll;
 int *dkrckf,*dkrdatf,*longbit,*freq;
 long int *sum;
 int *rstmore;
 {
 static int padd0,padd1,padd2,padd3,padd4,padd5;
 static int add0,add1,add2,add3,add4,add5;
 static int pcadd0,pcadd1,pcadd2,pcadd4;
 static int cadd0,cadd1,cadd2,cadd4;
 static int padd2d,add2d,padd4d,add4d;
 static int pmph,mph,pmph1,mph1,pmph2,mph2,pphasem1,pphasem2,pphasem3,pphasem4
            ,phasem1,phasem2,phasem3,phasem4,pmph0,mph0;
 static int ppph,pph,ppph1,pph1,ppph2,pph2,pphasep1,pphasep2,pphasep3,pphasep4
            ,phasep1,phasep2,phasep3,phasep4,ppph0,pph0;
 static int pdkup,dkup,pdkup1,dkup1,pdkup2,dkup2
           ,pdkdn,dkdn,pdkdn1,dkdn1,pdkdn2,dkdn2;
 static int pff1,ff1,pff2,ff2,pff3,ff3;
 static int pc,c,pdkcarryff,dkcarryff;
 static int pphff,phff,pophff,ophff;
 static int prst1,rst1,prst2,rst2,prst3,rst3;
 static int pfreq0,pfreq1,pfreq2,freq0,freq1,freq2;
 static int pfreq0d,pfreq1d,pfreq2d,freq0d,freq1d,freq2d;
 static int ptstresetd,tstresetd,pvaryplld,varyplld,prstmore;
 static int pwritecnt0,writecnt0,pwritecnt1,writecnt1;
```

```
 static int pwriteld,writeld,pwriteld1,writeld1,pdkrckf;
 static int dkrdat1,pdkrdat1,pdkrdatf,plongbit;
 static int pminm,minm,pminc0,minc0,pminc1,minc1;
 static int pmaxp,maxp,pmaxc0,maxc0,pmaxc1,maxc1;
 static int ptimeout0,timeout0;
 static int pc1,pc2,c1,c2,ppulsed,pulsed;
 static int pcorrpos0,pcorrpos1,pcorrpos2,pcorrneg0,pcorrneg1,pcorrneg2;
 static int corrpos0,corrpos1,corrpos2,corrneg0,corrneg1,corrneg2;
 static int psepdisfd, sepdisfd;
 int rst0,dkcarry,writecnt;
 int timeout1,timeout2;
 int writeper,freqgentle0,freqgentle1;
 int freqmin0,freqmin1,freqmin2,freqmax0,freqmax1,freqmax2;
 int phasep,phasem,cntup,cntdn;

 if (cpllr)
  {
  pvaryplld= varypll;
  freqgentle1= (phasereg & 0x40000000) !=0;
  freqgentle0= ((phasereg & 0x20000000) !=0) && !freqgentle1;
  ptstresetd= tstreset || PLLRST;
  prstmore= tstresetd || rstf;

  if (varyplld)
   {
   pcorrneg0= (phasereg >>  0) & 0xf;
   pcorrneg1= (phasereg >>  4) & 0xf;
   pcorrneg2= (phasereg >>  8) & 0xf;
   pcorrpos0= (phasereg >> 16) & 0xf;
   pcorrpos1= (phasereg >> 20) & 0xf;
   pcorrpos2= (phasereg >> 24) & 0xf;
   freqmin0= (freqreg  >>  0) & 0xf;
   freqmin1= (freqreg  >>  4) & 0xf;
   freqmin2= (freqreg  >>  8) & 0xf;
   freqmax0= (freqreg  >> 16) & 0xf;
   freqmax1= (freqreg  >> 20) & 0xf;
   freqmax2= (freqreg  >> 24) & 0xf;
   writeper= (freqreg  >> 12) & 0xf;
   writeper= writeper | ((freqreg >> (28-4)) & 0xf0);
   }
  else
   {
   pcorrneg0= (124 >> 0) & 0xf;
   pcorrneg1= (124 >> 4) & 0xf;
   pcorrneg2= 0;
   pcorrpos0= (1020 >> 0) & 0xf;
   pcorrpos1= (1020 >> 4) & 0xf;
   pcorrpos2= (1020 >> 8) & 0xf;
   freqmin0= (528 >> 0) & 0xf;
   freqmin1= (528 >> 4) & 0xf;
   freqmin2= (528 >> 8) & 0xf;
   freqmax0= (644 >> 0) & 0xf;
   freqmax1= (644 >> 4) & 0xf;
   freqmax2= (644 >> 8) & 0xf;
   writeper=56;
   }
 /*printf("<%2x>",writeper); */

  /* basic loop adder */

 pmph0=mph;
 pmph1=mph0;
```

```
pmph2=mph1;
ppph0=pph;
ppph1=pph0;
ppph2=pph1;
pfreq0d= freq0;
pfreq1d= freq1;
pfreq2d= freq2;

if (mph0) padd0= add0 + corrneg0;
if (mph1) padd1= add1 + corrneg1 + cadd0;
if (mph2)
  {
  padd2= add2 + corrneg2 + cadd1;
  padd4= add2 + corrneg2/2;
  }
if (pph0) padd0= add0 + corrpos0;
if (pph1) padd1= add1 + corrpos1 + cadd0;
if (pph2)
 {
 padd2= add2 + corrpos2 + cadd1;
 padd4= add2 + corrpos2/2;
 }
if (!(mph0 || pph0)) padd0= add0 + freq0d;
if (!(mph1 || pph1)) padd1= add1 + freq1d + cadd0;
if (!(mph2 || pph2))
 {
 padd2= add2 + freq2d + cadd1;
 padd4= add2 + freq2d/2;
 }
padd3= add3 + cadd2;
padd5= add3 + cadd4;

 /* carries */
 pcadd0= (padd0 & 0x10)!=0;
 pcadd1= (padd1 & 0x10)!=0;
 pcadd2= (padd2 & 0x10)!=0;
 pcadd4= (padd4 & 0x10)!=0;

 /* zeroing of adder during writeld times, reset */
 /* also masking off of extra bits */

 rst0=  (sepdisf && writeld) || *rstmore;
 prst1=rst0;
 prst2=rst1;
 prst3=rst2;

 if (rst0) padd0=0; else padd0= padd0 & 0xf;
 if (rst1) padd1=0; else padd1= padd1 & 0xf;
 if (rst2)
  {
  padd2=0;
  padd4=0;
  }
 else
  {
  padd2= padd2 & 0xf;
  padd4= padd4 & 0xc;
  }
 if (rst3)
  {
  padd3=0;
  padd5=0;
```

```
    }
   else
    {
    padd3= padd3 & 0x3;
    padd5= padd5 & 0x3;
    }
   padd4d=add4;
   padd2d=add2;

   /* what to look at for corrections */

  pc1= (add4d >> 2) + (add5 << 2);
  pc2= (add2d >> 2) + (add3 << 2);
  if (early)pc1= pc= (add4d >> 2) + (add5 << 2);
  else      pc2= pc= (add2d >> 2) + (add3 << 2);
  if (!fastaf) pc = pc >> 1;
  else         pc = pc & 0x7;

  /* frequency counter */

  if (dkup) pfreq0= freq0 +(4 - 3*(1 & freqgentle1) - 2*(1 & freqgentle0));
  if (dkdn) pfreq0= freq0 -(4 - 3*(1 & freqgentle1) - 2*(1 & freqgentle0));
  if (!(dkup || dkdn))  pfreq0=freq0;

  if (dkup1) pfreq1= freq1 + 1;
  if (dkdn1) pfreq1= freq1 - 1;
  if (!(dkup1 || dkdn1)) pfreq1= freq1;

  if (dkup2) pfreq2= freq2 + 1;
  if (dkdn2) pfreq2= freq2 - 1;
  if (!(dkup2 || dkdn2)) pfreq2= freq2;

  if (tstresetd) /* note that pipelining is ignored for reset */
   {
   pfreq0= freqmin0;
   pfreq1= freqmin1;
   pfreq2= freqmin2;
   }
  if (rstf) /* note that pipelining is ignored for reset */
   {
   pfreq0= (584 >> 0) & 0xf;
   pfreq1= (584 >> 4) & 0xf;
   pfreq2= (584 >> 8) & 0xf;
   }

  /* freq cntr carries */
   pdkup1= dkup && (pfreq0 & 0x10)!=0;
   pdkdn1= dkdn && (pfreq0 & 0x10)!=0;
   pdkup2= dkup1 && (pfreq1 & 0x10)!=0;
   pdkdn2= dkdn1 && (pfreq1 & 0x10)!=0;

  /* trim off extra freq bits */
  pfreq0= pfreq0 & 0xf;
  pfreq1= pfreq1 & 0xf;
  pfreq2= pfreq2 & 0xf;

/* freqmin,max stuff */

  pminm=  (freq2 < freqmin2) || ((freq2==freqmin2)&& minc1);
  pminc1= (freq1 < freqmin1) || ((freq1==freqmin1)&& minc0);
  pminc0= (freq0 < freqmin0);
  pmaxp=  (freq2 > freqmax2) || ((freq2==freqmax2)&& maxc1);
```

```
pmaxc1= (freq1 > freqmax1) || ((freq1==freqmax1)&& maxc0);
pmaxc0= (freq0 > freqmax0);


/* feedback correction */
ppulsed= pulse && !sepdisf;

cntup= (pulsed && (c < 4) && !phff && !maxp && !ophff)
      || (ff1 && !phff && !ophff && !maxp)
      || (minm);
pdkup=cntup;
cntdn= (pulsed && (c>= 4) &&  phff && !minm &&  ophff)
      || (ff1 &&  phff &&  ophff && !minm)
      || (maxp);
pdkdn=cntdn;
phasep= (pulsed && (c < 4))
      || (ff1 && !phff);
phasem= (pulsed && (c>=4))
      || (ff1 &&  phff);

pff1= (pulsed && ( c<3 || c>4)) || ff2;
pff2= (pulsed && ( c<2 || c>5)) || ff3;
pff3= (pulsed && ( c>6 || c<1));

if (pulsed)
 {
 pphff= (c>3);
 pophff= phff;
 }
if (rstf || tstresetd)
 {
 pphff= 0;
 pophff=0;
 }

dkcarry= !(c&4)&&(dkcarryff&4)
      || !(c&4)&&!(c&2)&&(dkcarryff&2)
      || (dkcarryff&4)&&(dkcarryff&2)&&!(c&2); /* 2 bit greater than */
pdkcarryff= c;
pdkrdat1= pulsed || (dkrdat1 && !dkcarry && !(*rstmore));
/* delay & hold to match clock delay */
pdkrdatf=( dkrdat1 && dkcarry || *dkrdatf && !dkcarry ) && !(*rstmore);

/*fprintf(fp2,"{%x%x}",pulsed,c);*/

/* extra phase correction when not fastaf */
pphasem1=phasem;
pphasem2=phasem1 && !fastaf;
pphasem3=phasem2;
pphasem4=phasem3;
pmph=phasem4 || phasem;
pphasep1=phasep;
pphasep2=phasep1 && !fastaf;
pphasep3=phasep2;
pphasep4=phasep3;
ppph=phasep4 || phasep;

/* write counter */

if (writeld || *rstmore) pwritecnt0= (writeper >> (1 + fastaf))&0xf;
else          pwritecnt0= (writecnt0 - 1)&0xf;
if (writeld1) pwritecnt1= (writeper >> (5+fastaf))&0x7;
```

```
  else          pwritecnt1= (writecnt1 - timeout0)&0x7;

  ptimeout0= (writecnt0==0);
  timeout2= (writecnt0==2);
  timeout1= (writecnt0==1);
  psepdisfd= sepdisf;

  pwriteld= rstf
    || (timeout2 && sepdisf && !(*longbit) && (writecnt1==0))
    || (timeout1 && sepdisf &&  (*longbit) && (writecnt1==0))
    || ((dkcarry || sepdisf) && !sepdisfd);

  plongbit= (writeld ^ *longbit)&& !(*rstmore)
    &&( (writeper&1)&& !fastaf || ((writeper&2)!=0 && fastaf) );
  /*
  fprintf(fp2,"|%x%x%x %x%x%x %x%x|",*longbit,writecnt1,writecnt0
     ,writeld,writeld1,timeout0,dkcarry,sepdisf);
  */
  pdkrckf= pwriteld;  /* same signal */
  pwriteld1=writeld || *rstmore;
  /*
  *sum= add0 + (add1 << 4) + (add2 << 8) + (add3 << 12)
       + ((long int)(add4) << 16) + ((long int)(add5) << 20);
  *freq= freq0 + (freq1 << 4) + (freq2 << 8);
  writecnt= writecnt0 + (writecnt1 << 4);

  fprintf(fp2,"|%x%x %x%x%x%x %x %4x %6lx %x%x %2x|\n",pulse,early
      ,mph,pph,dkdn,dkup
      ,c,*freq,*sum,*dkrckf,*dkrdatf,writecnt);
  */
  }

 if (!cpllr)  /* ffs setting, logic after FFs */
  {
  freq0d=pfreq0d;
  freq1d=pfreq1d;
  freq2d=pfreq2d;
  corrpos0=pcorrpos0;
  corrpos1=pcorrpos1;
  corrpos2=pcorrpos2;
  corrneg0=pcorrneg0;
  corrneg1=pcorrneg1;
  corrneg2=pcorrneg2;
  tstresetd=ptstresetd;
  *rstmore=prstmore;
  varyplld= pvaryplld;
  phasem1=pphasem1;
  phasem2=pphasem2;
  phasem3=pphasem3;
  phasem4=pphasem4;
  mph=pmph;
  mph0=pmph0;
  mph1=pmph1;
  mph2=pmph2;
  phasep1=pphasep1;
  phasep2=pphasep2;
  phasep3=pphasep3;
  phasep4=pphasep4;
  pph=ppph;
  pph0=ppph0;
  pph1=ppph1;
  pph2=ppph2;
```

```
add0=padd0;
add1=padd1;
add2=padd2;
add3=padd3;
add4=padd4;
add5=padd5;
add2d=padd2d;
add4d=padd4d;
cadd0=pcadd0;
cadd1=pcadd1;
cadd2=pcadd2;
cadd4=pcadd4;
rst1=prst1;
rst2=prst2;
rst3=prst3;
c=pc;
c1=pc1;
c2=pc2;
dkup=pdkup;
dkup1=pdkup1;
dkup2=pdkup2;
dkdn=pdkdn;
dkdn1=pdkdn1;
dkdn2=pdkdn2;
freq0=pfreq0;
freq1=pfreq1;
freq2=pfreq2;
minm=pminm;
minc0=pminc0;
minc1=pminc1;
maxp=pmaxp;
maxc0=pmaxc0;
maxc1=pmaxc1;
phff=pphff;
ophff=popphff;
ff1=pff1;
ff2=pff2;
ff3=pff3;
pulsed=ppulsed;
dkcarryff=pdkcarryff;
dkrdat1=pdkrdat1;
*dkrdatf=pdkrdatf;
writeld=pwriteld;
writeld1=pwriteld1;
writecnt0=pwritecnt0;
writecnt1=pwritecnt1;
timeout0= ptimeout0;
sepdisfd= psepdisfd;
*dkrckf= pdkrckf;
*longbit=plongbit;
*sum= add0 + (add1 << 4) + (add2 << 8) + (add3 << 12)
     + ((long int)(add4) << 16) + ((long int)(add5) << 20);
*freq= freq0 + (freq1 << 4) + (freq2 << 8);
writecnt= writecnt0 + (writecnt1 << 4);
writecnt=writecnt; /* to get rid of "unused" warning message */
/* writecnt is only used in print statement below, which is commented out*/
/*
printf("|%x%x %x%x%x%x %x%x%x %4x %6lx %x%x%x %2x|\n",pulse,early
    ,mph,pph,dkdn,dkup
    ,c,c1,c2,*freq,*sum,*dkrckf,*dkrdatf,*longbiten,writecnt);
*/
}
```

}

9.1.13 Disk Test Register. -

A new disk register has been defined to allow better testibility of and permit more visibility into the disk circuits. The following register description defines the bits within this register.

DSKTST 0x5A4

```
15  x
14  x
13  x
12  dsktstrd  (read tst info from dskbytr register) bits 15-9 switch
          15   dkstate 3
          14   dkstate 2    these 4 are state of disk state machine
          13   dkstate 1
          12   dkstate 0
          11   lenfin
          10   gapend
          09   sectend
11  secttstcnt -- count sector register once when writing register
                  with this bit high
10  x
 9  x
 8  x
 7  gaptstld   -- load gap counter from gap register
 6  gaptstcnt  -- count gap register once when this bit is written
 5  gapfc2     -- force a carry into bit 8 of the gap cntr so
                  that the higher bits count faster than normal
 4  gapfc1     -- force a carry into bit 4 of the gap cntr
 3  lentstcnt  -- count the length counter once
 2  lenfc3     -- force carry into bit 12 of length counter
 1  lenfc2     -- force carry into bit  8 of length counter
 0  lenfc1     -- force carry into bit  4 of length counter
```

All bits in DSKTST are reset on chip reset or at normal startup of the disk controller. The following paragraphs discuss some of the disk circuits, and how DSKTST can be used to verify proper operation.

To test the length counter it is necessary to get the disk controller out of the idle state. This is because the length counter is loaded continuously in state 0 (idle), and load overrides count. Once out of the idle state, the length counter can be loaded with a write to dsklen (lower 14 bits) or dsklenn(all 16 bits). Be sure to keep the dma on so as to keep the controller out of the idle state.

The other 2 counters can be tested, if desired, without starting up the controller at all. The sector counter can be loaded by DSKGAP, counted with the secttstcnt bit, and tested for zero using the DSKBYTR register in conjunction with the dsktstrd bit. The gap register can be loaded by DSKGAP, transfered into the upper 8 bits of the 12 bit gap counter with gaptstld (lower 4 bits are loaded to zero), counted with gattstcnt, and tested for zero with DSKBYTR.

The gapfc and lenfc bits have been provided to allow the gap and length counters to be fully tested in a relatively short period of time. Setting one of these bits forces a carry into the specified

position of the counter thus permitting the counter bits above that position to be exercised with every increment cycle. To test the normal carry path thru the counters, it is necessary to turn off the gapfc and lenfc control bits. For example, to check the normal carry path between bits 7 and 8 of the gap counter, turn off the gapfc2 bit and set the counter to 0x100.

9.1.14 Disk Hardware Description. -

This section describes each functional unit of the disk controller. The description relates functionality to schematics and C-code functional models. The reader should refer to the schematics, functional models, and state diagram as needed.

9.1.14.1 Clocks. -

(PLLCK; pllck.c)

The data separator and precomp sections use the clock 'CPLL'. The others use C14, the internal version of BUSCLK. The async interfaces provide a connection between the differently clocked sections.

This section generates CPLL. This clock signal has two speeds, 28M/4 and 28M, controlled by software. The switching between the two occurs when the clock is high. The only place the 28M clock is used is in generation of CPLL. This and the async interfaces allow the "28M" clock to vary independently of the BUSCLK.

9.1.14.2 Async Interfaces. -

(PLLCK; part dskio.c)

All interfaces between the C14 section and the CPLL section are designed to work asynchronously. The key translation from CPLL->C14 is the clock and data out of the data separator (dkrckf,dkrdatf) to the read circuitry (dkrck,dkrdat). The key translation from C14->CPLL is the clock and data from the output of the write section (dkwck,dkwdat) to the precomp section (dkwckfd,dkrdatfd). Several other signals are also translated, with the main goal that the part operate consistently under test with process variation.

A non return to zero (NRZ) technique is used for the key translations. This allows a new clock tick to be picked up at the receiving end every tick, and the same method works from a fast clock to a slow clk or a slow clock to a fast clock.

When translating the write section data from C14 to CPLL at high speeds, the time discreteness of C14 relative to CPLL is significant. A 2 bit fifo for the data was inserted (dkwckf,dkwdatf-> dkrckf,dkwdatfd) to even this out and maintain the 1/2 tick CPLL resolution for the output.

9.1.14.3 Data Separator. -

(DSKPLL; dskpll.c, part of dskio.c)

This section takes the input data stream and figures out the bit width and clocking points. It uses phase locked loop techniques to lock into the frequency of the data stream. The output is a clock at the end of each bit width (or window), which clocks in the data for that window (1 if there is a bit in the window, 0 if not).

The max and min frequency and the feedback for the phase locked loop (PLL) is variable with software.

Also in this section is a counter which defines the bit window time for write.

Both the read and write sections have resolution of 1/2 tick of CPLL. The main clock,data outputs are dkrckf,dkrdatf. These are used in both read and write modes.

9.1.14.4 General PLL Method. -

An adder adds typically a certain number (the frequency) every CPLL tick. The upper bits of the adder tell what fraction of the total adder cycle it is currently at. When a pulse comes in, the number added is made smaller (phasem) or larger (phasep) for a number of cycles depending on the difference from the center of the adder cycle (c2,c1,c0). If the difference from center is larger, a larger correction is made. The goal is to get the following pulses in the center of the adder cycle. In addition to these immediate corrections, a small correction (dkup,dkdn) can be made to the typical number (the frequency).

9.1.14.5 Precomp. -

(PRECOMP; precomp.c)

On a disk, if 2 '1' bits are written close together with a wide space on each side, the bits will tend to spread out when read back. This section compensates for that to some degree by pushing bits in this condition a bit closer together when written.

There are 4 different values of precomp (amount of push in). The values are 0,1,2 or 4 CPLL ticks. There are 3 different types of precomp. The 3 types depend on the number of bit windows between the closest spaced '1's. For GCR and Biphase-Mark, 1s can be in adjacent bit windows. For MFM, 1s at least 2 bit windows apart, and for RLL(2,7), they are at least 3 bit windows apart.

9.1.14.6 Read Circuitry. -

(DSKDP1,DSKCTL1; dskread.c)

The section takes the clock and raw data from the data separator and converts it to 32 bit wide data ready to go into memory. On the way, it can byte/word align the data using sync marks in the data

stream. It can also convert the raw bits into data for MFM, Biphase-Mark or RLL(2,7) codes.

The sync mark can be 32,16, or 8 bits long, and it can be included in the data stream or not. Also, the data can be sent to memory either LSB first (CD mode) or MSB first (other).

9.1.14.7 Write Circuitry. -

(DSKDP1,DSKCTL1; dskwrt.c)

The reverse of the read process occurs here. A 32 or 16 bit chunk of data is converted to a clock and one bit data to be sent to the precomp section.

The data chunks above come either from memory or the CRC shift register. The memory data can be sent out as raw bits or encoded data, and the output of the CRC shift register can be sent out either raw or encoded. The switch between these types can occur several times during a disk transfer, sometimes in the middle of a word. The type of the data (0-raw,1-encode,2-CRC encode, 3-sync(CRC raw)) comes through the fifo along with the data. For type 3 the CRC shift register is loaded from the sync register, then shifted out.

The data can be sent out either LSB first or MSB first.

In sector mode, data from memory is matched against data from the disk looking for a particular sector header. In these conditions, the write shift register is used to shift out the data from memory. The data out of the write shift register is compared with the data out of the read shift register. When the two continue to match until the end of the sector header, the header is deemed found.

9.1.14.8 Read/Write Shared Circuitry. -

(DSKDP1,DSKDP2; dskwrt.c,dskread.c,dskctl.c)

The 2*(32+3) bit fifo provides a buffer between the memory and the disk controller, allowing more dma delay and delay variation. The extra 3 bits transfer the data type (raw,encode,CRC,sync,16/32 bits).

The CRC register is used for generating and checking CRC, an error checking number. It also is used for shifting out sync marks in CD and trackplus modes.

The length registers/ counter count the number of 2 byte increments in a dma transfer. The 3 registers contain the address header length, data header length, and data length, respectively. They are loaded into the counter at the appropriate time by the control circuitry.

The gap register/counter counts the number of raw bit times from the end of a sector address header to the beginning of the sector data. On read it is used as a timeout in case the data header is not found, and on write it is used to time the start of writing sector data. It is only used in sector mode.

The sector counter does exactly that. It is used in sector, CD, and trackplus modes.

9.1.14.9 Control Circuitry. -

(DSKCTL2,3,4,5 -> DSKCTLB; dskctl.c)

This has a 4 bit state machine used to manipulate many controller operations. See the state diagram, dkstate.doc, and the regbits document for detail on the different modes.

It also has other bits of circuitry to generate dma and interrupt requests, detect errors, control the various counters, etc.

9.1.14.10 State Machine Description. -

Startup occurs when either the DSKLEN or DSKLENN register is written while the dma is on. In modes 1 and writing in mode 3, it goes to state 1, otherwise going to state 2, and skipping the action in states 1,9,B,3. Note that one of the dma enable bits is in the DSKLEN register. Because of delays in this enable bit, if the DSKLEN register is written to zero after an operation is finished, it must be written twice to start up, as the first time the dma enable bit in the register is not yet high.

States 1,9,B and 3 are states used generally for matching the data from memory with data from the disk, used to find address headers in mode 1. These states are also used to write the header in mode 3. Address header finding is accomplished by first filling up the fifo with data from memory, then waiting for a sync mark. After the sync mark, the data from memory is compared with data from the disk. A failure puts the machine back to state 1, where it waits for any pending requests to clear, then fills up the fifo again from the beginning of the address header.

States 2 and 6 are used rather differently in different modes. In modes (0,2,or 3 AND read) a sync mark is waited for and then it skips out of this section. If syncen is off, these states are just skipped through. In model AND write, state2 is a wait for the end of the gap between address header and data header, and state 6 writes the data header. In mode 1 and read, states 2 and 6 are where data header is looked for. If not found within a specified time, state E is entered, indicating a 'notfound' failure.

State 4 is where the main body of data is transfered.

State C is usually a CRC calculation/generation place. In mode 2 read, there is also a sync mark wait before transitioning into state 4 to start the next sector. There is also a wait so things are cleaned up before going into state D. As an example, in mode 1 write, there is a wait for the write enable to be false.

State D is a 'wait for readiness for dmarequest' state. Then it transitions into state 9 where the next sector header is looked for.

State 8 is a 'wait for everything to be done' state. A disk block interrupt is generated on the transition from state 8-> state 0.

State F is a 'wait for all loose ends to settle out' state. This is the failure stop state, and waits for things such as pending DMA requests to be satisfied before the transition to state 0. This way an immediate startup from state 0 should be clean. It generates a disk block interrupt on the transition from state F to state 0.

9.1.14.11 Signals And Meaning. -

dskdmaon - all the various disk dmaon and enable bits are high in DMACON or DMACONX and DSKLEN,N
dsklen - generated on writing of DSKLEN or DSKLENX registers
mode 0,1,2,3 - bits 3,2 in ADKCONX
fifordy - It is safe to send out a dmareq to fill/empty the fifo
fifordy1 - fifordy on write
fifordy2 - fifordy on read
reqpend - there are dma requests out, but not satified
startmatch - fifo filled up and sync mark occurance, generally
lenfin - length counter is zero- It is a down counter.
match - the data from the memory and the data from the disk match at this instant.
endmatch - last bit of match is over.
dskw, !dskw - write bit in DSKLEN,N register.
gapend - gap counter is zero. (It is a down counter)
sectend - sector counter is zero.(It is a down counter)
dkweslow - a inverted and delayed copy of the DKWE_ pin, high when write is enabled to the disk.
crcdone - crc calcs/generation finished.
wcnt=0 - the write bit counter is in the idle state.
wdlden - enable for the write bit counter to leave idle state
wdlden2 - enable for the write bit counter to leave idle state
disk overrun - when the overrun bit of the INTREQX register goes from low to high- can be caused by write or read overrun, or by a software write of the register.

9.2 Extended Audio Capabilities.

The new chip set is capable of much higher quality sound generation than older versions of the chip set. Maximum sampling frequency has been increased from ~31Khz to ~64Khz. When less than 12 bit volume precision can be tolerated (six bits), maximum sampling frequency peaks at 110Khz (turbo mode). On top of this, the extended audio software model supports up to 16 bits of audio data per channel and four new channels have been added to provide a total of 8. The on board DAC contains 16 bits of audio resolution. Digital left, and digital right audio is brought out of the chip on serial data ports. With appropriate external hardware in place, 20 bit audio data can be shifted to external 20 bit DACs.

Old audio RGA registers have been left intact to retain software compatibility. All new modes simply extend old functionality; the old software model still applies to the audio circuits. (See the original Amiga Hardware Manual for an excellent description of the audio hardware software model. The extended audio registers which invoke new audio modes (see Appendix F) are straight forward extensions to this software model.)

The following table summarizes improvements made to the audio circuits.

	old	new
sample rate	29khz	64khz (110Khz in turbo mode)
channels	4	8
volume bits	6	12 (signed)
volume aliasing	yes	no (done by hardware multiply)
sample size	8 bits	8 or 16 bits
digital out	no	yes
dynamic range	15 bits*	16 bits* (@6db/bit -> 96db)
channels on left	2 (0,3)	0-8 (select any or all)
channels on right	2 (1,2)	0-8 (select any or all)
global mono bit	no	yes
output time reso	280ns	280ns
period resolution	280ns	280/64 ns (4.38ns)**

*Dynamic range-- The old chip set had 8 bits of data, 6 bits of volume, with 1 D->A converter/channel. With 2 channels on one pin, this gives 8+6+1=15 bits of dynamic range. The volume was done with pulse modulation techniques which caused aliasing on periods not multiples of 64 CCK cycles, but it did give a very wide dynamic range. The new chip set avoids the aliasing and retains the same dynamic range by doing digital 18 bit calculations for volume and channel summing, then outputting the final result with 16 bit resolution.

**Period/Output time resolution-- The new chip allows calculation of the output time of a given sample to 4.38 ns, but actually puts the output of any given sample on a 280ns boundary. This will cause some distortion, but it will be very small if used carefully. Compare this to a CD, which puts samples on 22,676ns boundaries.

9.2.1 Audio Hardware Operation. -

In the original Paula chip, each channel was a separate hardware entity, each channel had its own length register, length counter, period register, and period counter. In the new system, there is only one set of calculation logic for all channels and for all logic calculations. The calculations are sequenced through 4 calculations per channel, and all 8 channels are calculated in sequence. This sequential calculation spends less silicon for the same functionallity.

9.2.1.1 Algorithm Used By The New Audio Processor. -

Each channel has 4 "calculation periods" -- length (L), period (P), data buffer (B) and accumulator (A). As there are 8 channels total, the total calculation cycle is 32 ticks, a tick being one 14m clock period (one 70ns BUSCLK period).

L-- calculate new value for length counter (load or count or same)
P-- calculate new value for period counter (load,count,same)
and calculate new audio states and control outputs
B-- load the data buffer and volume buffer registers from the volume (on appropriate state and control information)
A-- calculate data*volume/32768. calculate states for multiply machine

The calculation information comes from 4 places:

1) parameter ram -- set by processor or data dma (eg. period register)
2) variable ram -- set by logic (eg. period counter)
3) fixed control -- set by processor (eg. attach volume control bit)
3) variable control -- set by logic (eg. audio state)

It takes 4 ticks to do one channels calculations:
1) read parameter ram, variable ram, fixed control, variable control(Read)
2) reclock and calculate first set of logic (CalcA)
3) reclock and calculate second set of logic (CalcB)
4) store results back in variable ram and variable control (Sto)

For example of the A calculations (multiply):
1) read parameter ram (volume and data buffered, and accumulator) and variable control (mpy state) and fixed control (volume format)
2) reclock, propagate logic for deciding inputs to adder-- decide control for audio data path logic (andu,andl,sh1,2,3,m1-6,sub...)
3) do adding as described by above control bits
4) store result-- accumulator with new value, mpy state with new value.

The 4 (L,P,B,A) calculations are pipelined, starting one tick behind the other:

ram read address	L	P	B	A	Time slot
0	Read0	--	--	--	A
1	CalcA0	Read0	--	--	L
2	CalcB0	CalcA0	Read0	--	P
3	Sto0	CalcB0	CalcA0	Read0	B
4	Read1	Sto0	CalcB0	CalcA0	A
5	CalcA1	Read1	Sto0	CalcB0	L
6	CalcB1	CalcA1	Read1	Sto0	P
7	Sto0	CalcB1	CalcA1	Read1	B
8	Read2	Sto1	CalcB1	CalcA1	A
...	...	...	...	...	...
31	Sto7	CalcB7	CalcA7	Read7	B
0	Read0	Sto7	CalcB7	CalcA7	A
1	CalcA0	Read0	Sto7	CalcB7	L
2	CalcB0	CalcA0	Read0	Sto7	P
3	Sto0	CalcB0	CalcA0	Read0	B
...					

-- stuff from channel 7 of previous cycle

Careful note is taken of how these cycles interact with each other so that when things cross boundaries, they work ok. One such issue is when length finished occurs and it needs to affect the audio state calcs. It is delayed one tick so that it shows up at the right time slot for the calculations. The opposite example is when the audio state calculations need to affect the length counter. In this case, the state calculations (or at least the effects on the length counter) need to be stored one tick early so that they come out at the right spot for the length calculation on the next loop around. In the case of stuff like this, control information is stored one tick early. This means doing a store after the CalcA phase, which in turn implies that the control time previous to this calculation must not store control information. All the calcs are actually done during the P and A times, with both early and regular storage of control bits. This gives 16 bits of control storage for these times, with none for the L and B times. The L and B times can, however, use the values that are "early stored" by the P and A times, respectively. The P and A calcs that need "early stored" values must delay them an extra tick.

Control logic flow:

read addr	length	period	buffer	accum	time
0	read0				A7
1	use	read0			L0
2	v------	CalcA0	read0		P0
3	StoEarly0	delay	use	read0	B0
4	read1	StoNorm0	v------	CalcA0	A0
5	nop	read1	StoEarly0	delay	L1
6		CalcA1		StoNorm0	P1

9.2.1.2 Main Calculation Loop. -

The logic in the main timing loop is basically as follows. When the period counter times out, it is reloaded, a new sample starts, and the buffer register is loaded. This also triggers the multiplication sequence, where the data in the buffer register is multiplied by the volume and the most sig 18 bits kept. The multiply is done sequentially, 2 bits at a time. A DMA request (auddr) occurs every other sample, along with counting of the length counter. When the length counter times out, a DMA restart request (auddsr) is issued and the length counter is reloaded. The main calculation loop takes 32 ticks (2.2us). Timing resolution of 280ns for a sample is needed for compatibility with the old Amiga system.

9.2.1.3 Length Finished. - The decision to count the length counter occurs at 'P' time. It is transmitted to the length counter by an 'early store' (see above), but the length calculation occurs on the next loop, and the result 2 ticks later. This means that the result isn't available at 'P' time until 2 loops after the original request was made. This causes unpleasantness, because the data write usually triggers the length counter, and the result of the new length calculation being 2 loops later would require (on startup, with even numbered lengths with 32 bit transfers) that the dma transfers wait at least 4.5us (2 loops) to happen. It also means that in normal operation, that the data come at least 2 loop times before the period times out. This would subtract 4 loop times (9us) from the latency, which at maximum rate is 6 loop times, leaving only 2 loop times (4.5us). The fancy lenfin calc basically looks at all the inputs to the length counter and figures out from that whether it will time out or not, effectively trimming the calculation time down to 1 loop. This means the minimum dma response is one loop (startup condition) and the lastest time in normal operation is 1 loop from the end, leaving 4 loops (9.5us) for other stuff.

9.2.1.4 Retiming Loop. -

To get the 280ns resolution, a retiming circuit is used. The period counter 3 LSBs are loaded into the retiming circuit along with the data to be output. Any channels timing out (multiplication finishing) in a 2.2us window are output from the retiming circuit in the NEXT 2.2us window, with the exact point in the window determined by the period counter LSBs.

There are 2 retiming circuits, each operating on 4 channels sequentially. Each channel takes 70ns, for a total of 280ns, maintaining the required resolution.

The lower three bits of the period are saved until the next period timeout, so that the retiming section gets the 'remainder' that was in the period in the LAST sample, not the current one. The main loop gives period timeout to 2.2us (8 CCK) increments, and sends the remainder to the retiming loop. For a period of 33 ccks, the main loop will timeout in 4 loops for 7 samples, and then 5 loops for the 8th sample. The last sample it times out in 4 loops, it must send out the maximum delay to the retiming circuit (7) and when it times out in 5 loops, it must send the minimum (0), in order to keep the actual output evenly spaced. This requirement means that the 3 LSBs of period from the previous sample time must be used to send to the retiming section.

9.2.1.5 Holding Ram. -

Following the retiming circuits are two holding registers, each doing 4 channels, again circulating at 280ns. These holding registers feed the output summing network, with control bits determining whether the channels go to the left or right side, or both.

9.2.1.6 Scaling And Clipping. -

Results are carried at 18 bits/channel up to the point they are to be summed together for application to the output DAC's. The multiply algorithm can only generate a half full scale number, therefore audio data at this point in the chip can be viewed as 17 bit data. When 8 channels are added together, a 20 bit result is obtained.

The final stage before the D->A is a dividing/clipping network. For compatibility with the previous Amiga, two channels on the left (or right) should make a full scale signal. On the other hand, in the new system there may be as many as 8 channels on the left side, and they must be settable so as not to clip. The dividing network divides the signal by 1,2,or 4, and the output is then clipped so that if it was divided by 4 it will not quite clip with 8 channels full scale, and if not divided it will not quite clip with 2 channels full scale.

9.2.1.7 Analog Output. -

16 of the 20 bits output from the scaling and clipping circuits are input to the D->A converters, one for the left side, and one for the right (AUDL, AUDR, AUDZL,AUDZR). The current out (iAUDL + iAUDZL for example) an output pair sums to the same value always. The AUD and AUDZ pins for each side should be summing nodes a tad above ground so that there is no leakage current and so that the N channel switches are really off, unless they can be REALLY (1 lsb is .112ua) off. In all cases, some of the initial 16 bit accuracy is lost with one channel on the analog (16 bit) output:

clip with	2 channels	--lose 1 bit
	4 channels	2 bits
	8 channels	3 bits

9.2.1.8 Digital Output. -

Twenty (20) bits are run out digital audio output pins, in a serial fashion. There are 2 serial outputs (AUDDIGL and AUDDIGR) with a strobe (AUDDIGLD) to tell when a data word is finished. To receive the data, use a shift register clocked on the rising edge of 14m BUSCLK, and a synchronous holding register clocked on BUSCLK and load enabled by the AUDDIGLD signal. Bit 0 comes out first, bit 19 last, so that the external circuit can pick up as many bits as it wants. The left and right sides come out with the same timing.

All of the accuracy of the calculations are available on the digital output. Each 16 bit channel is data*volume, with a 17 bit effective result, summing up to a 20 bit number (8 channels). Although this output is scaled and clipped the same as the analog output, the extra 4 bits maintain full accuracy at all settings, as long as clipping does not occur.

9.2.2 Compatibility. -

A given audio channel can be run either in 'old' (compatibility) mode or "new" mode. A channel goes into oldmode on power up, and whenever the old DMACON bit is on or when a write is done to the 16bit AUDXDAT register for that channel. Newmode happens when the new DMACONX dma bit is used or the new AUDXDATX data register is used. The channel will remain in the mode it was set into until one of the above actions occurs. Old Mode actions win over newmode ones if there is a fight. The idea is that any old programs will run regardless of what mode a previous newmode program left the new control bits in.

In oldmode, attach period and volume come from the ADKCON register (ch 0-3) and from the control register (ch 4-7). The old volume format is used, and 8 bit data format is used. Channels 0,3 are forced to the left side, Channels 1,2 to the right. Channels 4-7 cannot be operated in oldmode.

In newmode, the AUDXCTL registers for all channels have full effect, and the ADKCON attach bits and previous fixed channel left/right allocations are ignored. Default is new volume format, 8 bit data, no attach, both left and right bits off.

There are two known incompatibilities with the old system. 1) A '0' period means '65536' in the old system, and '262144' in the new one. This is a 54.5 Hz sample rate, pretty uncommon. Other periods work the same. 2) The resolution of the writing of the registers for attach period and attach volume has 2.2us resolution in the new system, vs 280ns in the old system. As the writing of these registers has no effect on the channel sending out the data until the beginning of the next sample, as long as the attach writes occur sometime during the desired sample, the exact same results will occur. If the attach write happens near a sample edge, the volume/period change may occur one sample before or after the original desired one.

Some incompatibilities may occur due to the slower state machine tick time (2240ns vs 280ns). Dmaon has been delayed to cover one case of this.

9.2.3 Stopping The Audio Channels. - The original Agnus does not send out the first word of audio data as a result of the first audio DMA request. It sends out the first data the NEXT time a request is made. As a result of this, on dma startup, the audio must throw away the first piece of data it receives, and use the 2nd data as the first useful one. This also means that the sample being played is often several samples behind the one being requested, and that stopping is sometimes a bit awkward. The audstop bit helps this out. The positioning of the DMA request also varies with the data and sample type, which further complicates matters, especially since a full 32 bit buffer was not used. The logic for the intreq was made so that it comes out as the sample 1 word from the end of the buffer starts being played. (on 8 bit samples, 2nd to last sample, on 16 bit ones, the last sample). This interrupt positioning is independent of the data transfer type (32/16 bits), and whether there is an odd or even number of two byte chunks in the buffer with 32 bit transfers.

Stopping audio DMA was a mess with the old system, because of the Agnus pipeline, and also the fact that when the intreq at the end on a buffer occurs, a dma request occurs at the same time, requesting the first two samples of the next buffer. At this time, the 2nd to last sample of the current buffer is starting (going to output). If software responds to the intreq by turning off the dma, the audio state machine stops, but the samples from Agnus are still floating around somewhere, and there is no message to know that they have come out. As long as the interrupt was not reset, the data will ping harmlessly on the data register and never get to the output. A two horizontal line wait should be safe for this, maybe even one. Another method is shown below.

If the software resets the interrupt when the request comes out at this point, and then waits, the two samples will go into the audio output queue. Assuming that the dma was also turned off when the intreq came, another intreq will occur as the first of these samples starts playing. If the software responds by setting the period to 1 and resetting the interrupt, the audio will finish playing the first of these samples, then play the 2nd,1st(intreq),and 2nd very quickly, then stop. Once the software responds to this interrupt, the audio will be in the idle state with no data floating around anywhere. This assumes that the audio does the last 3 steps very much more quickly than the software can respond. In the old system, this happens in two CCKs after the intreq (0.56us).

End of Buffer (old system):

Software	Hardware
init: int clred	
	send intreq, dmareq(for 1st 2 samp- a,b)
dmaoff,clear intreq	Agnus sends data <- 2 things in parallel
	play data, send intreq
clear intreq,per=1	
	play data 'a' normal speed
	play data 'b' one tick
	intreq, play data a one tick
	play data b one tick
	-> idle state
observe intreq	

In the new system, this takes two loop times after the intreq (4.4us). To help compatiblility, the new system has a delay on turnon when the dma is set so that if the software responds immediately (<2.2us) to the

intreq and turns the dma back on, the audio state machine will be in the idle state before it detects the turnon of the dma, and will turn on as it would have before. Also, for periods < 8, the new system will not output any new data.

If the dma is turned off in the middle of a buffer, the situation is a little different, but similiar. If the interrupt was cleared before, the audio will send out an intreq as it begins to play the last samples it received, after the pending samples were received from Agnus. If the intreq is responded to by clearing it and setting the period to 1, the current sample will finish normally, followed by the 2nd one quickly and making an intreq, the 1st one again quickly, the 2nd one quickly, then back to the idle state because the intreq(send as it did the 2nd one quickly) was not reset. Again, the time from that final intreq and the time the audio is in the idle state is two state transition times (.56us in old system, 4.4us in new).

Middle of buffer (old system):

Software	Hardware
init state: int clred	
	(Agnus sends a and b here just before off)
turn dma off	
	(or Agnus sends samples a and b here)
	send intreq, start sample a
reset intreq	
period=1	
	finish sample a (normal speed)
	do sample b in 1 tick, send intreq
	do sample a in 1 tick
	do sample b in 1 tick
	-> idle state
notice intreq	
turn dma on or whatever	

See! It is a mess, either at the end or the middle of the buffer.

End of Buffer (New system):

Software	Hardware
	previous buffer intreq
set audstop	
reset intrpt	
	beginning last sample send intreq(set int)
	end last sample
	go to idle state, send overrun intreq
observe overrun	

middle of buffer (new system):

init: int reset	
turn off dma	
	finish current sample (normal rate)
	(may pick up new data from Agnus in this time)
	send intreq (set intrpt)
	play data (normal rate)
	send overrun req and enter idle state
observe overrun	

9.2.4 Audio Period Fine Mode. -

The new mode audio circuits provide a means of specifing sample periods to a resolution of 4.38 nsec. This mode has been implemented for two reasons. 1) To permit the generation of accurate, low frequency tones from relatively short sample lists (compared to the size of the list required using the old mode audio device when certain specific tones are desired, whose fundamental frequency is not an even multiple of the base period frequency of 279.3651 nsec). 2) To permit the playback of audio data samples at frequencies very close to industry standard sampling frequencies (see section 9.3.3, Standard Sampling Rates).

This mode does not permit sample rates faster than 65Khz (or 110Khz in turbo mode). Maximum sampling frequency is limited by the amount of time the hardware takes to multiply each audio sample by the specified volume.

When period fine mode is enabled, hardware divides the value found in the period register by 64. The remainder of this division is treated as a fractional period which is accumulated from sample to sample. As in old mode operation a copy of the period is decremented every 279.3651 nsec. Each time the integral part of this working period register times out for the sample being played, the fractional period remainder is added to the original period which was programmed to calculate the period which will be used for the next sample to be played. Therefore, depending on the value programmed into the period register, every so often one data sample will be played 279.3651 nsec longer than the other data samples are played. For example, suppose it is desired to play four samples, each with a period of 3.4. The following table shows the value the hardware working period register would contain at any given clock cycle, and the number of clock cycles which each sample is played.

clock cycle	working period register	comment	sample being played
0 start	3.4		sample 1
1	2.4		sample 1
2	1.4	timeout after 3	sample 1
3	3.8	3.4 + 0.4 = 3.8	sample 2
4	2.8		sample 2
5	1.8	timeout after 3	sample 2
6	3.C	3.4 + 0.8 = 3.C	sample 3
7	2.C		sample 3
8	1.C	timeout after 3	sample 3
9	4.0	3.4 + 0.C = 4.0	sample 4
10	3.0		sample 4
11	2.0		sample 4
12	1.0	timeout after 4	sample 4
13	3.4	3.4 + 0.0 = 3.4	sample 1
14	2.4		sample 1
15	1.4	timeout after 3	sample 1
16	3.8	3.4 + 0.4 = 3.8	sample 2
17	2.8		sample 2
18	1.8	timeout after 3	sample 2
19	3.C	3.4 + 0.8 = 3.C	sample 3
20	2.C		sample 3
21	1.C	timeout after 3	sample 3
22	4.0	3.4 + 0.C = 4.0	sample 4
23	3.0		sample 4

24	2.0		sample 4
25	1.0	timeout after 4	sample 4
26	3.4	3.4 + 0.0 = 3.4	sample 1
27	2.4		sample 1
28	1.4	timeout after 3	sample 1
29	3.8	3.4 + 0.4 = 3.8	sample 2

etc

Note that in this example, the samples are played the same way every iteration thru the list. In this case, the last sample is played for one clock cycle longer than samples 1, 2, and 3. If the period programmed had been 3.8, every other sample would have one cycle longer. If the period programmed had been 3.2, every 8th sample would have one cycle longer. Each time the fractional part sums to 1, the next sample is played for one cycle longer than the others.

This algorithm induces distortion into the resulting audio signal. However, when this mode is used carefully, the distortion can be minimized. To minimize the distortion, the pattern of short to long samples should be set to repeat at least as often as the waveform being played is repeated. This means that the fractional part should be the same at the first sample of the waveform every time it is played. If a waveform is N samples, the period should be set so that (periodregister * N) mod 64 is 0. Another case where the distortion is very small is when the pattern repeats over a small number of cycles of the waveform--((periodreg * N * M) mod 64)==0, where M is a relatively low number. Even without these considerations, the distortion is still pretty small. Suppose the audio circuits are programmed to play at a 71.40626khz sampling rate to make a 5.4928 khz complicated waveform. (There would be 13 samples, and the period register would be set to 3201.) The pattern repeats every 13 x 64 = 832 samples, 64 times through the waveform. Every 64 samples, one is 280ns longer. The short cycles would be 50 x 279.3651 nsec long, and the long sample would be 51 x 279.3651 nsec long. There are several waveforms in the 64 that have one sample out of 13 2% longer than the rest, so the distortion in those waveforms is around 2%/13, or 0.15%. These are roughly every 5 waveforms. The shorter ones are about .15%/5 too short, and the long one is 0.15 x (4/5) too long, so the average frequency distortion is around (5x(0.15/5)+(4/5)x0.15)/5 = .054%. These calculations are very crude, but give an estimate of the expected distortion.

9.2.5 Test Mode Registers. -

Five registers are allocated for test mode: AUDTST0W,R,AUDTST1,W,R, AUDOUTR. The first 4 registers allow internal setting/reading of stuff in the main audio calculation loop directly from the processor. The last allows direct reading of the values going to the D->A converters.

9.3 Pots.

The AAA pot circuits have undergone a redesign to provide consistent operation across the wide range of monitors the AAA chip set can support. The old pot circuits operated by incrementing once each physical horizontal scan line until the pot voltage reached a predefined threshold. This meant the pot could achieve a maximum count

of 255 in NTSC systems (263 minus an eight line discharge interval). With the new display resolutions now possible, the same counter could theorically reach a maximum count of 1076 during the same vertical scan time of 16ms. But since the circuit still incorporates an 8 bit counter, it would roll over up to 4 times before the threshold was reached and the terminal count latched. To resolve this problem (or circumvent the requirement that a different pot resistance be used for each system using a different monitor resolution), a 'horizontal' line counter has been built into MARY's pot circuits to replace physical horizontal line sync strobes and control pot counter increment with a constant 63usec clock. This counter always times 'NTSC' lines regardless of the type monitor connected to the system. By implementing this self-contained 'NTSC' counter (always increments the pot counters once every 910 14.3Mhz BUSCLK periods), the pots provide consistent results on any resolution monitor which uses a 60hz frame rate.

9.3.1 Audio Sampling. -

Another modification has been made to the pot circuits. They can now be programmed to count at a BUSCLK rate (14.3Mhz) as well as at the traditional 16Khz rate. When this new mode is invoked, the pot period is established by audio channel three's period instead of the traditional monitor vertical sync period. This new higher speed mode of operation makes it possible to use the pot circuits as an audio sampling circuit.

When audio sampling mode is invoked, audio channel 3 can no longer be used in the normal output mode. The channel is sacrificed to support audio sampling. The value programmed into this channels period register sets the pot circuit audio sampling rate. Audio channel 3's DMA cycles are used to write the sampled pot data into memory.

Two audio sampling modes are possible (Note: Mary R0 does not incorporate all the necessary logic required to support these modes as defined. The following sections describe audio sampling as it will operate with the R1 version of MARY. Please consult with chip circuit designers for modes available in the R0 version of MARY.)

9.3.1.1 Internal Circuit Audio Sampling Mode. -

Some of MARY's POTMOUSE inputs and circuits are used to provide 8 bit stereo sampling. The sampling rate is set by programming channel 3's period. The analog audio waveform to be sampled must be an AC current, varying with respect to the amplitude of the signals to be sampled. The currents are input to the chip through the POTRX,POTRY pins. The following list shows how the chips are programmed to operate in this mode.

POTMOUSE Circuits Audio Sampling Mode

CPU: -write AUD3PER sampling rate
-write AUD3VOL 0x0000 (silence)
-write AUD3CNTL 16bit samples
-write AUD3LEN sample length
-write AUD3LC(X) starting address of sample.
-write to POTGO (enable sampling mode)
-write DMACONX 16 bit dma for aud3

```
MARY: -decode POTGO; set internal audio sampling mode.
ANDREA: -decode POTGO; set audio sampling mode. (DMA write
        to memory)
MARY: -aud3 period times out.;++aud3len
      -AUD3_RESET signals when data moves from retiming
       ram to holding ram.
      -Latch potbuf value into the sample-buffer latch;
      -reset the pot counter and the potbuf latches.
      -dmal audio channel 3 bit set.
ANDREA: -respond to dmal;
        -put AUD3DAT on bus, prepare for DMA memory write
         rather than normal AUD3DAT memory read.
MARY: -decode AUD3DAT and set POT1DAT decode. The two
       8-bit samples in POT1DAT will be put on the AD
       bus.
ANDREA: -complete memory write.
```

9.3.1.2 External Circuit Audio Sampling Mode. -

The chips can be programmed to disable internal analog to digital convertor circuits and instead drive external analog to digital convertor circuits to provide 16 bit (or 8 bit) stereo (or mono) sampling. One of MARY's output pins provides a clock period (established by audio channel 3) to the external circuits as an indicator of when to begin a conversion cycle. Another of MARY's output pins, SAMPRGA*, provides an enable signal to the external circuits to define the time when the external circuitry should drive sample value(s) (8,8) or (16,16) bits onto the AD bus. This occurs when an audio channel 3 DMA request is serviced by ANDREA. The SAMPRGA* pin has been designed so that it may connected directly to the enable input of a (for example) 74LS244 buffer. When external circuit audio sampling mode is enabled, none of the POTMOUSE logic is used.

It be noted that data DMA'ed from the external logic need not be audio data. The mode described in this section could be used to DMA a variety of data into chip memory. This mode actually provides a general purpose DMA input channel.

The following list shows how the chips are programmed to operate in this mode.

External Circuits Audio Sampling Mode

```
CPU: -write AUD3PER sampling rate
     -write AUD3VOL 0x0000 (silence)
     -write AUD3CNTL 16bit samples
     -write AUD3LEN sample length
     -write AUD3LC starting address of sample
     -write to POTGO (enable external sampling mode)
     -write DMACONX 16 or 32 bit dma for aud3
MARY: -decode POTGO; set external sampling mode
ANDREA: -decode POTGO; set audio sampling mode
MARY: -aud3 period times out.;++aud3len
      -AUD3_RESET signals when data moves from retiming
       ram to holding ram sets SAMPPER pad.
      -dmal audio channel 3 bit set
EXTERNAL: -detect sampling strobe on SAMPPER pad, sampling
          analog inputs now.
ANDREA: -respond to dmal
```

-put AUD3DAT on bus, prepare for DMA memory write rather than normal AUD3DAT memory read.
MARY: -decode AUD3DAT, rga strobe on SAMPRGA* pad.
EXTERNAL: -detect AUD3DAT strobe on SAMPRGA* pad put sample value(s) onto the AD bus.
ANDREA: -complete memory write.

9.3.2 Audio Sampling Tradeoffs. -

Normal pot operation is not possible when in the sampling mode. This is because one counter is used to drive both left and right ports. When the POTRX and POTRY are being used for audio sampling, POTLX and POTLY will not operate.

In normal audio playback, 32 bit dma fetch cycles imply two data samples are being processed. Audio playback hardware waits for two period timeouts before fetching more data. As a result of this, when using audio channel 3 to drive an audio sampling mode, the period programmed into channel 3 must actually be twice the period desired. This limits the maximum sampling rate to 55KHz.

9.3.3 Standard Sampling Rates. -

The following table shows decimal period values that should be programmed to acheive industry 'standard' sampling rates.

Standard	f/280nS	PERIOD	Actual Frequency
32.0KHz	111.61	56	31.89KHz
44.1KHz	80.99	40	44.64KHz
48.0KHz	74.40	37	48.26KHz

TURBO mode in conjunction with the PERFINE mode can minimize the discrepancies between the audio frequencies MARY can achieve and those necessary to meet audio standards.

9.3.4 External Audio Hardware. -

External hardware is required to preprocess the audio waveform into a format the internal circuits can convert. The signal applied to the POTRX and POTRY pins is an AC current used to charge the pot capacitor. The current must proportional to the audio waveform. When the audio signal being sampled reaches maximum positive amplitude, the current applied to the pot pin must be relatively small such that the pot capacitor being charged just charges to the pot circuit threshold as the pot counter reaches 255. When the audio signal reaches maximum negative amplitude, the current applied should be relatively large so that the pot capacitor charges to the pot circuit threshold as the pot counter reaches a count of 1. When the audio signal has zero amplitude, the current should charge the pot capacitor to the pot circuit threshold as the pot counter reaches 128.

At the end of each sample period, the pot capacitors must be discharged before the cycle can begin again. The discharge period is sixteen BUSCLK cycles (~1.2usec). It is not practical to design a large enough discharge transistor into the chip to discharge the pot

capacitors in the short time alloted. To accomodate capacitor discharge, two of the mouse pins are redefined to become outputs. These outputs should be used to drive external bipolar transistors which in turn are used to discharge the pot capacitors. One of the mouse pins was chosen for this function so that all of the signals necessary to support audio sampling mode would be available on the right mouse port connector. (The left mouse port connector is still available for the standard mouse functions.) The RRIGHT pin should be connected to the gates of two bipolar transistors. One transistor source is connected to the POTRX and the other is connected to the POTRY pin. The drain of both transistors is tied to ground. The chip drives these transistors for sixteen BUSCLK cycles at the end of each sampling period to discharge the capacitors associated with the POTRX and POTRY pins. After sixteen BUSCLK cycles, the transistors are switched off, and the external current source begins charging the capacitors. When the pot threshold is reached, a schmidt trigger switches and causes the current potcounter value to be latched.

A different set of external hardware is required when an external A to D convertor is used. This hardware cannot be situated on the right mouse port because the external device must have access to the chip AD bus. In addition to the A to D convertor, a tristate output D flip-flop function must be provided. The flip-flop latches the A to D convertor output. The tristate output drives the flip-flop output onto the custom chip AD bus. MARY drives the SAMPPER output pin at a rate exactly equal to half the period programmed into audio channel 3. This signal is used to latch the output of the D to A convertor, and to start the next conversion. It is a positive going pulse which lasts for one BUSCLK cycle (~70 nsec). A chip bus DMA cycle is requested and run after each sample is latched. When the DMA cycle is granted and run, MARY drives a logic low on the SAMPRGA* pin. This signal is used to enable the tristate buffer onto the AD bus.

9.4 UART.

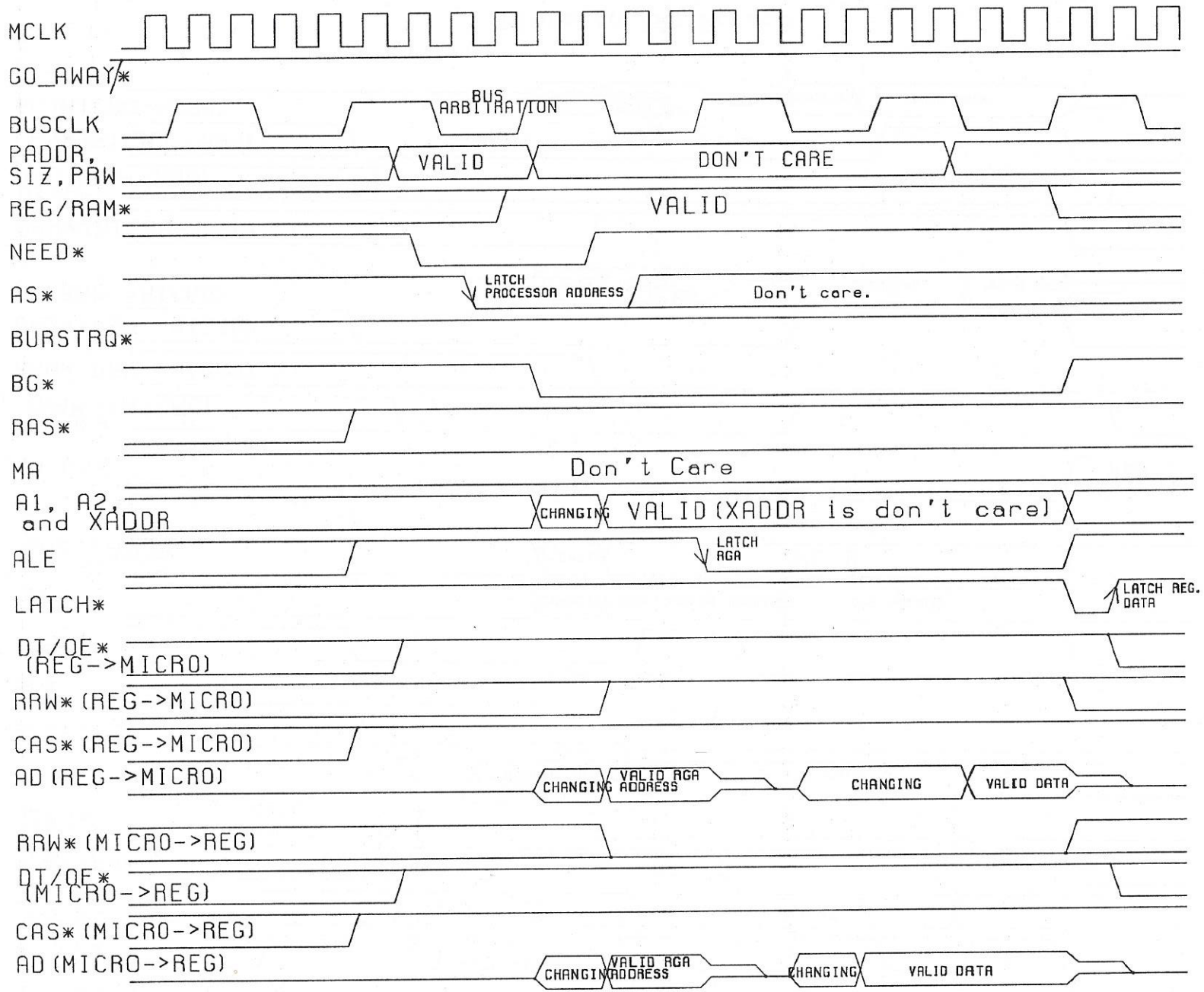
A second UART has been added to the system. This UART is identical to the original UART in every respect except of course the RGA addresses. See the regbits document for the second UART register addresses.

In addition to the new UART, both UARTs have been extended to contain a receive port fifo buffer. These fifo's are 4 bytes long and allow more system latency time to support higher receive baud rates. A new bit has been added to the SERDATR register, namely RMORE. This bit is used to determine if there is more valid data in the fifo. If this bit is set, then the UART device driver should read another word from the receive port. If the bit is clear, the fifo is empty. The serial recieve ports operate in the same manner as they have worked in the past. In order to flush the current receive word from the fifo, the RFB bit of INTENA must be cleared by software. If this action is not taken, then the processor will reread the same data word with each receive buffer access. The envisioned sequence of events then is to wait for a receive buffer interrupt. When this happens, software should:

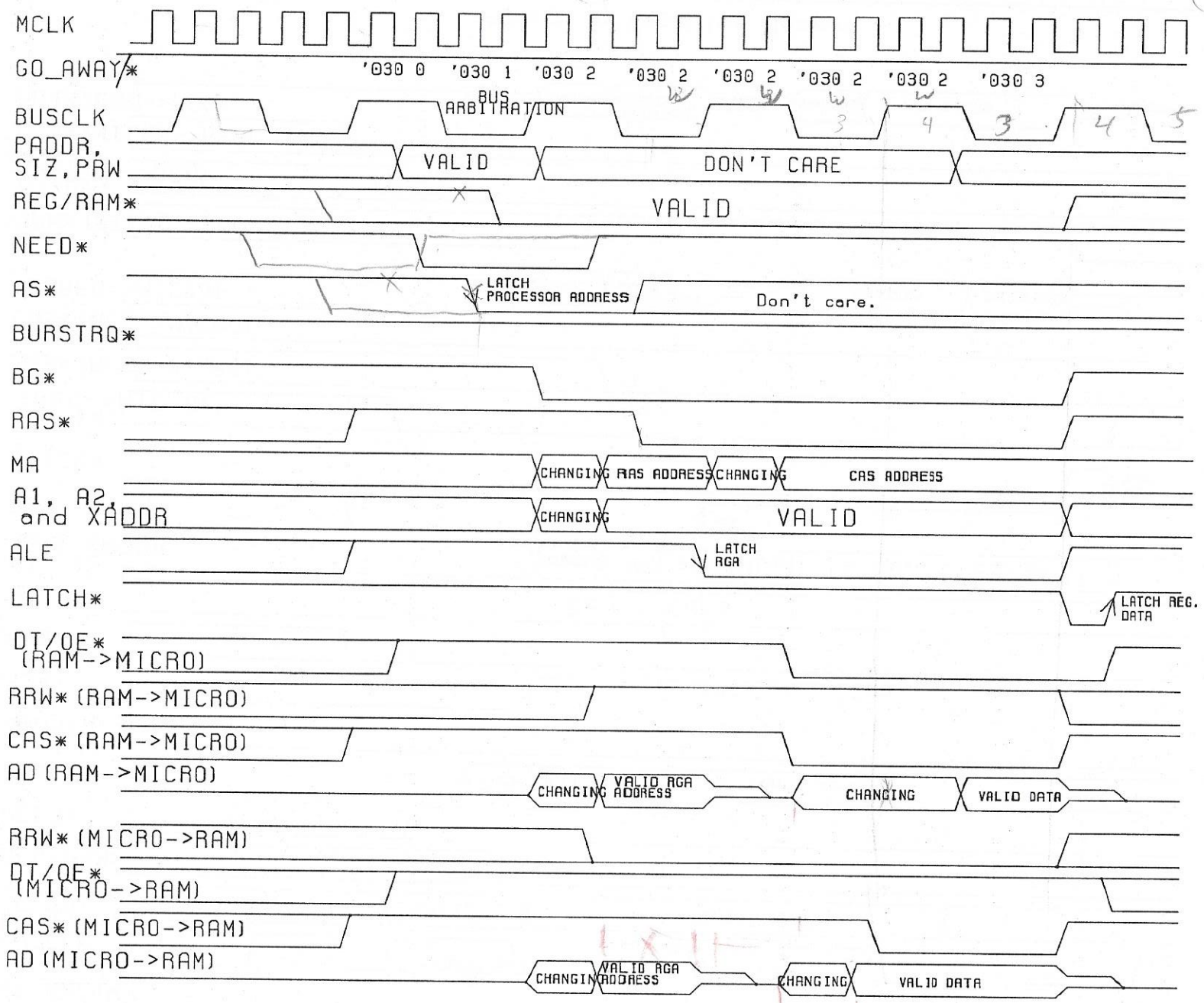
1) read the SERDATR register.
2) clear RFB of INTENA.
3) check RMORE from the SERDATR access. If 1, go to step 1 for more data. If 0, finish UART processing and return from interrupt.

APPENDIX A

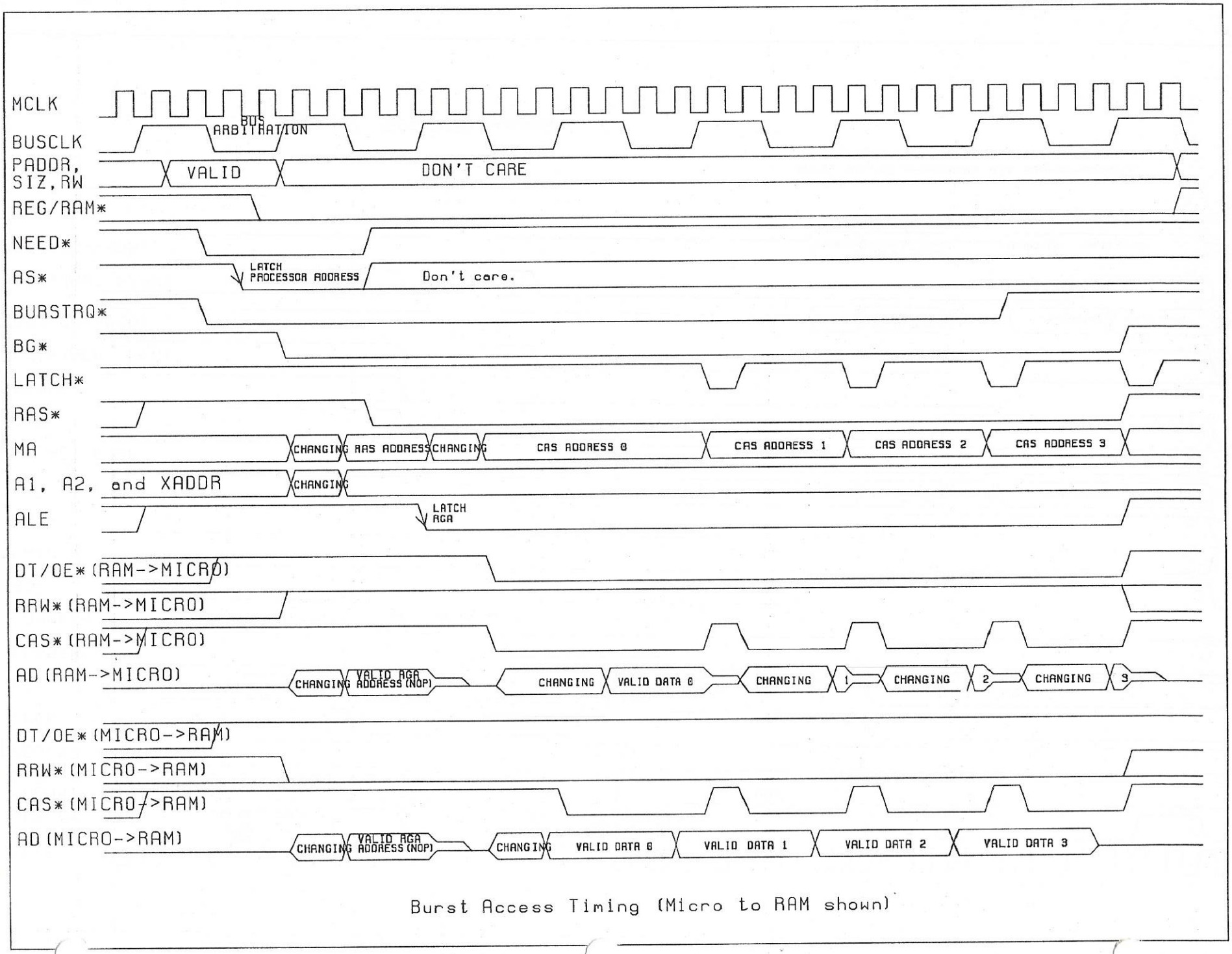
BUS TIMING DIAGRAMS.



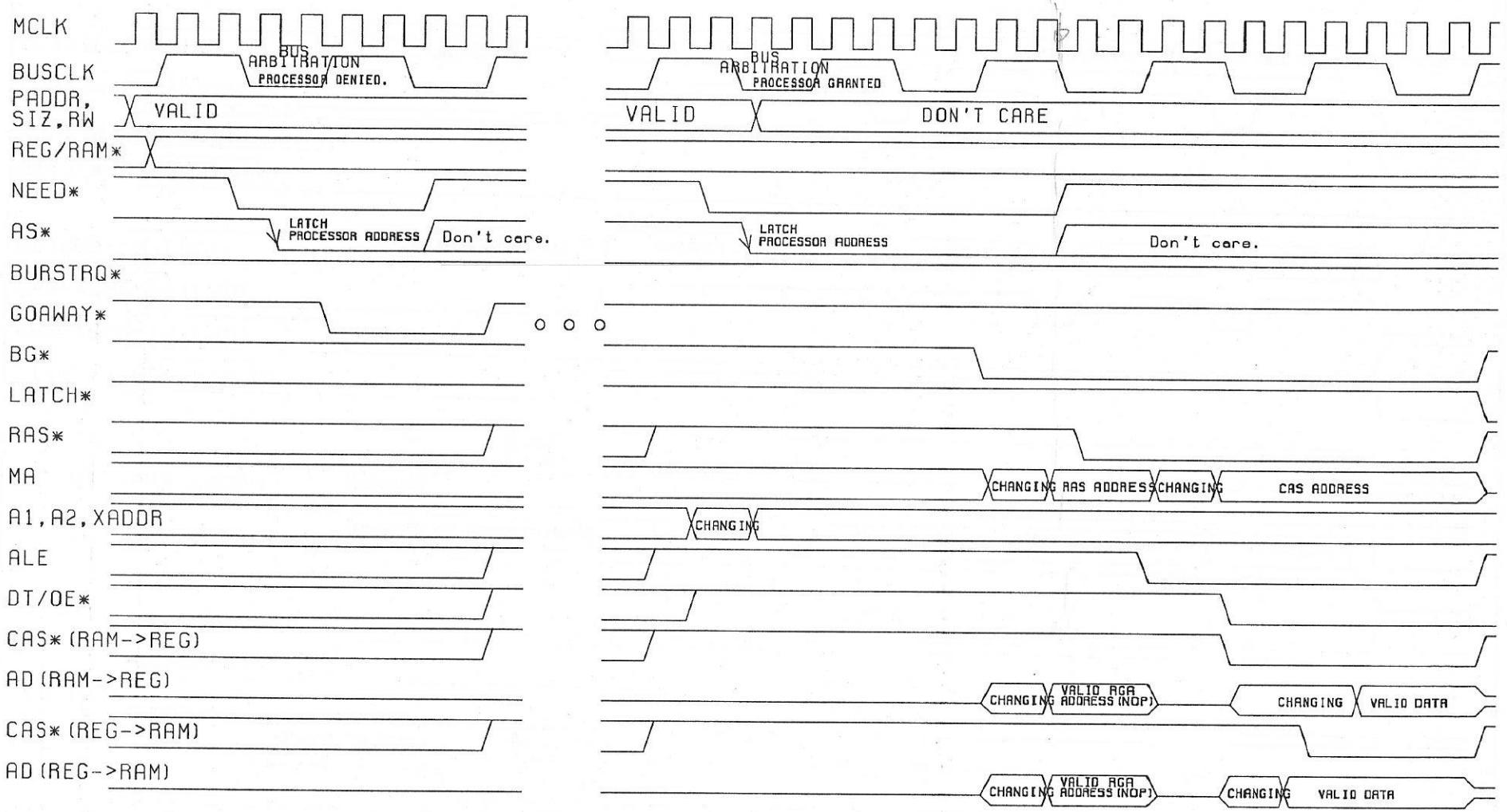
Micro to Register Access



Micro to RAM Access

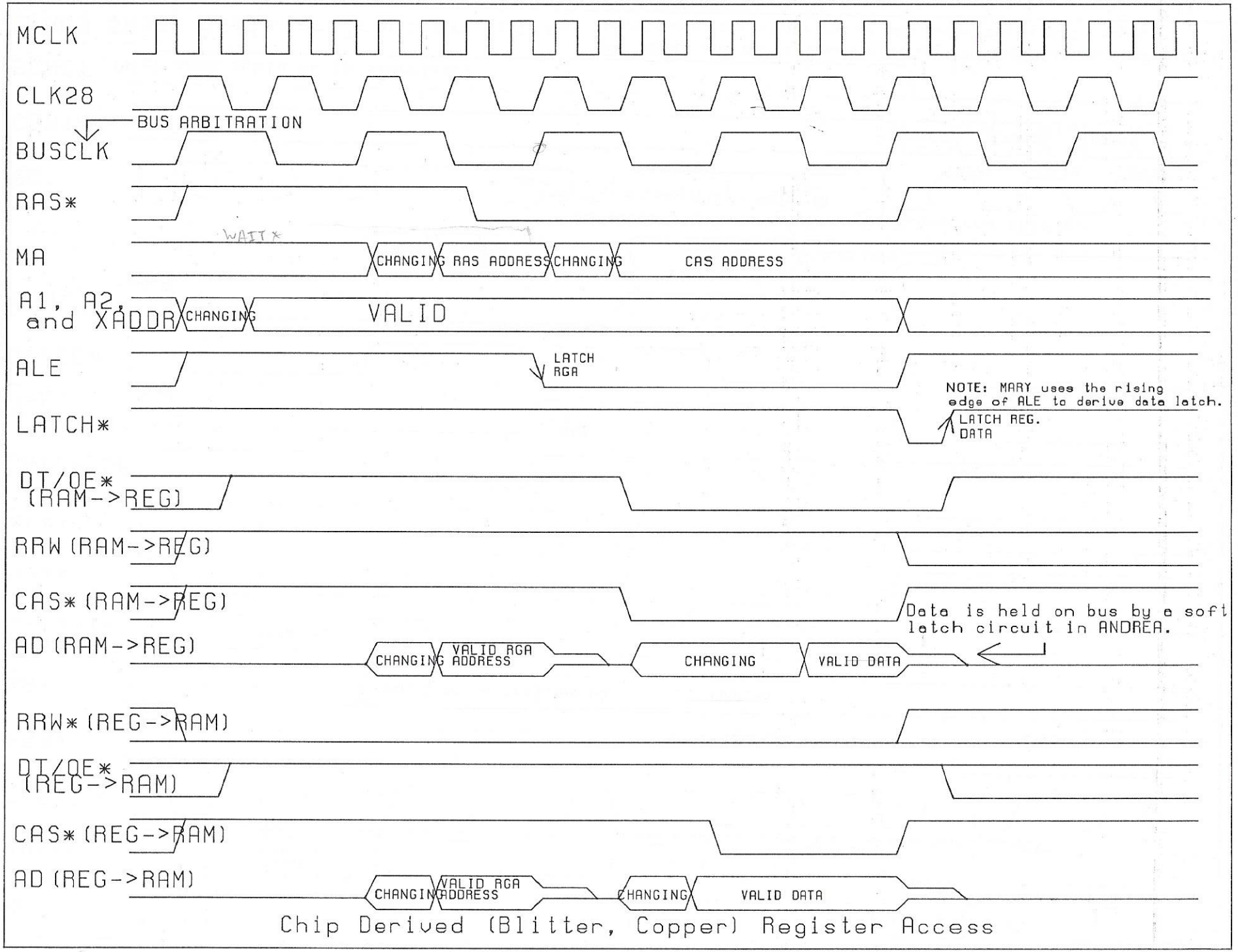


Burst Access Timing (Micro to RAM shown)

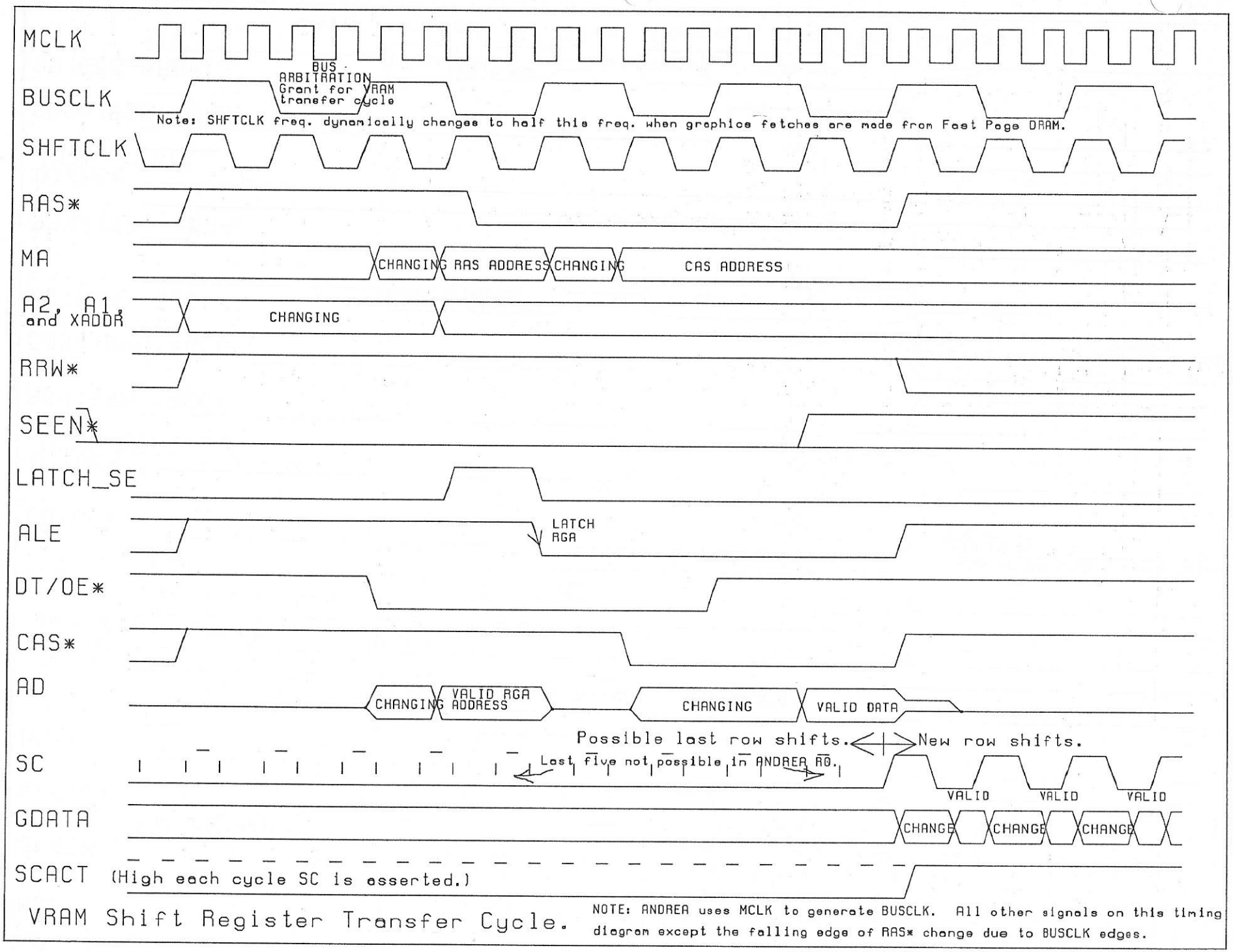


NOTE: ANDREA processes AS* asynchronously.
AS* is used as a final qualification of NEED*.

Typical 'GOAWAY' Sequence.

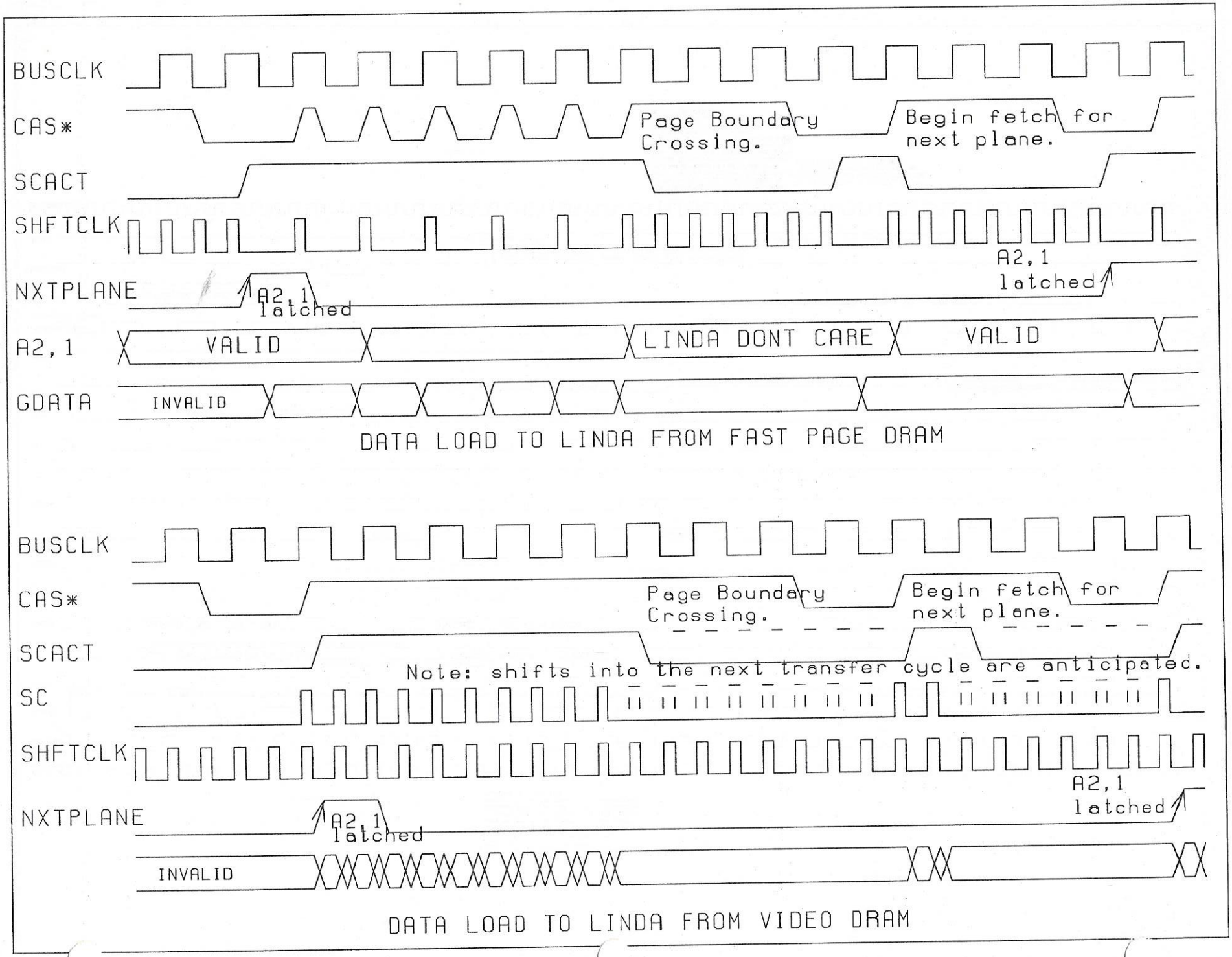


Chip Derived (Blitter, Copper) Register Access



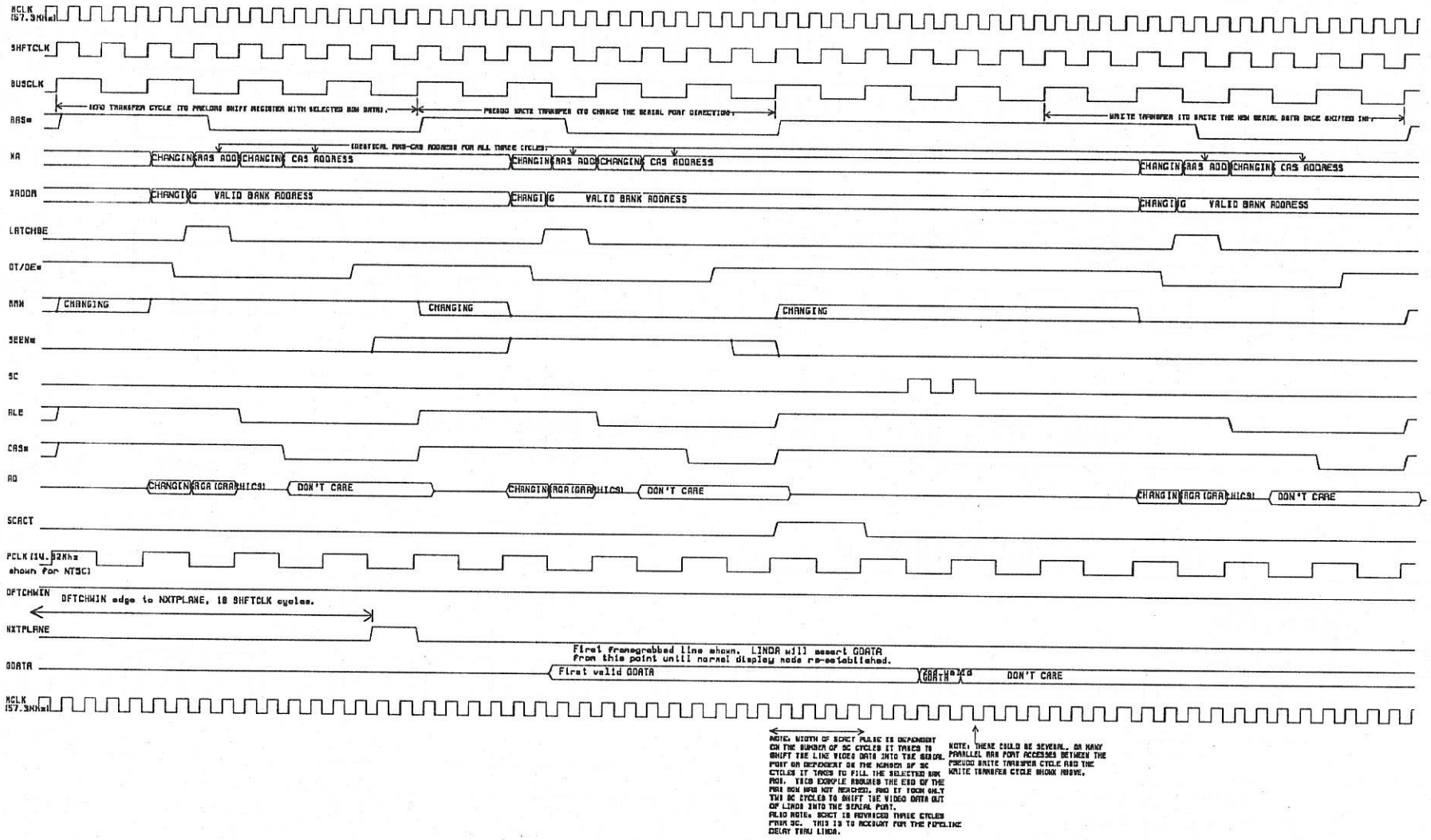
VRAM Shift Register Transfer Cycle.

NOTE: ANDREA uses MCLK to generate BUSCLK. All other signals on this timing diagram except the falling edge of RAS* change due to BUSCLK edges.


BUSCLK
CAS*
Page Boundary Crossing.
Begin fetch for next plane.
SCACT
SHFTCLK
A2,1 latched
NXTPLANE
A2,1 latched
A2,1
VALID
LINDA DONT CARE
VALID
GDATA
INVALID
DATA LOAD TO LINDA FROM FAST PAGE DRAM
BUSCLK
CAS*
Page Boundary Crossing.
Begin fetch for next plane.
SCACT
Note: shifts into the next transfer cycle are anticipated.
SC
SHFTCLK
A2,1 latched
NXTPLANE
A2,1 latched
INVALID
DATA LOAD TO LINDA FROM VIDEO DRAM

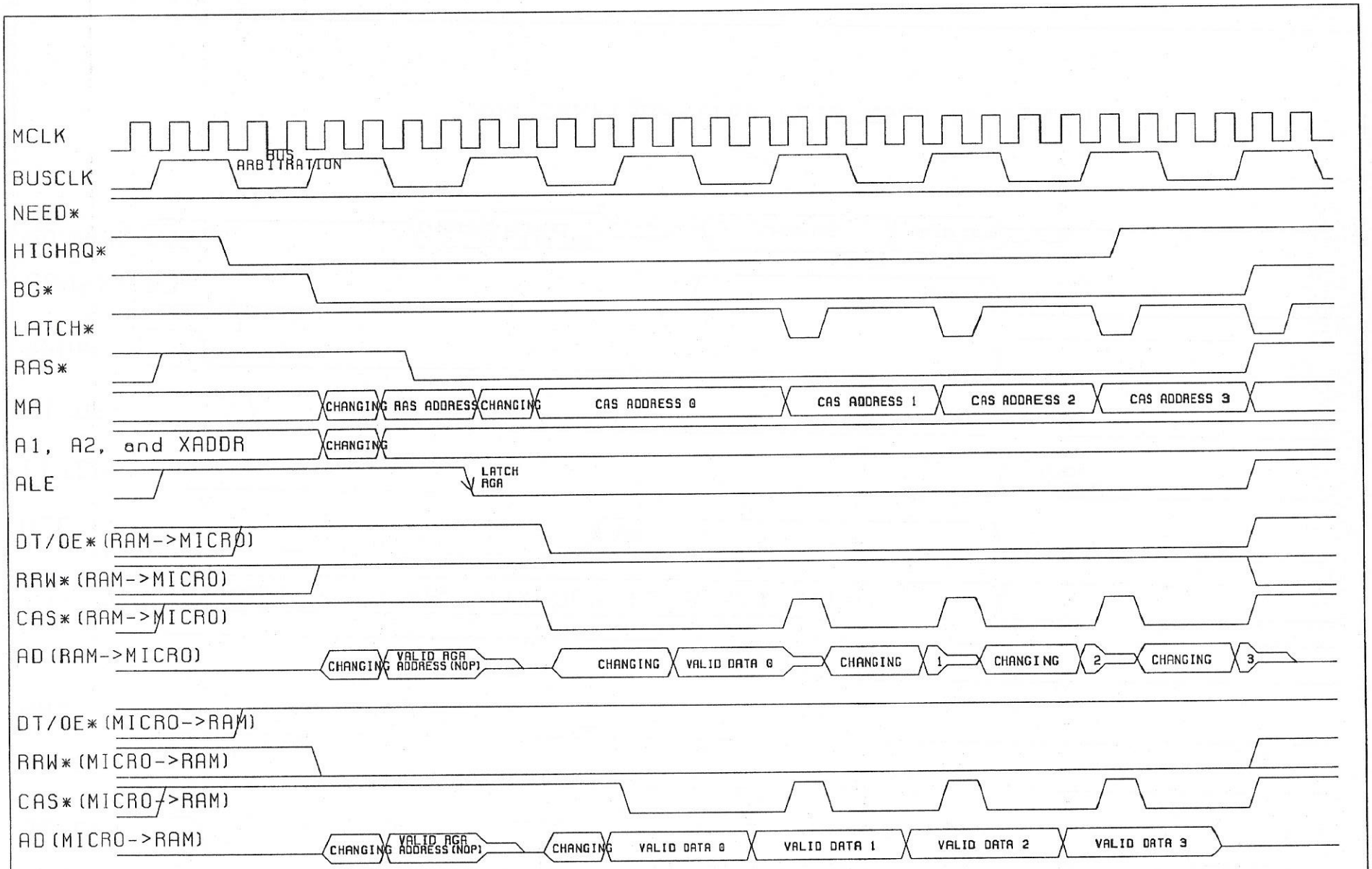
SERIAL PORT WRITE SEQUENCE

THIS SEQUENCE OF CYCLES IS IMPLEMENTED TO SUPPORT FRAMEGRABBER MODE. IT IS REPEATED ONCE FOR EACH ACTIVE SCAN LINE (TWICE TIMES FOR HYBRID PIXELS... DELAYED ONE LINE TO ACCOUNT FOR LINE DELAY THRU LINDA), AND IT IS ASSERTED WHENEVER A RAM PAGE BOUNDARY IS CROSSED WHILE SHIFTING IN A SCAN LINE.



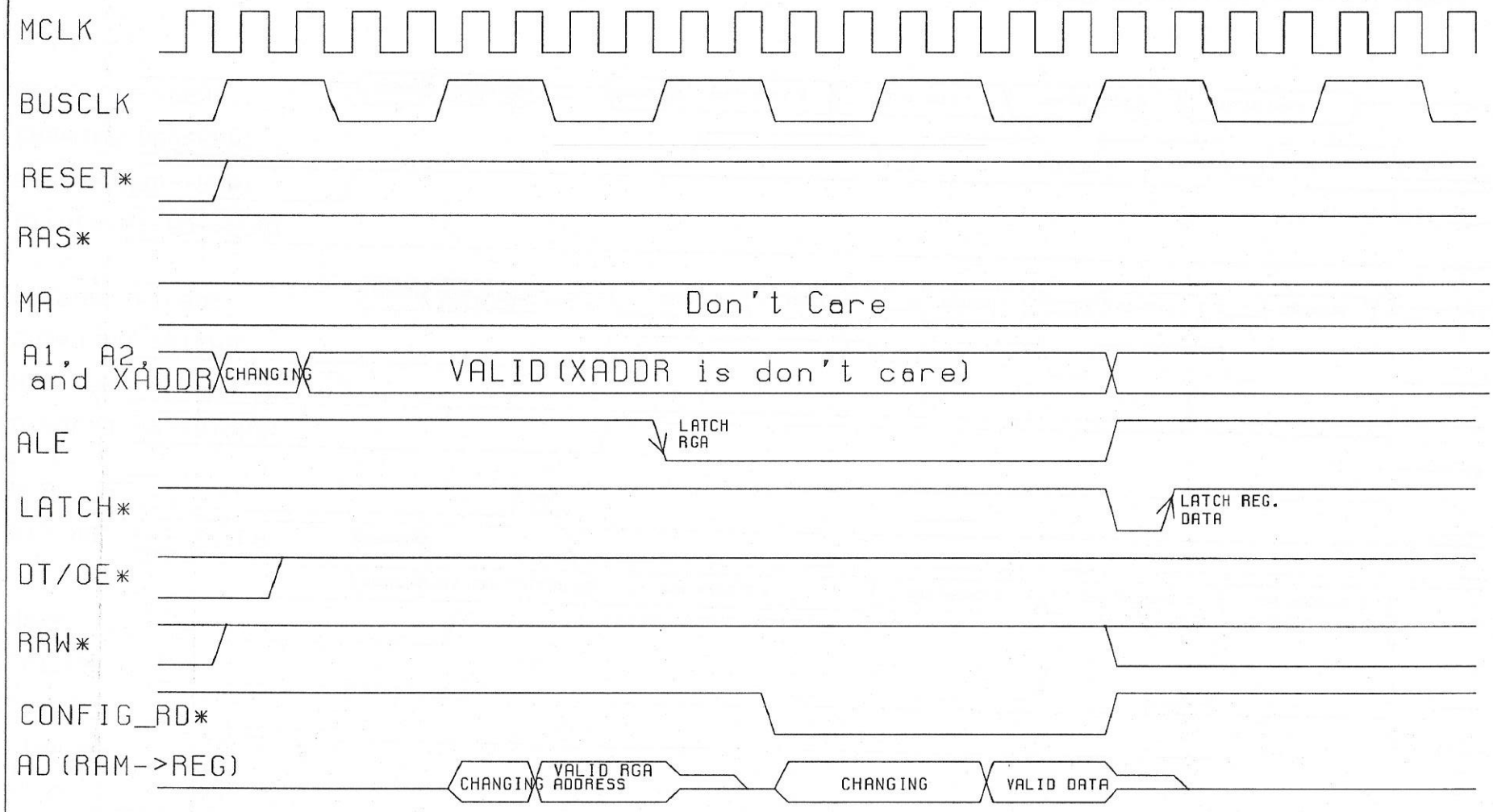
NOTE: WIDTH OF SCACT PULSE IS DEPENDENT ON THE NUMBER OF SC CYCLES IT TAKES TO SHIFT THE LINE VIDEO DATA INTO THE SERIAL PORT (A DEPENDENT ON THE NUMBER OF SC CYCLES IT TAKES TO FILL THE SELECTED RAM PAGE. THIS EXAMPLE ASSUMES THE END OF THE PAGE HAS NOT REACHED, AND IT TOOK ONLY TWO SC CYCLES TO SHIFT THE VIDEO DATA OUT OF LINDA INTO THE SERIAL PORT.
ALSO NOTE: SCACT IS ADVANCED THREE CYCLES FROM SC. THIS IS TO ACCOUNT FOR THE PIPELINE DELAY THRU LINDA.

NOTE: THERE COULD BE SEVERAL, OR MANY PARALLEL RAM PORT ACCESSES BETWEEN THE PSEUDO WRITE TRANSFER CYCLE AND THE WRITE TRANSFER CYCLE SHOWN ABOVE.

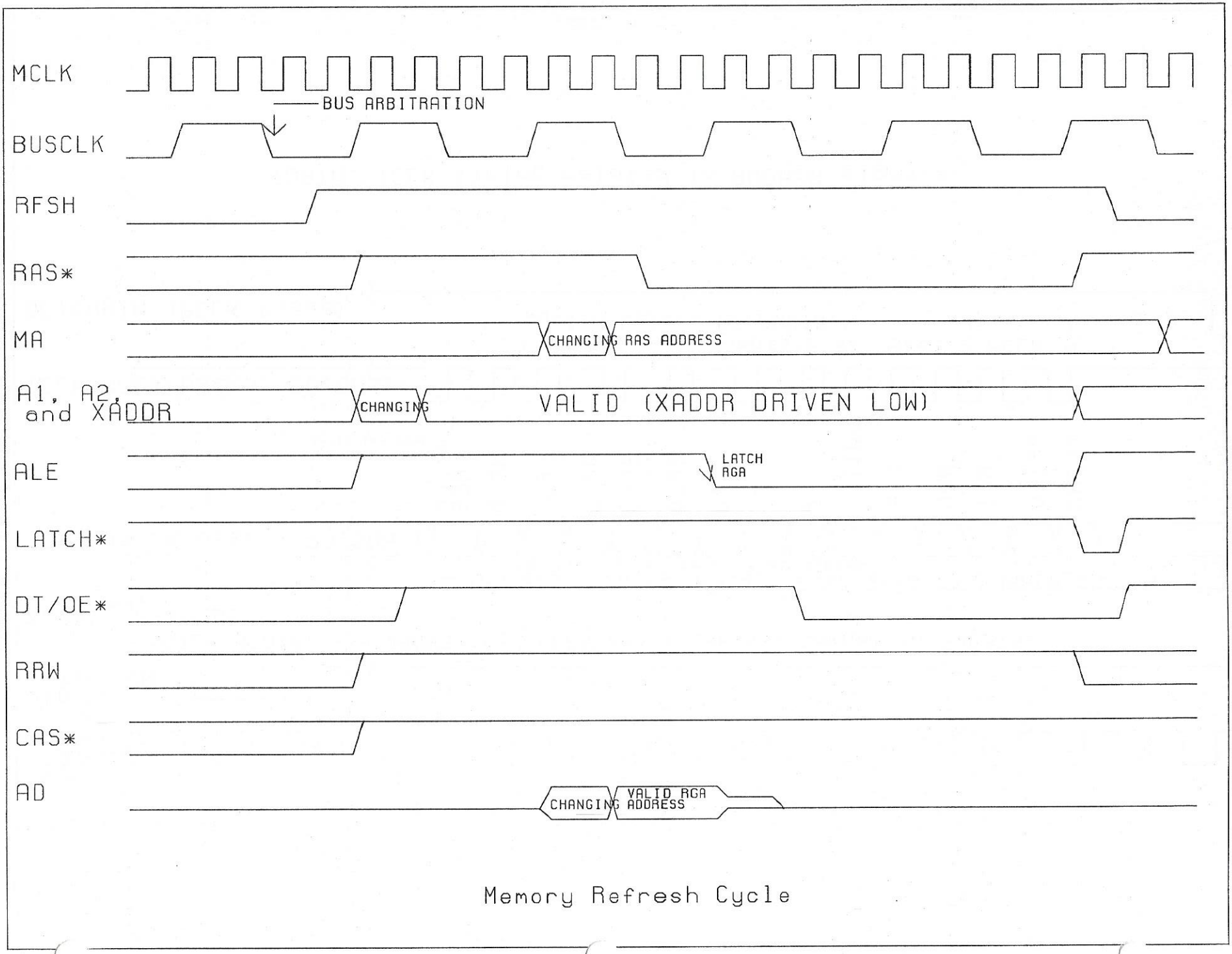


HIGHRQ* Cycle Timing (Four transfer burst cycle shown)

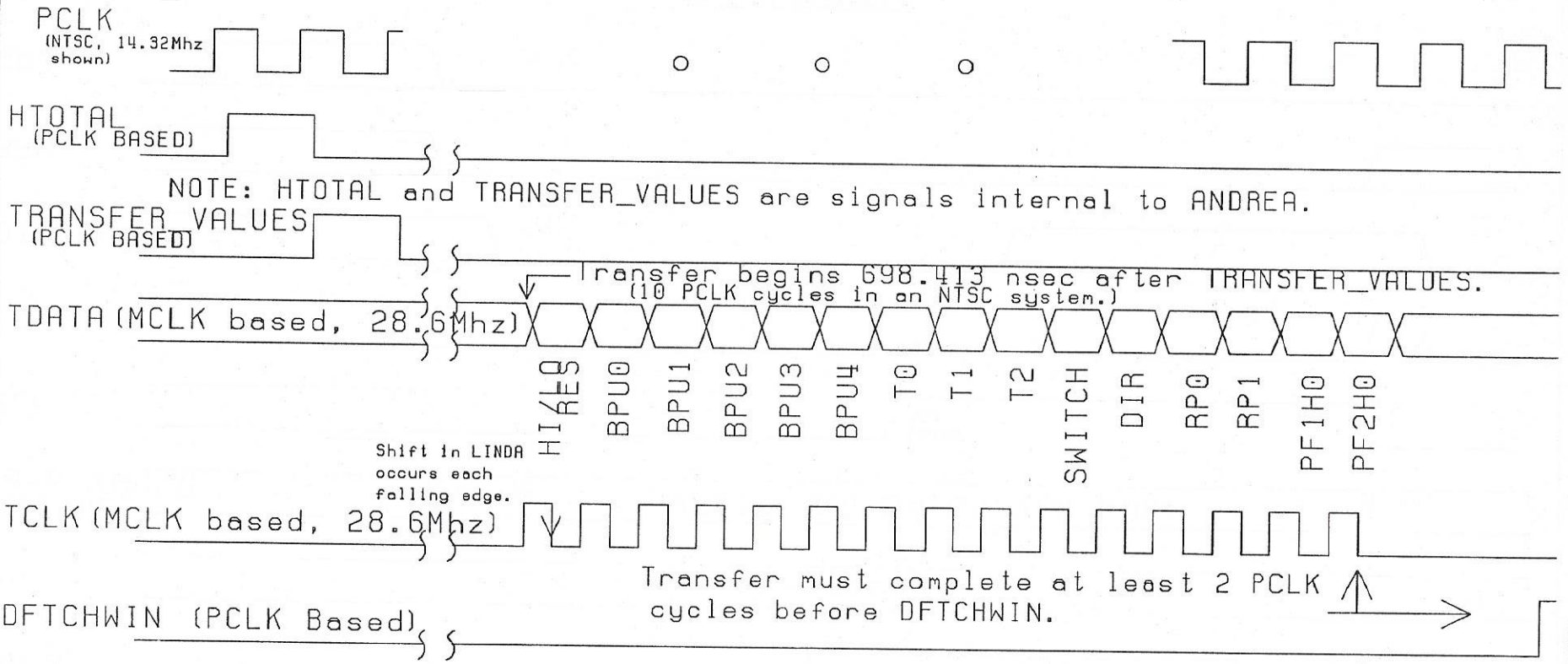
Note: ANDREA tristates all bus control signals during this cycle (see written spec for list). The external bus master circuits are expected to provide the timings shown for all bus control signals. Also, it is not possible to write ANDREA's registers with this cycle type.



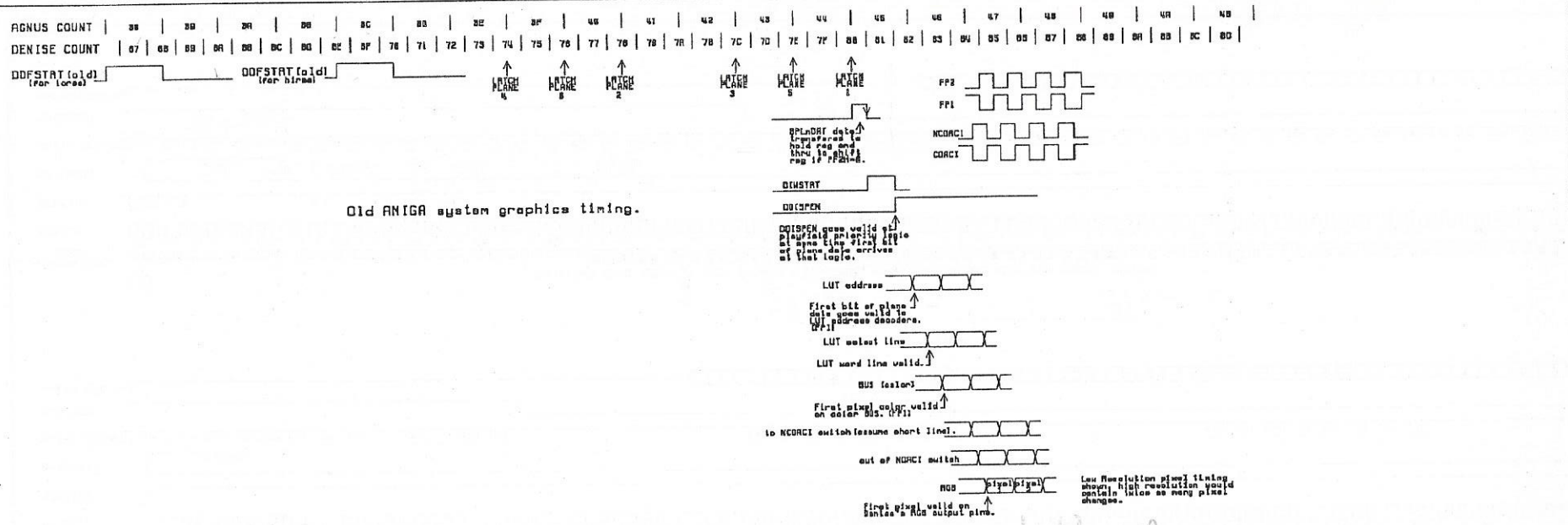
RAM Configuration Data Read Access



Memory Refresh Cycle



TDATA, TCLK timing related to ANDREA signals.

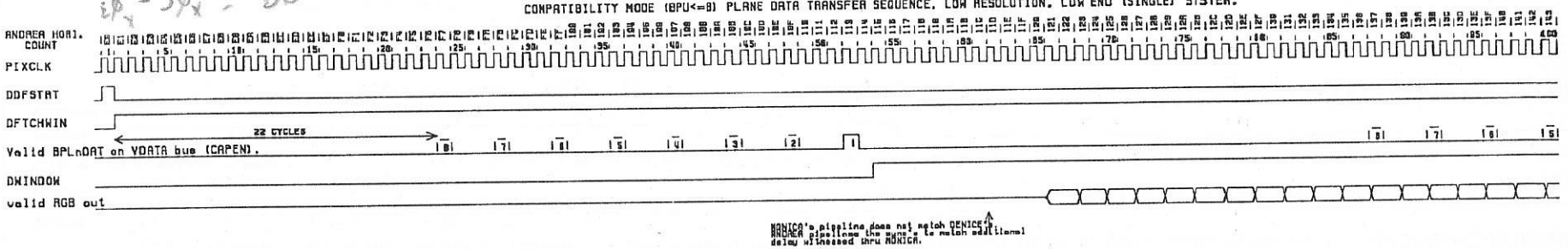


Old AMIGA system graphics timing.

7 pixels thru Denise

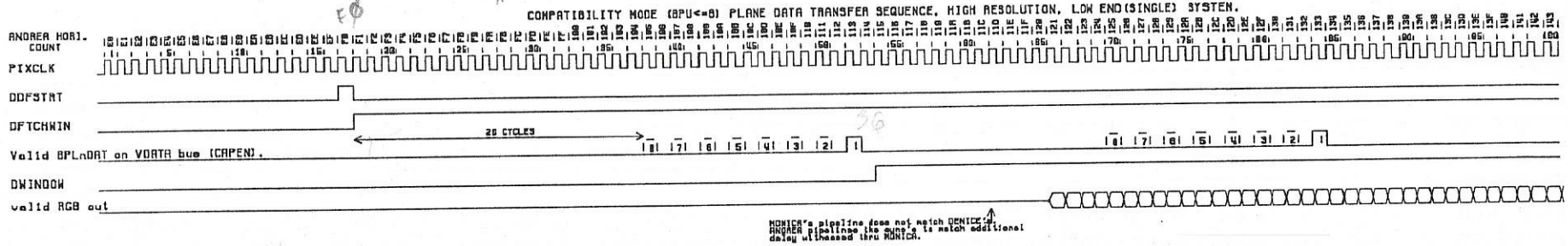
$E\phi_x - 3\phi_x = B\phi$

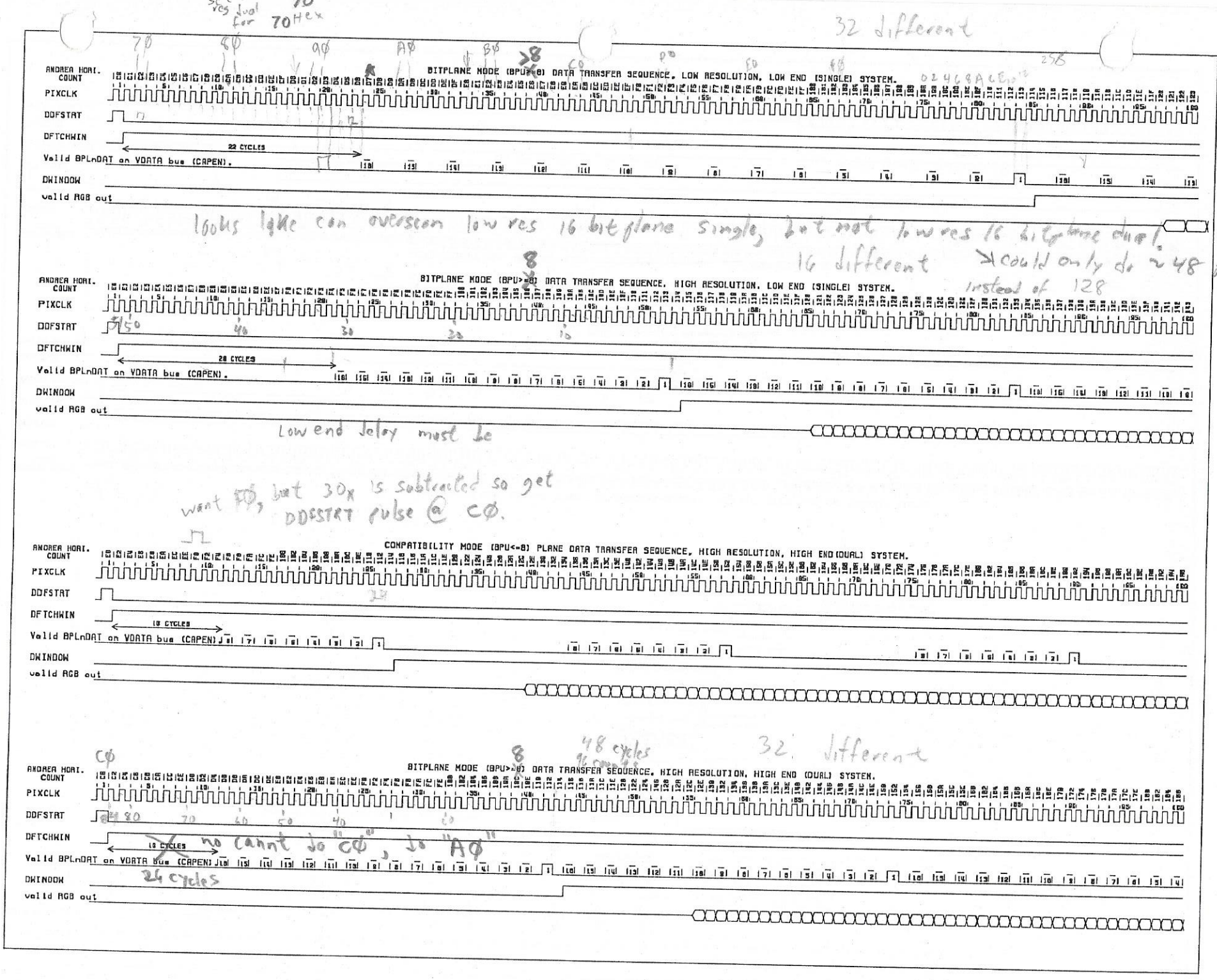
COMPATIBILITY MODE (BPU<=8) PLANE DATA TRANSFER SEQUENCE, LOW RESOLUTION, LOW END (SINGLE) SYSTEM.



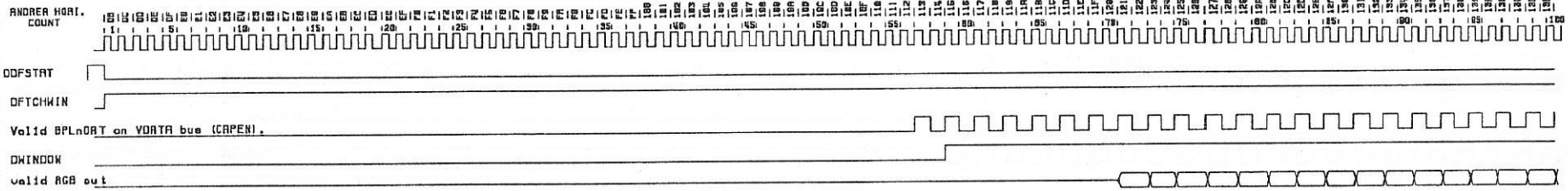
$F\phi_x - 3\phi_x = C\phi$

COMPATIBILITY MODE (BPU<=8) PLANE DATA TRANSFER SEQUENCE, HIGH RESOLUTION, LOW END (SINGLE) SYSTEM.

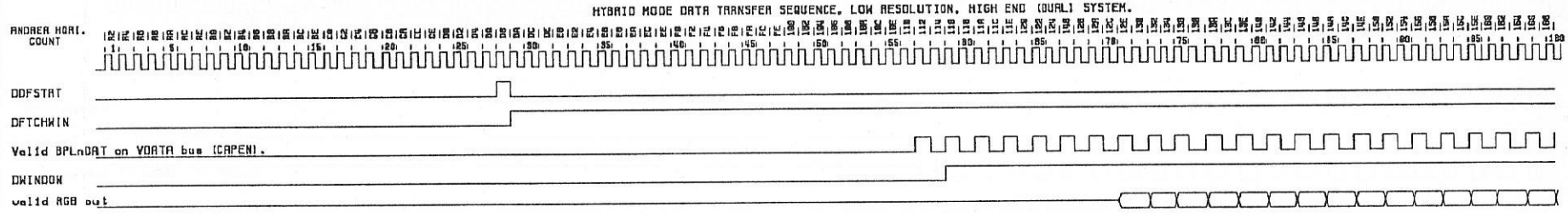



ANDREA HORI. COUNT
BITPLANE MODE (BPU>=8) DATA TRANSFER SEQUENCE, LOW RESOLUTION, LOW END (SINGLE) SYSTEM.
PIXCLK
DDFSTAT
DFTCHWIN
22 CYCLES
Valid BPLnDAT on VDATA bus (CAPEN).
DWINDOW
valid RGB out
looks like can overscan low res 16 bitplane single, but not low res 16 bitplane dual.
16 different
could only do ~48 pixels
instead of 128
BITPLANE MODE (BPU>=8) DATA TRANSFER SEQUENCE, HIGH RESOLUTION, LOW END (SINGLE) SYSTEM.
28 CYCLES
Low end delay must be
want F0, but 30x is subtracted so get DDFSTRT pulse @ C0.
COMPATIBILITY MODE (BPU<=8) PLANE DATA TRANSFER SEQUENCE, HIGH RESOLUTION, HIGH END (DUAL) SYSTEM.
10 CYCLES
BITPLANE MODE (BPU>=8) DATA TRANSFER SEQUENCE, HIGH RESOLUTION, HIGH END (DUAL) SYSTEM.
48 cycles
32 different
no cannot do "C0", do "A0"
24 cycles

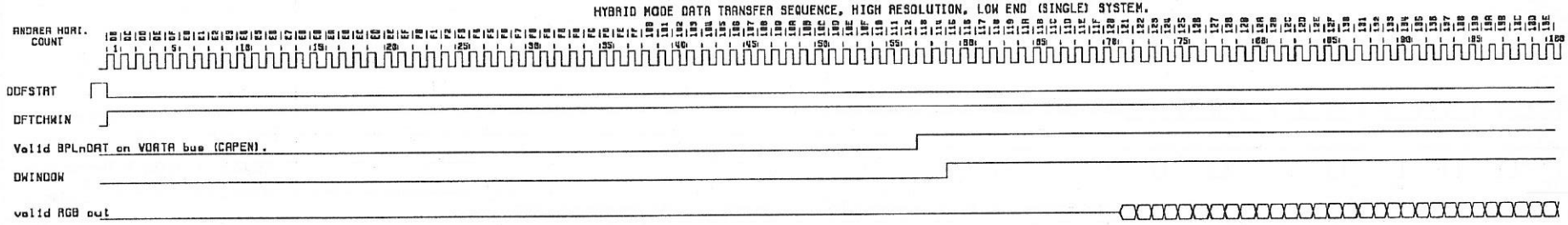
HYBRID MODE DATA TRANSFER SEQUENCE, LOW RESOLUTION, LOW END (SINGLE) SYSTEM.



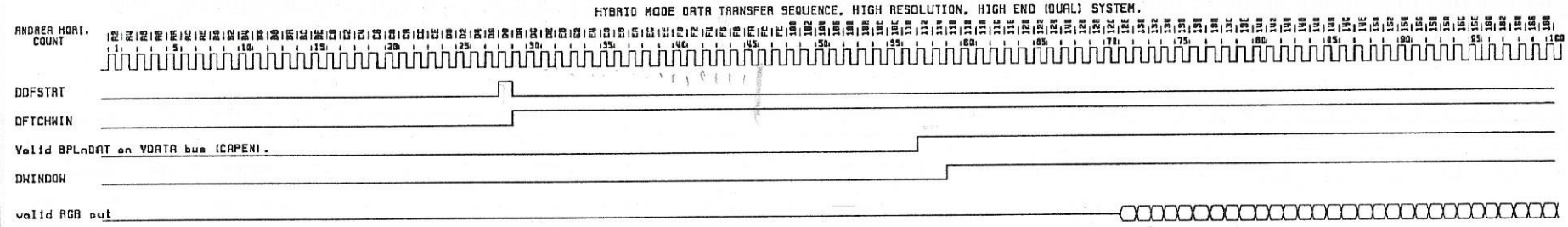
HYBRID MODE DATA TRANSFER SEQUENCE, LOW RESOLUTION, HIGH END (DUAL) SYSTEM.

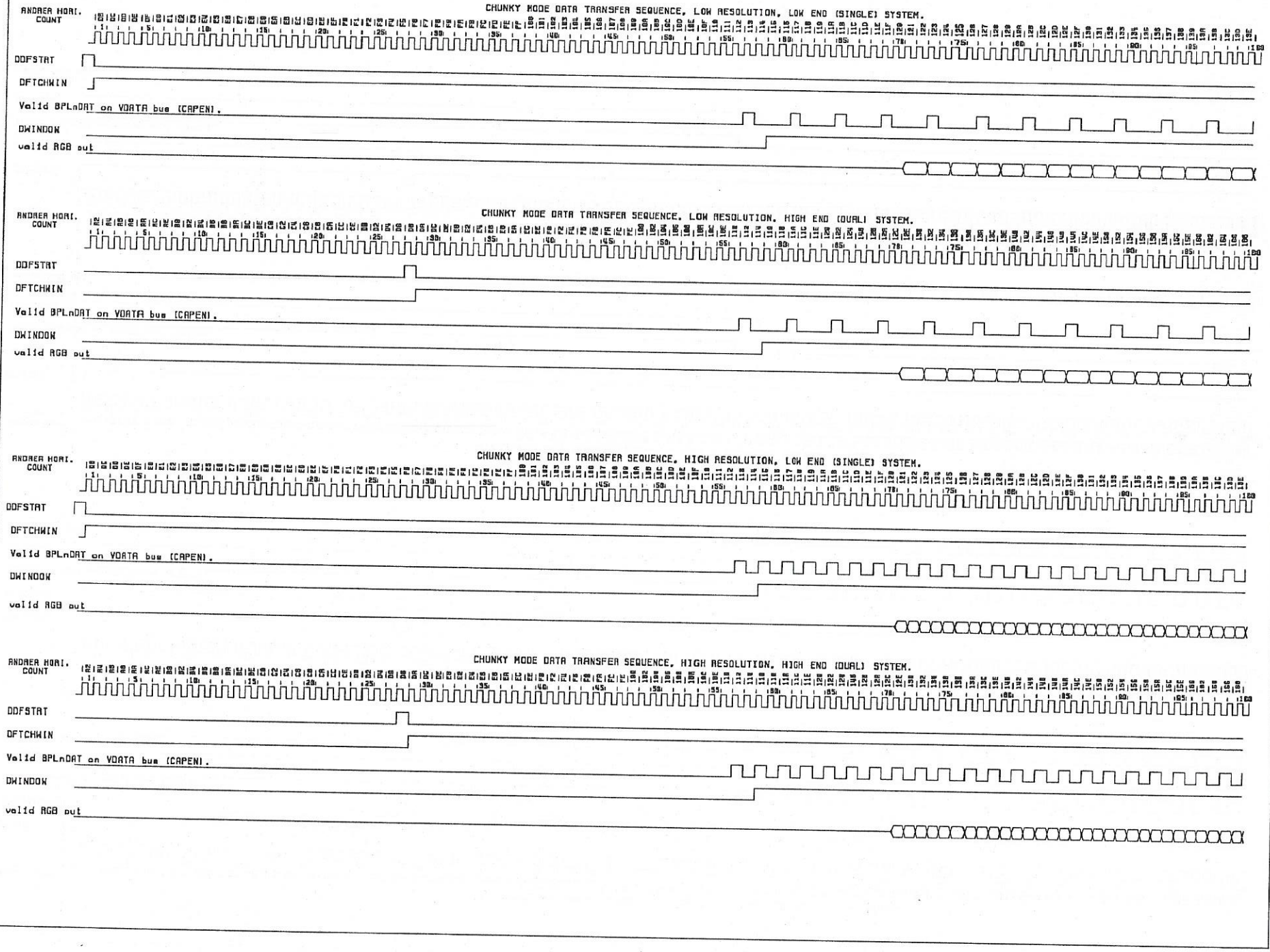


HYBRID MODE DATA TRANSFER SEQUENCE, HIGH RESOLUTION, LOW END (SINGLE) SYSTEM.

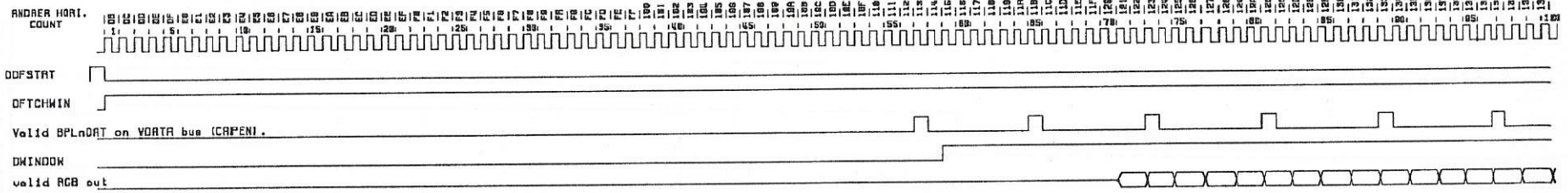


HYBRID MODE DATA TRANSFER SEQUENCE, HIGH RESOLUTION, HIGH END (DUAL) SYSTEM.

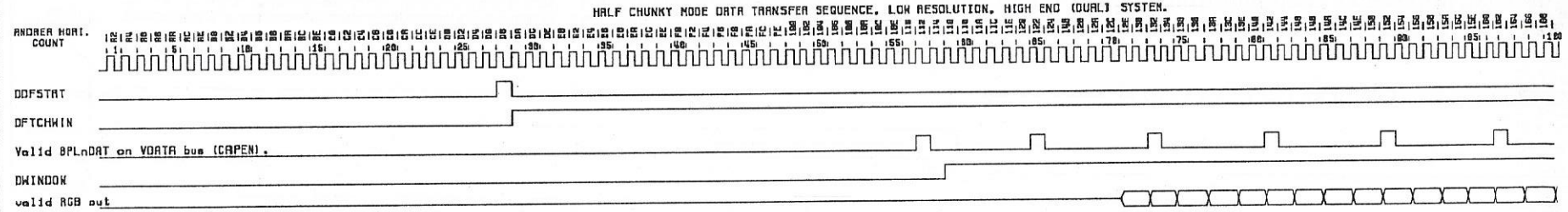



CHUNKY MODE DATA TRANSFER SEQUENCE, LOW RESOLUTION, LOW END (SINGLE) SYSTEM.
ANDREA HORI. COUNT
DDFSTRT
DFTCHWIN
Valid BPLnDAT on VDATA bus (CAPEN).
DWINDOW
valid RGB out
CHUNKY MODE DATA TRANSFER SEQUENCE, LOW RESOLUTION, HIGH END (DUAL) SYSTEM.
ANDREA HORI. COUNT
DDFSTRT
DFTCHWIN
Valid BPLnDAT on VDATA bus (CAPEN).
DWINDOW
valid RGB out
CHUNKY MODE DATA TRANSFER SEQUENCE, HIGH RESOLUTION, LOW END (SINGLE) SYSTEM.
ANDREA HORI. COUNT
DDFSTRT
DFTCHWIN
Valid BPLnDAT on VDATA bus (CAPEN).
DWINDOW
valid RGB out
CHUNKY MODE DATA TRANSFER SEQUENCE, HIGH RESOLUTION, HIGH END (DUAL) SYSTEM.
ANDREA HORI. COUNT
DDFSTRT
DFTCHWIN
Valid BPLnDAT on VDATA bus (CAPEN).
DWINDOW
valid RGB out

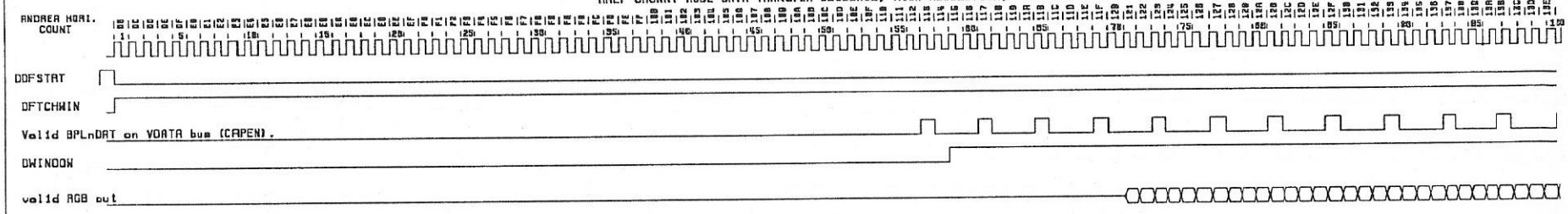
HALF CHUNKY MODE DATA TRANSFER SEQUENCE, LOW RESOLUTION, LOW END (SINGLE) SYSTEM.



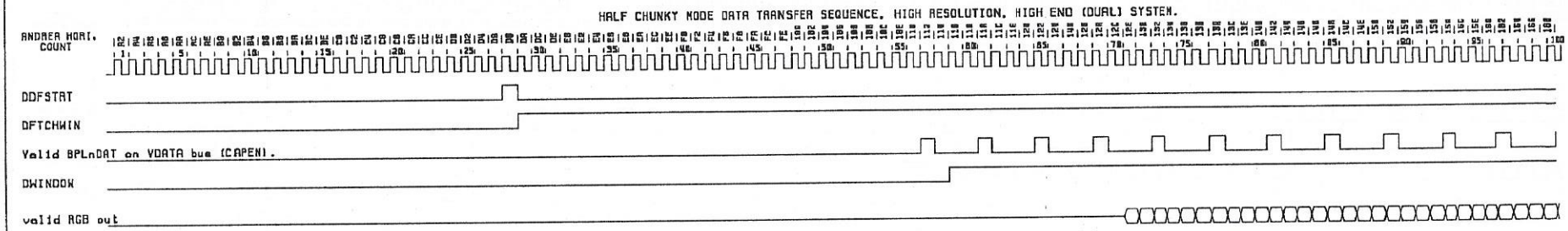
HALF CHUNKY MODE DATA TRANSFER SEQUENCE, LOW RESOLUTION, HIGH END (DUAL) SYSTEM.

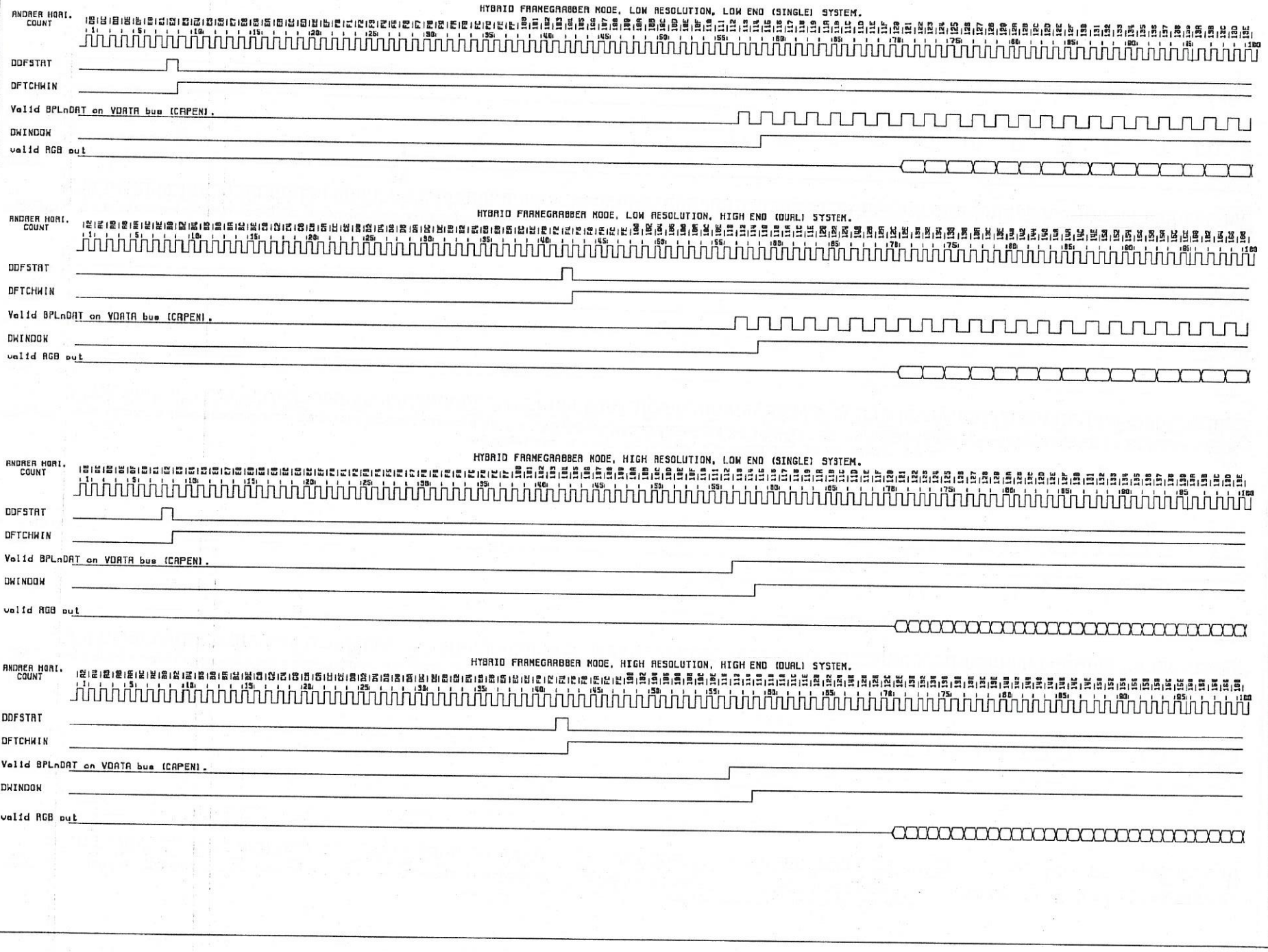


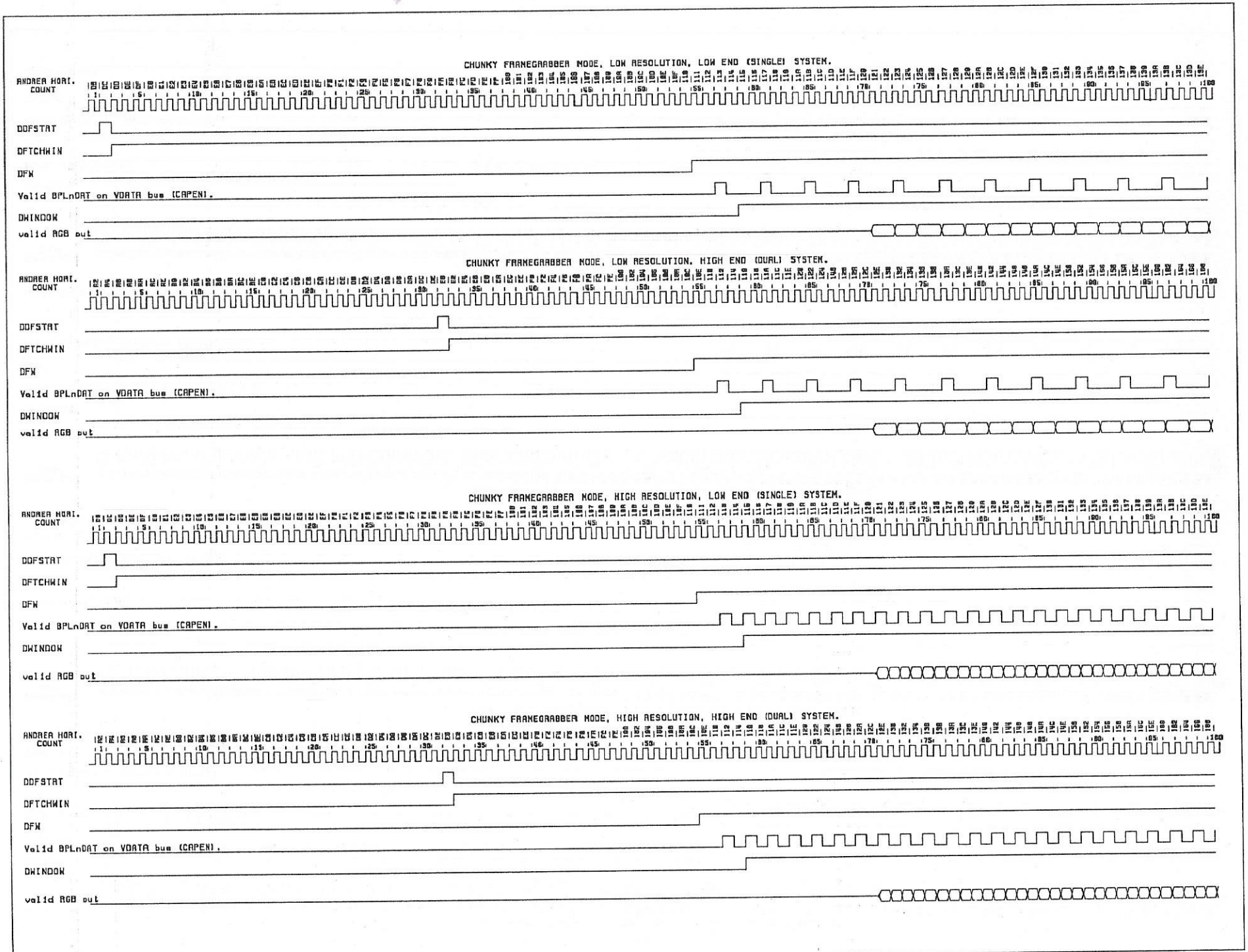
HALF CHUNKY MODE DATA TRANSFER SEQUENCE, HIGH RESOLUTION, LOW END (SINGLE) SYSTEM.

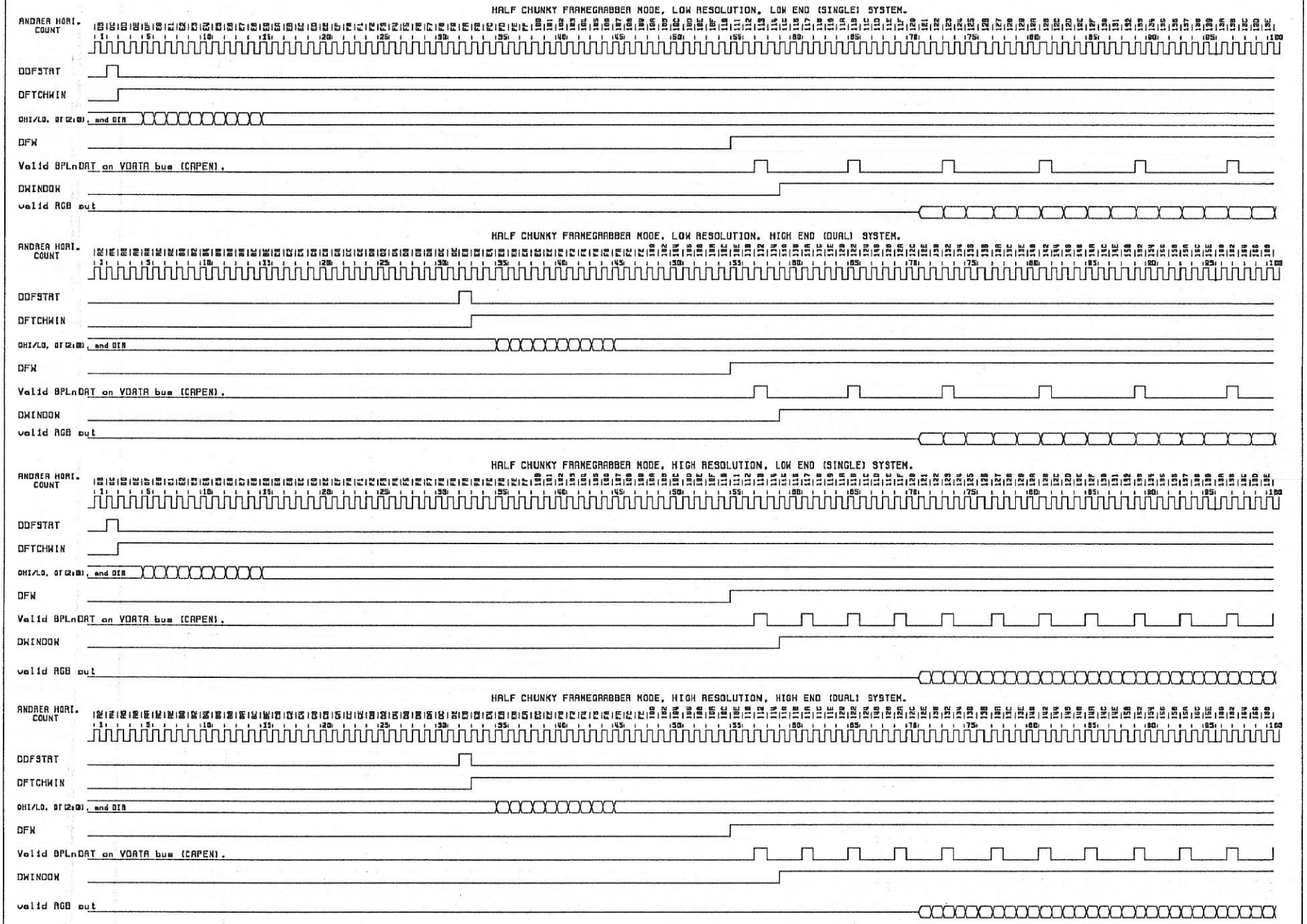


HALF CHUNKY MODE DATA TRANSFER SEQUENCE, HIGH RESOLUTION, HIGH END (DUAL) SYSTEM.




HYBRID FRAMEGRABBER MODE, LOW RESOLUTION, LOW END (SINGLE) SYSTEM.
ANDREA HORI. COUNT
DDFSTRT
DFTCHWIN
Valid BPLnDAT on VDATA bus (CAPEN).
DWINDOW
valid RGB out
HYBRID FRAMEGRABBER MODE, LOW RESOLUTION, HIGH END (DUAL) SYSTEM.
ANDREA HORI. COUNT
DDFSTRT
DFTCHWIN
Valid BPLnDAT on VDATA bus (CAPEN).
DWINDOW
valid RGB out
HYBRID FRAMEGRABBER MODE, HIGH RESOLUTION, LOW END (SINGLE) SYSTEM.
ANDREA HORI. COUNT
DDFSTRT
DFTCHWIN
Valid BPLnDAT on VDATA bus (CAPEN).
DWINDOW
valid RGB out
HYBRID FRAMEGRABBER MODE, HIGH RESOLUTION, HIGH END (DUAL) SYSTEM.
ANDREA HORI. COUNT
DDFSTRT
DFTCHWIN
Valid BPLnDAT on VDATA bus (CAPEN).
DWINDOW
valid RGB out


CHUNKY FRAMEGRABBER MODE, LOW RESOLUTION, LOW END (SINGLE) SYSTEM.
ANDREA HORI. COUNT
DDFSTRT
DFTCHWIN
DFW
Valid BPLnDAT on VDATA bus (CAPEN).
DWINDOW
valid RGB out
CHUNKY FRAMEGRABBER MODE, LOW RESOLUTION, HIGH END (DUAL) SYSTEM.
ANDREA HORI. COUNT
DDFSTRT
DFTCHWIN
DFW
Valid BPLnDAT on VDATA bus (CAPEN).
DWINDOW
valid RGB out
CHUNKY FRAMEGRABBER MODE, HIGH RESOLUTION, LOW END (SINGLE) SYSTEM.
ANDREA HORI. COUNT
DDFSTRT
DFTCHWIN
DFW
Valid BPLnDAT on VDATA bus (CAPEN).
DWINDOW
valid RGB out
CHUNKY FRAMEGRABBER MODE, HIGH RESOLUTION, HIGH END (DUAL) SYSTEM.
ANDREA HORI. COUNT
DDFSTRT
DFTCHWIN
DFW
Valid BPLnDAT on VDATA bus (CAPEN).
DWINDOW
valid RGB out

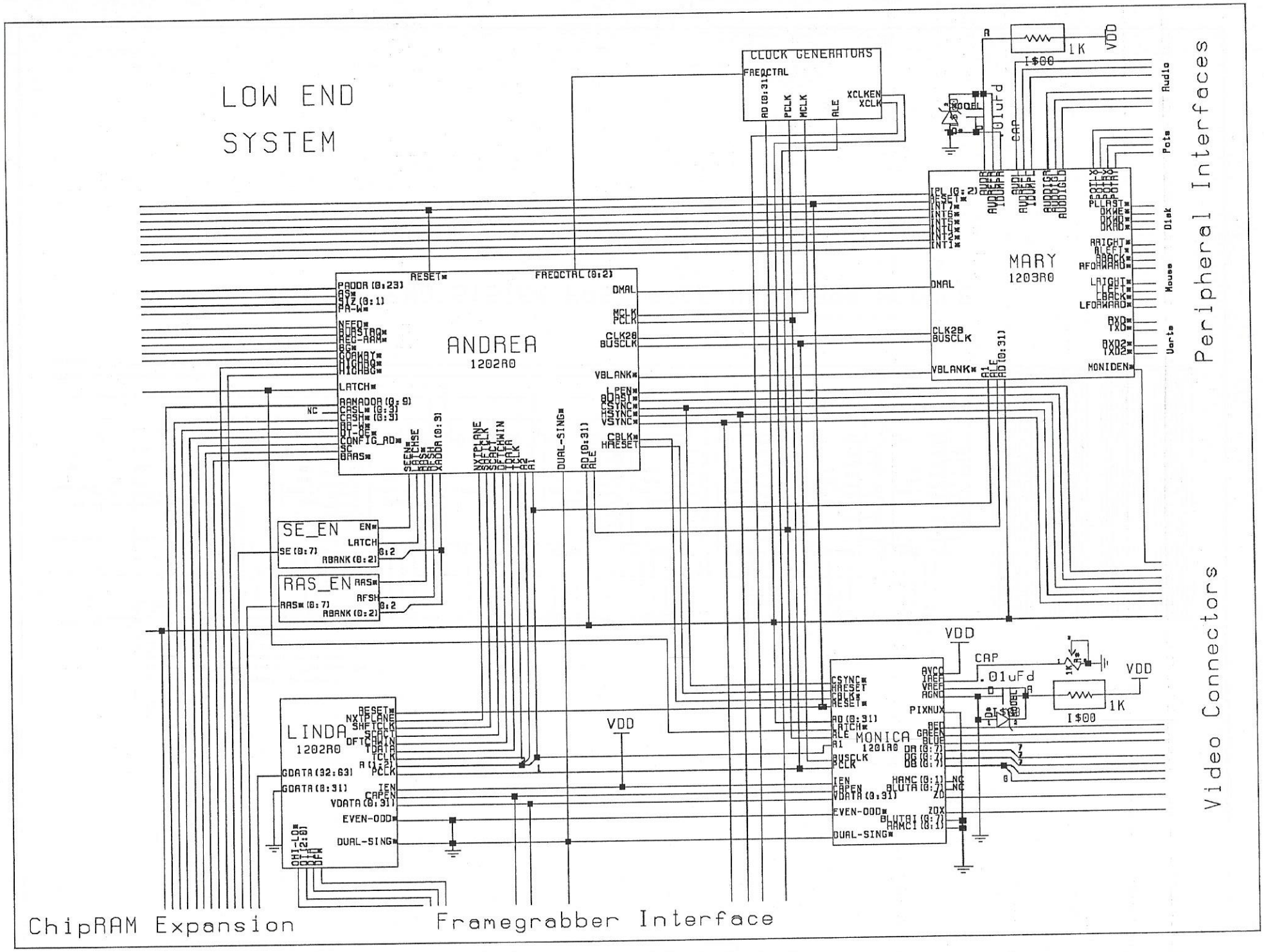

HALF CHUNKY FRAMEGRABBER MODE, LOW RESOLUTION, LOW END (SINGLE) SYSTEM.
ANDREA HORI. COUNT
DDFSTAT
DFTCHWIN
OHI/LO, OF(2:0), and OIR
DFW
Valid BPLnDAT on VDATA bus (CAPEN).
DWINDOW
valid RGB out
HALF CHUNKY FRAMEGRABBER MODE, LOW RESOLUTION, HIGH END (DUAL) SYSTEM.
ANDREA HORI. COUNT
DDFSTAT
DFTCHWIN
OHI/LO, OF(2:0), and OIR
DFW
Valid BPLnDAT on VDATA bus (CAPEN).
DWINDOW
valid RGB out
HALF CHUNKY FRAMEGRABBER MODE, HIGH RESOLUTION, LOW END (SINGLE) SYSTEM.
ANDREA HORI. COUNT
DDFSTAT
DFTCHWIN
OHI/LO, OF(2:0), and OIR
DFW
Valid BPLnDAT on VDATA bus (CAPEN).
DWINDOW
valid RGB out
HALF CHUNKY FRAMEGRABBER MODE, HIGH RESOLUTION, HIGH END (DUAL) SYSTEM.
ANDREA HORI. COUNT
DDFSTAT
DFTCHWIN
OHI/LO, OF(2:0), and OIR
DFW
Valid BPLnDAT on VDATA bus (CAPEN).
DWINDOW
valid RGB out

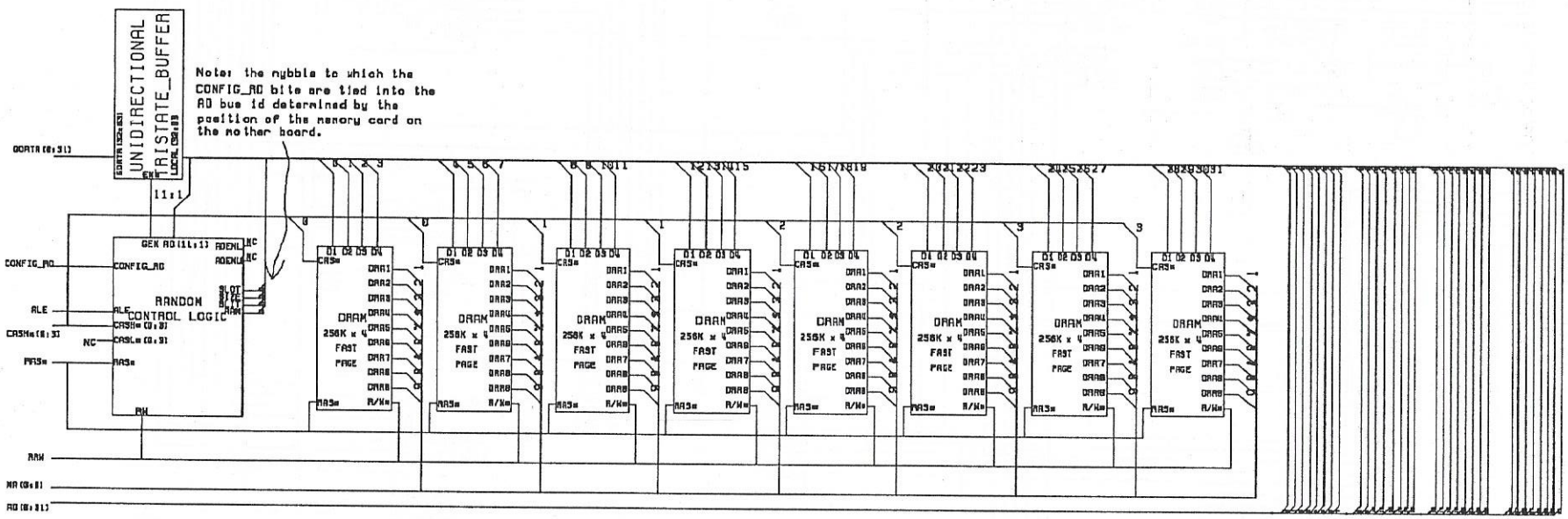
APPENDIX B

AC CHARACTERISTICS.

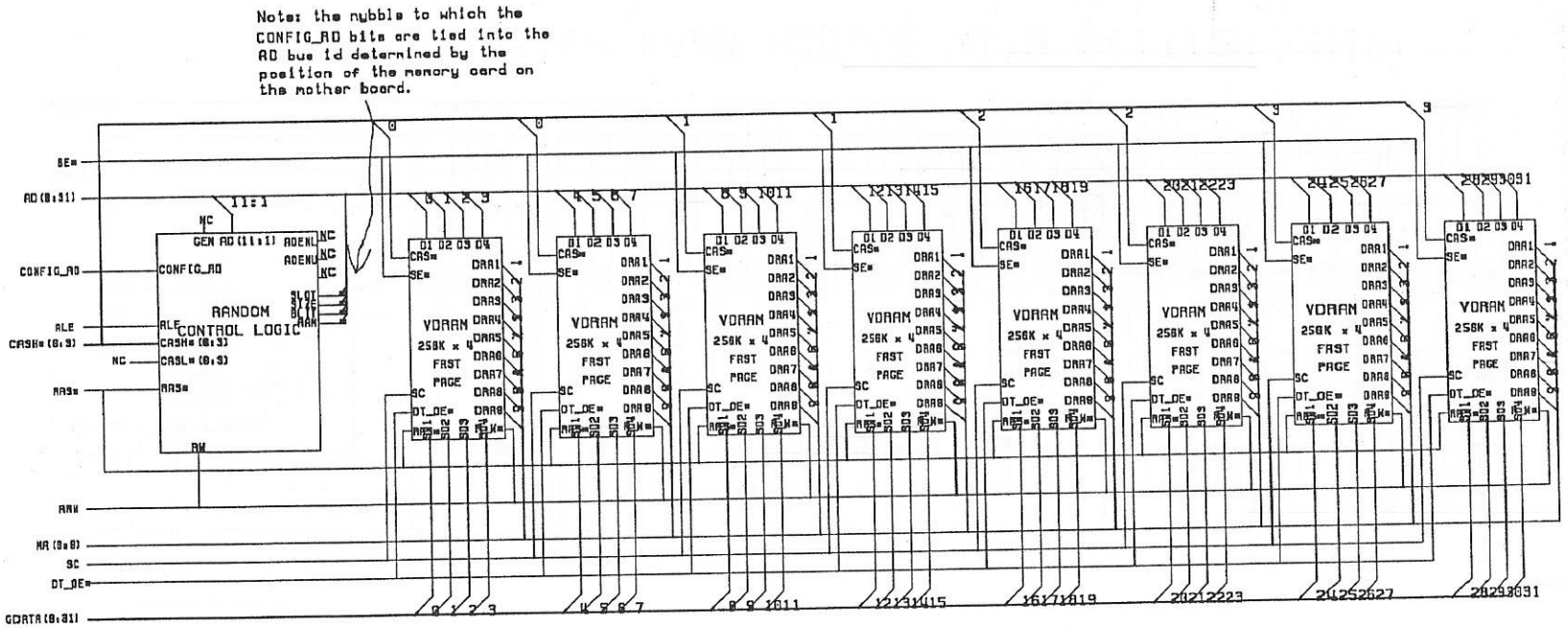
APPENDIX C

DETAILED BLOCK DIAGRAMS.

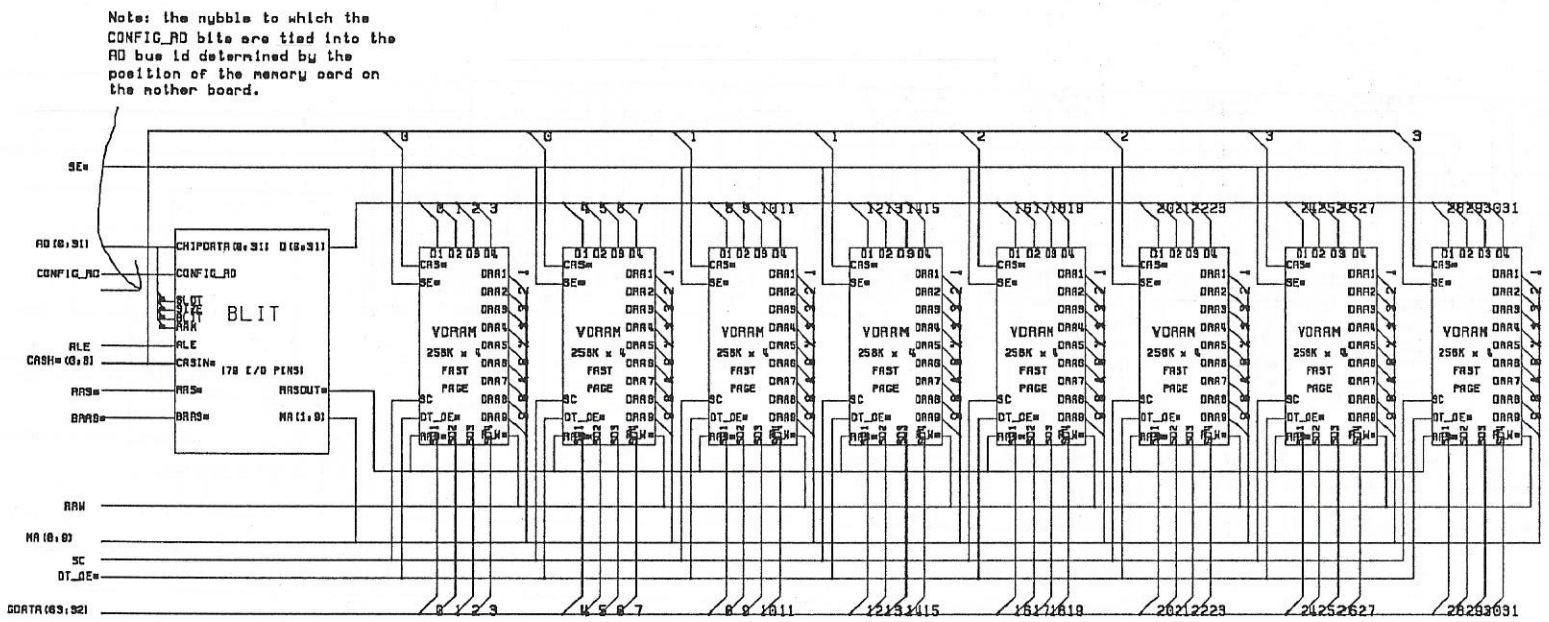

LOW END
SYSTEM
CLOCK GENERATORS
FREQCTRL
XCLKEN
XCLK
PCLK
MCLK
ALE
ANDREA
1202R0
RESET
FREQCTRL (0:2)
DMAL
MCLK
PCLK
CLK28
BUSCLK
VBLANK
LATCH
NC
DUAL-SING
MARY
1203R0
CLK28
BUSCLK
VBLANK
MONIDEN
SE_EN
SE (0:7)
RBANK (0:2)
RAS_EN
RAS (0:7)
RBANK (0:2)
LINDA
1202R0
GDATA (32:63)
GDATA (0:31)
VDATA (0:31)
EVEN-ODD
DUAL-SING
MONICA
1201A0
VDD
CAP
.01uFd
1K
I$00
Audio
Pots
Disk
Mouse
Ports
Peripheral Interfaces
Video Connectors
ChipRAM Expansion
Framegrabber Interface



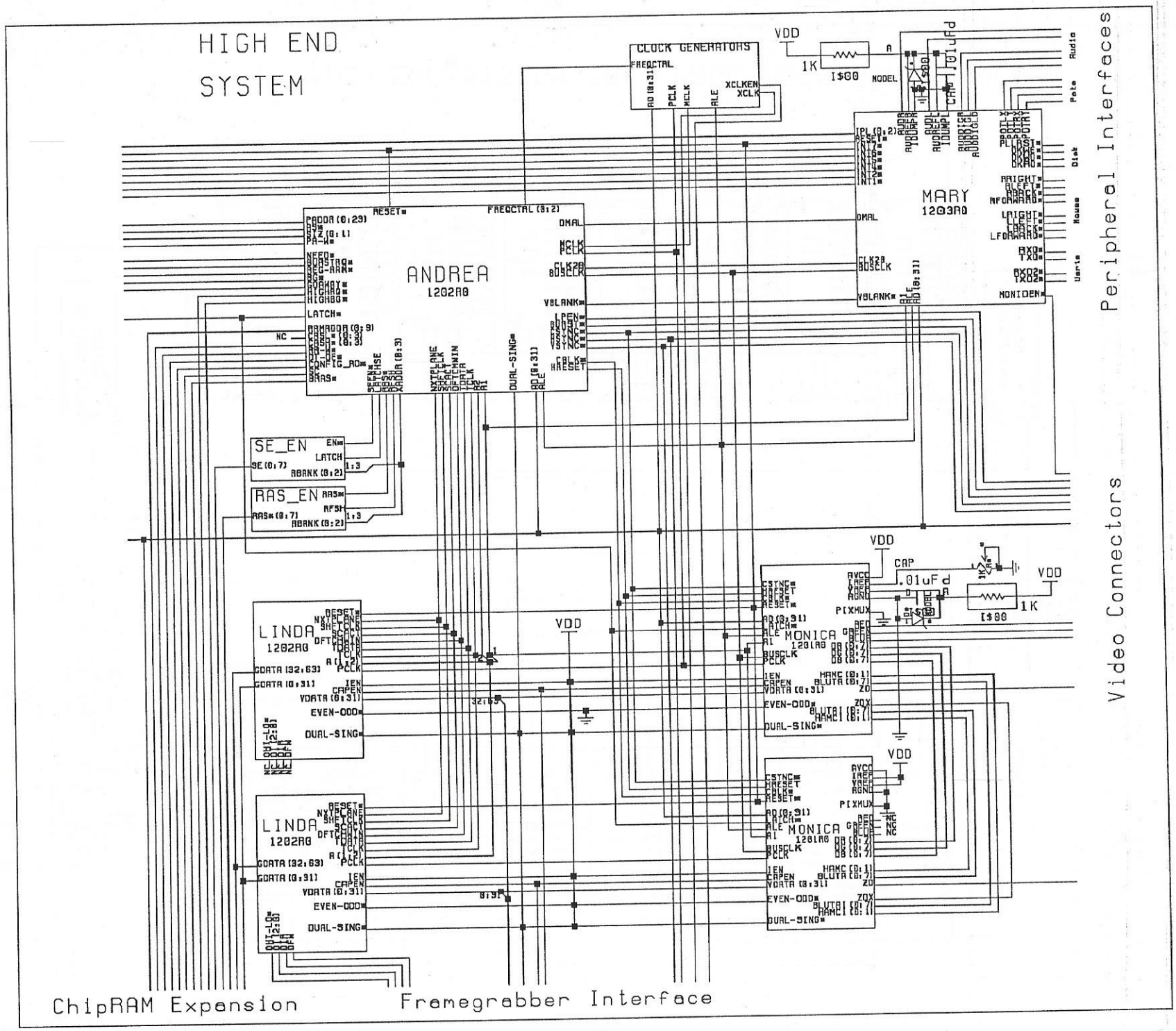
LOW END SYSTEM FAST PAGE MODE RAM MODULE

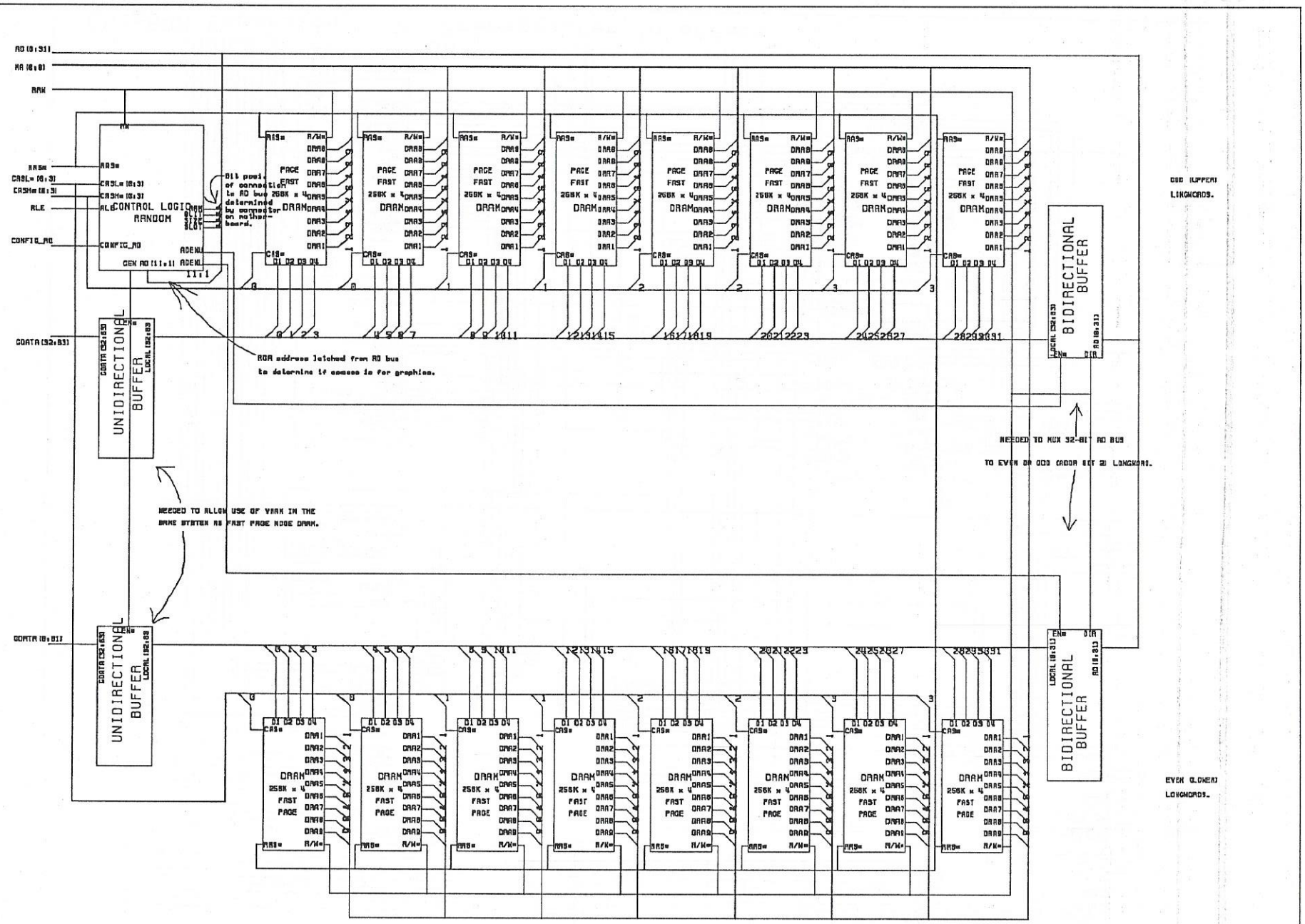


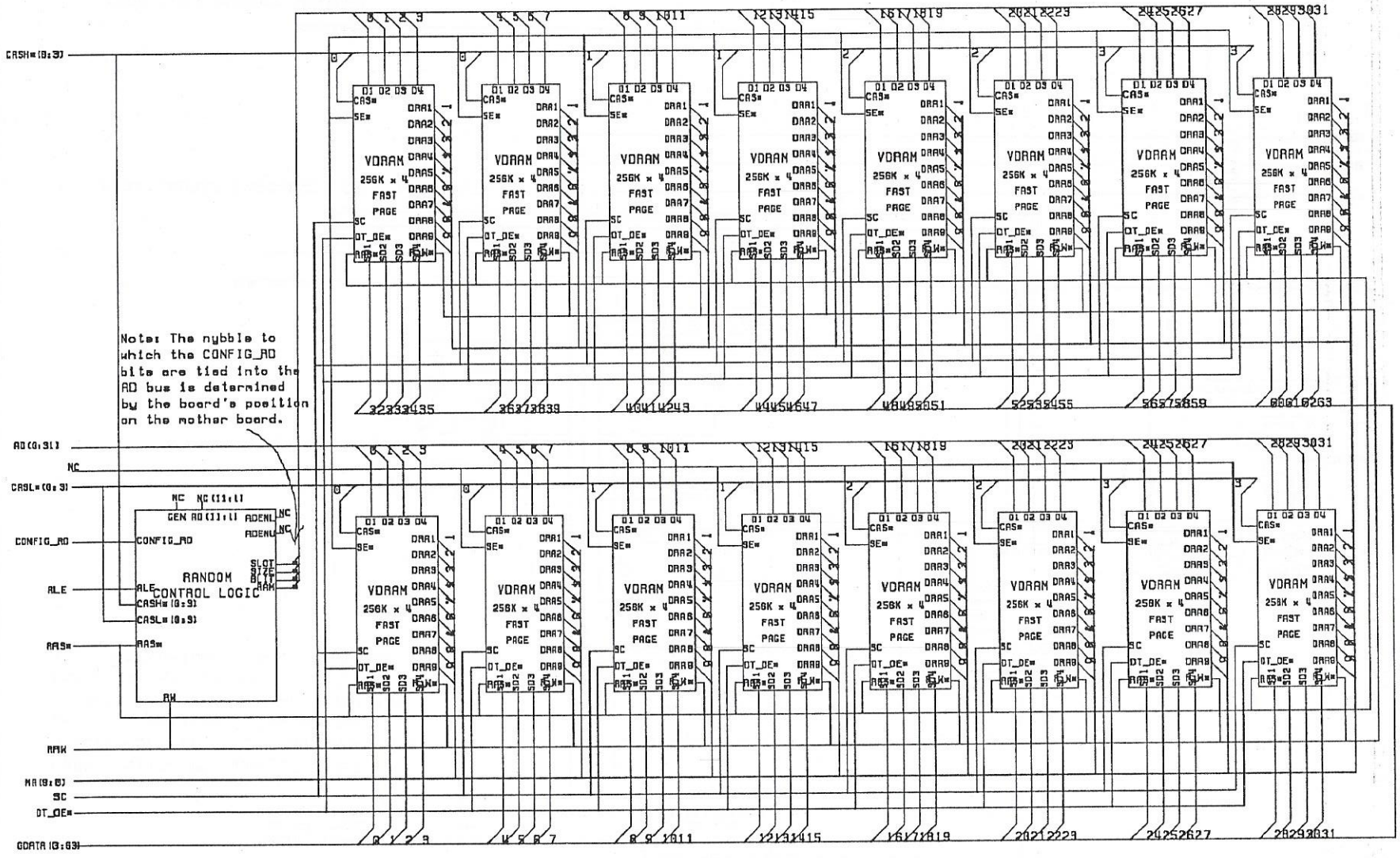
LOW END SYSTEM VRAM MODULE



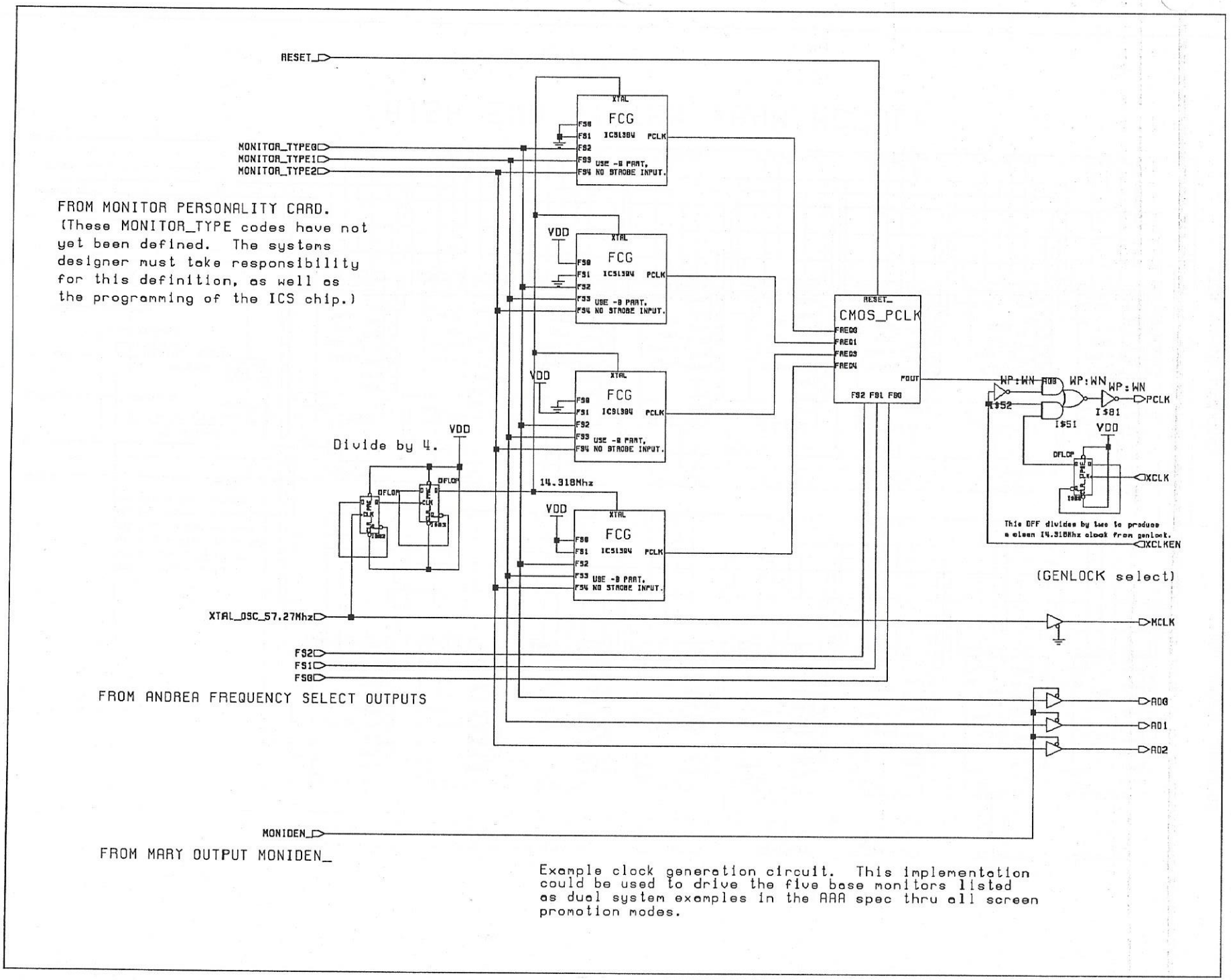
LOW END SYSTEM VRAM MODULE WITH BLITTER CHIP


HIGH END
SYSTEM
CLOCK GENERATORS
FREQCTRL
XCLKEN
XCLK
VDD
1K
I$00
.01uFd
CAP
ANDREA
1202A0
MARY
1203A0
MONICA
1201A0
LINDA
1202A0
SE_EN
RAS_EN
RESET
FREQCTRL (0:2)
DMAL
MCLK
PCLK
CLK28
BUSCLR
V8LANK
LATCH
DUAL-SING
MONIDEN
PIXMUX
EVEN-ODD
DUAL-SING
Peripheral Interfaces
Video Connectors
ChipRAM Expansion
Framegrabber Interface

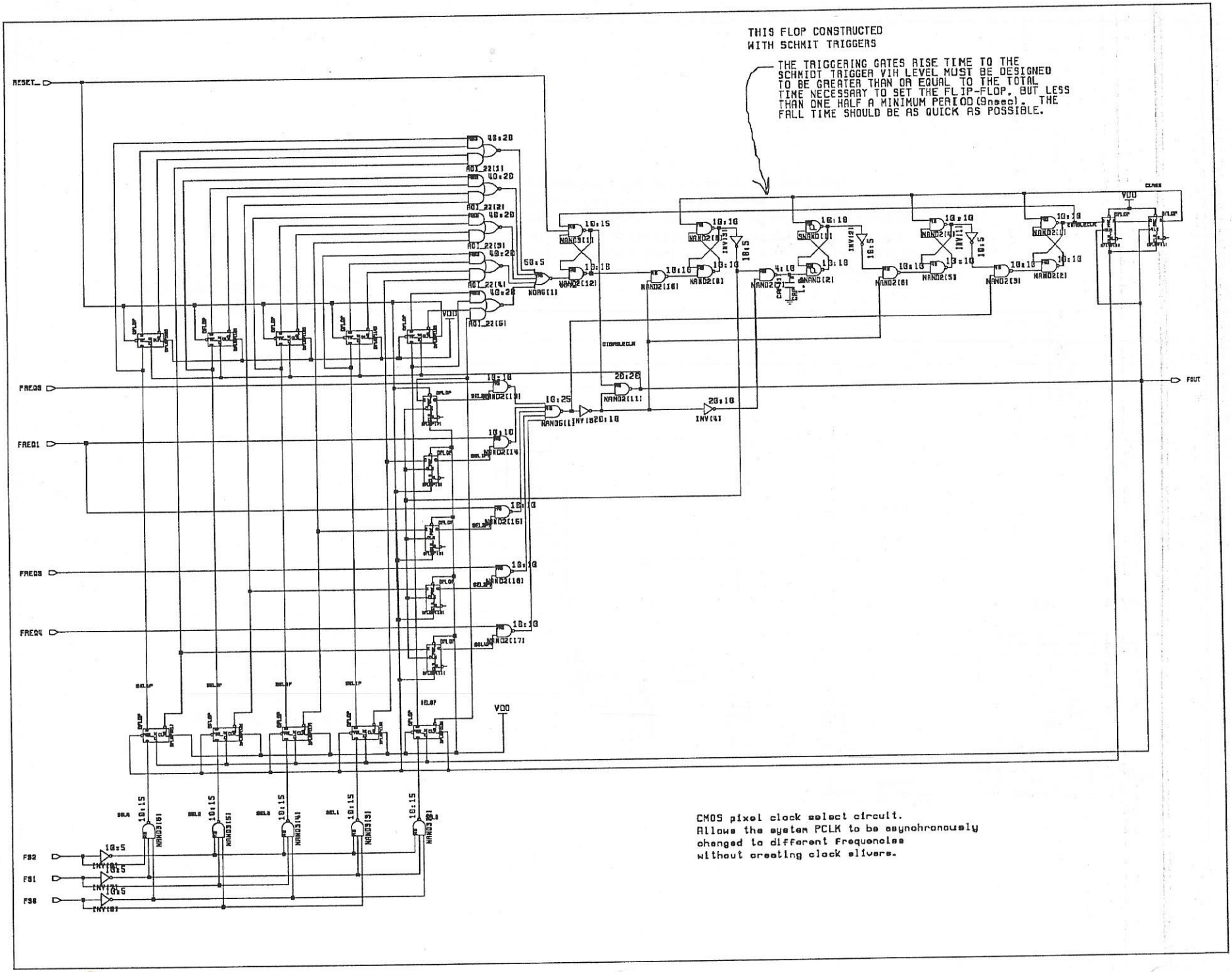

CONTROL LOGIC
RANDOM
DRAM
256K x 4
FAST
PAGE
UNIDIRECTIONAL
BUFFER
BIDIRECTIONAL
BUFFER
ODD (UPPER)
LONGWORDS.
EVEN (LOWER)
LONGWORDS.
NEEDED TO MUX 32-BIT AD BUS
NEEDED TO ALLOW USE OF VRAM IN THE
SAME SYSTEM AS FAST PAGE MODE DRAM.
HIGH END SYSTEM FAST PAGE MODE RAM MODULE

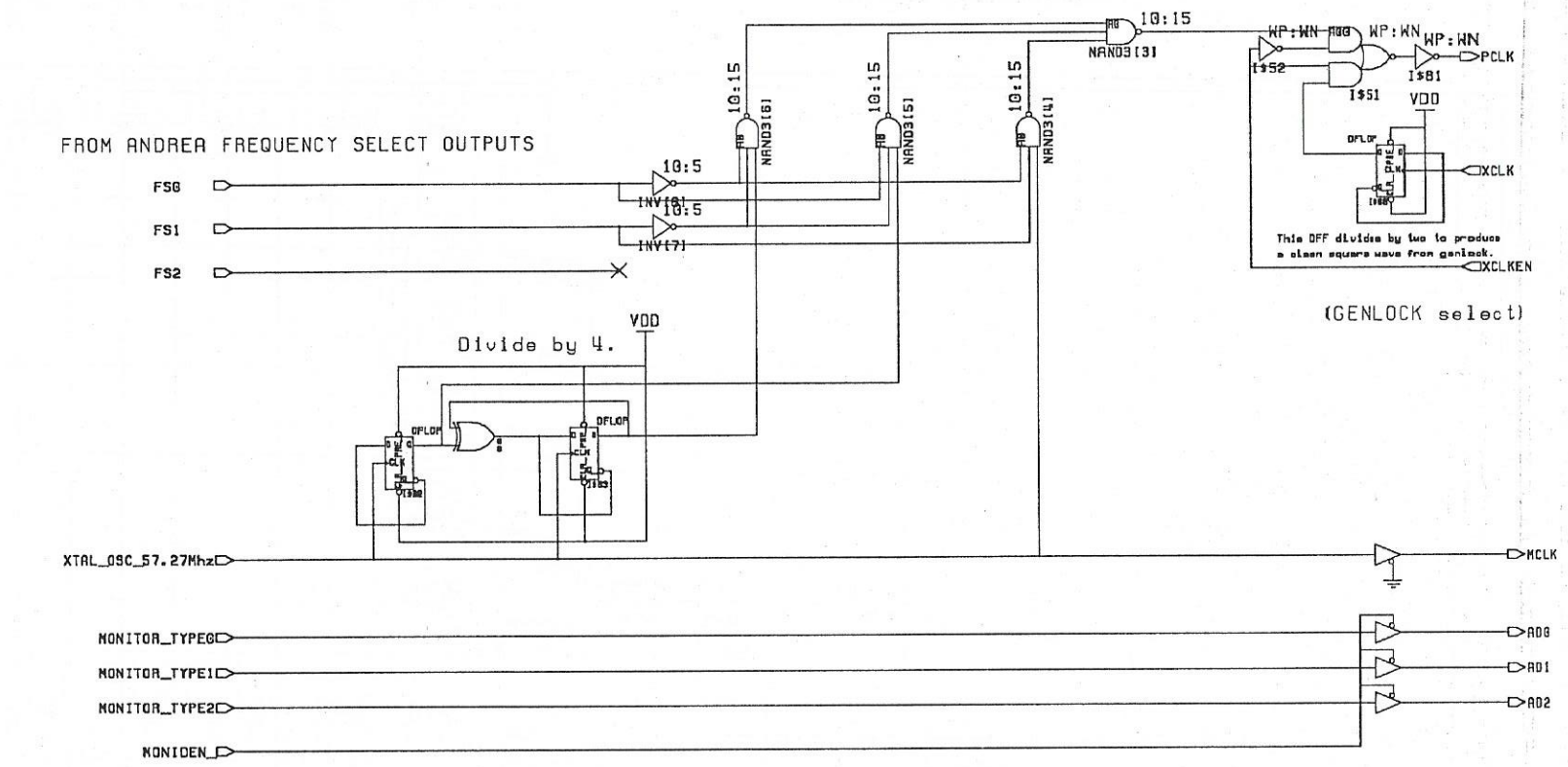


HIGH END SYSTEM VRAM MODULE



Example clock generation circuit. This implementation could be used to drive the five base monitors listed as dual system examples in the AAA spec thru all screen promotion modes.


THIS FLOP CONSTRUCTED
WITH SCHMIT TRIGGERS
THE TRIGGERING GATES RISE TIME TO THE SCHMIDT TRIGGER VIH LEVEL MUST BE DESIGNED TO BE GREATER THAN OR EQUAL TO THE TOTAL TIME NECESSARY TO SET THE FLIP-FLOP, BUT LESS THAN ONE HALF A MINIMUM PERIOD (9nsec). THE FALL TIME SHOULD BE AS QUICK AS POSSIBLE.
RESET_
FREQ0
FREQ1
FREQ3
FREQ4
FS2
FS1
FS0
FOUT
VDD
CMOS pixel clock select circuit.
Allows the system PCLK to be asynchronously
changed to different frequencies
without creating clock slivers.



Cheap clock generation circuit.
This circuit could be implemented to provide all display capabilities envisioned for the AA+ systems. Note since all pixel clock frequencies generated are a multiple of 14.318Mhz, no special synchronizing circuits are required.

APPENDIX D

OLD RGA REGISTERS NOT SUPPORTED IN NEW AMIGA CHIP SET.

The following group of registers have been eliminated in the AAA chip set.

->BEAMCON0 H 1DC W A Beam Counter control bits

Bit	Function
15	x
14	HARDDIS
13	x
12	VARVBEN
11	LOLDIS
10	CSCBEN
9	VARVSYEN
8	VARHSYEN
7	VARBEAMEN
6	DUAL
5	PAL
4	VARCSYEN

Most of the bits in this register are being eliminated because most of the old hardware stops are not present in the new chip set. Therefore, most of these control bits are irrelevent.

Due to the multimonitor control capability of the new chip set, the ECS DUAL mode has been abandonded.

The new chip set horizontal counters have the capacity to count pixels rather than the old style color burst clocks. Therefore, setting the HTOTAL register to 909 will provide the NTSC spec horizontal line rate of 227.5 color burst clocks. The Long line/Short line display technique is no longer required or used. All associated display data pipeline delay registers which were present on the old AGNUS and DENISE chips for longline displays have been removed.

VARVBEN= Use the comparator generated Vertical Blank (from VBSTRT,VBSTOP) to run the internal chip stuff- sending RGA signals to Denise, starting sprites, resetting light pen. It also disables the hard stop on the vertical display window.

LOLDIS= Disable long line/short line toggle. This is useful for DUAL mode where even multiples are wanted, or in any single display where this toggling is not desired.

CSCBEN= The variable composite sync comes out on the HSY* pin, and the variable composite blank comes out on the VSY* pin. The idea is to allow all the information to come out of the chip for a DUAL mode display. The normal monitor uses the normal composite sync, and the variable composite sync & blank come out the HSY* & VSY* pins. The bits VARVSYEN & VARHSYEN (below) have priority over this control bit.

VARVSYEN= comparator VSY -> VSY* pin. The variable VSY is set vertically on VSSTRT, reset vertically on VSSTOP, with the horizontal position for set & reset HSSTRT on short fields (all fields are short if LACE=0) and HCENTER on long fields (every other field if LACE=1)

VARHSYEN= comparator HSY -> HSY* pin. Set on HSSTRT value, reset on HSSTOP value.

VARBEAMEN= Enables the variable beam counter comparators to operate (allowing different beam counter total values) on the main horiz counter. It also disables hard display stops on both horizontal & vertical.

DUAL= Run the horizontal comparators with the alternate

horizontal beam counter, and starts the UHRES pointer chain with the reset of this counter rather than the normal one. This allows the UHRES pointers to come out more than once in a horizontal line, assuming there is some memory bandwidth left (It doesn't work in 640*400*4 interlace mode) Also, to keep the 2 displays synced, the horizontal line lengths should be multiples of each other. If you are amazingly clever, you might not need to do this.

VARCSYEN= enable CSY* from the variable decoders to come out the CSY* (VARCSY is set on HSSTRT match always, and also on HCENTER match when in vertical sync. It is reset on HSSTOP match when VSY*, and on both HBSTRT & HBSTOP matches during VSY. A reasonable composite can be generated by setting HCENTER half a horiz line from HSSTRT, and HBSTOP at (HSSTOP-HSSTRT) before HCENTER, with HBSTRT at (HSSTOP-HSSTRT) before HSSTRT.

->BPLCON0 h 100 W A D Bit plane control reg.(misc control bits)

UHRES= ultrahi res enables the UHRES pointers (for 1k*1k) (also needs bits in DMACON) (hires chips only) Disables hard stops for vert., horiz. display windows

BPLHWRM= Swaps the polarity of ARW* when the BPLHDAT comes out so that external devices can detect the RGA and put things into memory. (hires chips only)

SPRHWRM= Same as BPLHWRM, but with SPRHDAT.(hires chips only)

SHRES= superhires (640*400 noninterlace) sets the bit plane control for this- Doubles speed of output of a given bitplane over HRES. (hires chips only) 2 bitplanes maximum. Collsions don't apply, and priority is simplified in this mode. If priority is less than 4, the 1 available sprite has priority. If >=4, the sprite & bitplane bits are XORed. Disables hard stops in vert., horiz. display windows

The four bits shown in this register have been eliminated. These bits support ultra hires display mode. This display mode is not supported in the new chip set. The following registers have been eliminated for this reason.

->BPLHDAT H 07A W ext logic UHRES bit plane pointer identifier

->BPLHMOD H 1E6 W A UHRES bit plane modulo

This is the number (sign extended) that is added to the UHRES bit plane pointer (BPLHPTL,H) every line, and then another 2 is added, just like the other modulos.

->BPLHSTOP H 1D6 W A UHRES bit plane vertical stop

This controls the line when the data fetch stops for the BPLHPTH,L pointers. V10-V0 on DB10-0.

->BPLHSTRT H 1D4 W A UHRES bit plane vertical start

This controls the line when the data fetch starts for the BPLHPTH,L pointers. V10-V0 on DB10-0.

->BPLHPTH H 1EC W A UHRES (VRAM) bit plane pntr (High 5 bits)

->BPLHPTL H 1EE W A UHRES (VRAM) bit plane pntr (Low 15 bits)

When UHRES is enabled, this pointer comes out on the 2nd 'free' cycle after the start of each horiz. line. Its modulo is added every time it comes out. 'Free' means priority above the copper and below the fixed stuff (audio,sprites...) BPLHDAT comes out as an identifier on the RGA lines when the pointer address is valid so that external detectors can use this to do the special cycle for the VRAMs. The SHRHDAT gets the first and third free cycles.

->SPRHDAT H 078 W ext logic UHRES sprite identifier & data
This identifies the cycle when this pointer address is on the bus accessing the memory.

->SPRHPTH H 1E8 W A UHRES sprite pointer (High 5 bits)
->SPRHPTL H 1EA W A UHRES sprite pointer (Low 15 bits)
This pointer is activated in the 1st & 3rd 'free' cycles (see BPLHPTH,L) after horiz line start. It increments for the next line.

->SPRHSTOP H 1D2 W A UHRES sprite vertical display stop
->SPRHSTRT H 1D0 W A UHRES sprite vertical display start

Bit#	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
	x	x	x	x	x	v10	v9	v8	v7	v6	v5	v4	v3	v2	v1	v0

APPENDIX E

RGA REGISTER NAMES.

REGNAME.HR(10-Apr-92) Advanced AMIGA chip regname

&=Register used by DMA channel only.
%=Register used by DMA channel usually, processors sometimes.
+=Address register pair. Low word uses DB1-DB15, High word DB0-DB4.
~=Address not writable by the Coprocessor unless COPCON bit 1 is set true.
h=new for HiRes chip set
A=ANDREA chip, M=Monica chip, P=Mary chip
W=Write, R=Read,
DR=DMA channel read. This is a DMA data transfer to RAM, from either the Disk or from the Blitter.
PTL,PTH=20 bit word Pointer that addresses DMA data. Must be reloaded by a processor before use (Vertical blank for Bit Plane and Sprite pointers, and prior to starting the Blitter for Blitter pointers).(old chips- 18 bits)
PTX=22 bit long word pointer that addresses DMA data.
LCL,LCH=20 bit Location (word starting address) of DMA data. Used to automatically restart pointers, such as the Coprocessor program counter (during vertical blank), and the Audio sample counter (whenever the audio length count is finished) (Old chips- 18 bits)
LCX=22 bit location (long word starting address of DMA data
MOD=15 bit Modulo. A number that is automatically added to the memory address at the end of each line to generate the address for the beginning of the next line. This allows the Blitter (or the Display Window) to operate on (or display) a window of data that is smaller than the actual picture in memory. (memory map) Uses 15 bits, plus sign extend.

Old RGA Addresses

NAME		ADD	R/W	CHIP		FUNCTION
BLTDDAT	&	~000	DR	A		Blitter dest. early read (dummy address)
DMACONR		~002	R	A		DMA control (and blitter status) read
VPOSR		~004	R	A		Read Vert most sig. bits (and frame flop)
VHPOSR		~006	R	A		Read Vert and horiz Position of beam
DSKDATR	&	~008	DR		P	Disk data early read (dummy address)
JOY0DAT		~00A	R		P	Joystick-mouse 0 data (vert,horiz)
JOY1DAT		~00C	R		P	Joystick-mouse 1 data (vert,horiz)
CLXDAT		~00E	R	M		Collision data reg.(Read and clear)
ADKCONR		~010	R		P	Audio, disk control register read
POT0DAT		~012	R		P	Pot counter pair 0 data (vert,horiz)
POT1DAT		~014	R		P	Pot counter pair 1 data (vert,horiz)
POTINP		~016	R		P	Pot pin data read
SERDATR		~018	R		P	Serial Port Data and Status read
DSKBYTR		~01A	R		P	Disk Data byte and status read
INTENAR		~01C	R		P	Interrupt Enable bits Read
INTREQR		~01E	R		P	Interrupt Request bits read
DSKPTH	+	~020	W	A		Disk pointer (High 5 bits)
DSKPTL	+	~022	W	A		Disk pointer (Low 15 bits)
DSKLEN		~024	W		P	Disk length
DSKDAT	&	~026	W		P	Disk DMA Data write
REFPTR	&	~028	W	A		Refresh pointer
VPOSW		~02A	W	A		Write Vert most sig. bits (and frame flop)
VHPOSW		~02C	W	A		Write Vert and horiz Position of beam
COPCON		~02E	W	A		Coprocessor control register (CDANG)
SERDAT		~030	W		P	Serial Port Data and stop bits write
SERPER		~032	W		P	Serial Port Period and control
POTGO		~034	W		P	Pot count start,pot pin drive enable and data
JOYTEST		~036	W		P	Write to all 4 Joystick-mouse counters at once.
		~038				

		~03A					
		~03C					
		~03E					
BLTCON0		~040	W	A			Blitter control register 0
BLTCON1		~042	W	A			Blitter control register 1
BLTAFWM		~044	W	A			Blitter first word mask for source A
BLTALWM		~046	W	A			Blitter last word mask for source A
BLTCPTH	+	~048	W	A			Blitter Pointer to source C (High 5 bits)
BLTCPTL	+	~04A	W	A			Blitter Pointer to source C (Low 15 bits)
BLTBPTH	+	~04C	W	A			Blitter pointer to source B (High 5 bits)
BLTBPTL	+	~04E	W	A			Blitter pointer to source B (Low 15 bits)
BLTAPTH	+	~050	W	A			Blitter Pointer to source A (High 5 bits)
BLTAPTL	+	~052	W	A			Blitter Pointer to source A (Low 15 bits)
BLTDPTH	+	~054	W	A			Blitter Pointer to destn. D (High 5 bits)
BLTDPTL	+	~056	W	A			Blitter Pointer to destn. D (Low 15 bits)
BLTSIZE		~058	W	A			Blitter start and size (window width, height)
BLTCON0L	h	~05A	W	A			Blitter control 0 lower 8 bits(minterms)
BLTSIZV	h	~05C	W	A			Blitter V size (for 15 bit vert size)
BLTSIZH	h	~05E	W	A			Blitter H size & start (for 11 bit H size)
BLTCMOD		~060	W	A			Blitter Modulo for source C
BLTBMOD		~062	W	A			Blitter Modulo for source B
BLTAMOD		~064	W	A			Blitter Modulo for source A
BLTDMOD		~066	W	A			Blitter Modulo for destn. D
		~068					do not use
		~06A					do not use
		~06C					do not use
		~06E					do not use
BLTCDAT	&	~070	W	A			Blitter source C data reg
BLTBDAT	&	~072	W	A			Blitter source B data reg
BLTADAT	&	~074	W	A			Blitter source A data reg
		~076					do not use
		~078					do not use
		~07A					
MONICAID	h	~07C	R		M		Chip Revision level for MONICA (video out chip)
DSKSYNC		~07E	W			P	Disk sync pattern register for disk read.
COP1LCH	+	080	W	A			Coprocessor first location reg (High 5 bits)
COP1LCL	+	082	W	A			Coprocessor first location reg. (Low 15 bits)
COP2LCH	+	084	W	A			Coprocessor second location reg. (High 5 bits)
COP2LCL	+	086	W	A			Coprocessor second location reg(Low 15 bits)
COPJMP1		088	S	A			Coprocessor restart at first location
COPJMP2		08A	S	A			Coprocessor restart at second location
COPINS		08C	W	A			Coprocessor inst. fetch identify
DIWSTRT		08E	W	A			Display Window Start (upper left vert-hor pos)
DIWSTOP		090	W	A			Display Window Stop (lower right vert-hor pos)
DDFSTRT		092	W	A			Display bit plane data fetch start(hor pos)
DDFSTOP		094	W	A			Display bit plane data fetch stop(hor pos)
DMACON		096	W	A		P	DMA control write(clear or set)
CLXCON		098	W		M		Collision control
INTENA		09A	W			P	Interrupt Enable bits (clear or set bits)
INTREQ		09C	W			P	Interrupt Request bits (clear or set bits)
ADKCON		09E	W			P	Audio, Disk, UART, Control
AUD0LCH	+	0A0	W	A			Audio channel 0 location (High 5 bits)
AUD0LCL	+	0A2	W	A			Audio channel 0 location (Low 15 bits)
AUD0LEN		0A4	W			P	Audio Channel 0 length
AUD0PER		0A6	W			P	Audio channel 0 Period
AUD0VOL		0A8	W			P	Audio Channel 0 Volume
AUD0DAT	%	0AA	W			P	Audio channel 0 Data
		0AC					
		0AE					
AUD1LCH	+	0B0	W	A			Audio channel 1 location (High 5 bits)
AUD1LCL	+	0B2	W	A			Audio channel 1 location (Low 15 bits)
AUD1LEN		0B4	W			P	Audio Channel 1 length

AUD1PER		0B6	W		P		Audio channel 1 Period
AUD1VOL		0B8	W		P		Audio Channel 1 Volume
AUD1DAT	%	0BA	W		P		Audio channel 1 Data
		0BC					
		0BE					
AUD2LCH	+	0C0	W	A			Audio channel 2 location (High 5 bits)
AUD2LCL	+	0C2	W	A			Audio channel 2 location (Low 15 bits)
AUD2LEN		0C4	W		P		Audio Channel 2 length
AUD2PER		0C6	W		P		Audio channel 2 Period
AUD2VOL		0C8	W		P		Audio Channel 2 Volume
AUD2DAT	%	0CA	W		P		Audio channel 2 Data
		0CC					
		0CE					
AUD3LCH	+	0D0	W	A			Audio channel 3 location (High 5 bits)
AUD3LCL	+	0D2	W	A			Audio channel 3 location (Low 15 bits)
AUD3LEN		0D4	W		P		Audio Channel 3 length
AUD3PER		0D6	W		P		Audio channel 3 Period
AUD3VOL		0D8	W		P		Audio Channel 3 Volume
AUD3DAT	%	0DA	W		P		Audio channel 3 Data
		0DC					
		0DE					
BPL1PTH	+	0E0	W	A			Bit plane 1 pointer (High 5 bits)
BPL1PTL	+	0E2	W	A			Bit plane 1 pointer (Low 15 bits)
BPL2PTH	+	0E4	W	A			Bit plane 2 pointer (High 5 bits)
BPL2PTL	+	0E6	W	A			Bit plane 2 pointer (Low 15 bits)
BPL3PTH	+	0E8	W	A			Bit plane 3 pointer (High 5 bits)
BPL3PTL	+	0EA	W	A			Bit plane 3 pointer (Low 15 bits)
BPL4PTH	+	0EC	W	A			Bit plane 4 pointer (High 5 bits)
BPL4PTL	+	0EE	W	A			Bit plane 4 pointer (Low 15 bits)
BPL5PTH	+	0F0	W	A			Bit plane 5 pointer (High 5 bits)
BPL5PTL	+	0F2	W	A			Bit plane 5 pointer (Low 15 bits)
BPL6PTH	+	0F4	W	A			Bit plane 6 pointer (High 5 bits)
BPL6PTL	+	0F6	W	A			Bit plane 6 pointer (Low 15 bits)
BPLCON0		100	W	A	M		Bit plane control register(misc control bits)
BPLCON1		102	W		M		Bit plane control reg (scroll value PF1, PF2)
BPLCON2		104	W		M		Bit plane control reg (priority control)
BPLCON3		106	W		M		Bit plane control reg (enhanced features)
BPL1MOD		108	W	A			Bit plane modulo (odd planes)
BPL2MOD		10A	W	A			Bit Plane modulo (even planes)
BPLCON4		10C	W		M		Bit plane control reg (enhanced features)
		10E					
BPL1DAT	&	110	W		M		Bit plane 1 data (Parallel to serial convert)
BPL2DAT	&	112	W		M		Bit plane 2 data (Parallel to serial convert)
BPL3DAT	&	114	W		M		Bit plane 3 data (Parallel to serial convert)
BPL4DAT	&	116	W		M		Bit plane 4 data (Parallel to serial convert)
BPL5DAT	&	118	W		M		Bit plane 5 data (Parallel to serial convert)
BPL6DAT	&	11A	W		M		Bit plane 6 data (Parallel to serial convert)
		11C					
		11E					
SPR0PTH	+	120	W	A			Sprite 0 pointer (High 5 bits)
SPR0PTL	+	122	W	A			Sprite 0 pointer (Low 15 bits)
SPR1PTH	+	124	W	A			Sprite 1 pointer (High 5 bits)
SPR1PTL	+	126	W	A			Sprite 1 pointer (Low 15 bits)
SPR2PTH	+	128	W	A			Sprite 2 pointer (High 5 bits)
SPR2PTL	+	12A	W	A			Sprite 2 pointer (Low 15 bits)
SPR3PTH	+	12C	W	A			Sprite 3 pointer (High 5 bits)
SPR3PTL	+	12E	W	A			Sprite 3 pointer (Low 15 bits)
SPR4PTH	+	130	W	A			Sprite 4 pointer (High 5 bits)
SPR4PTL	+	132	W	A			Sprite 4 pointer (Low 15 bits)
SPR5PTH	+	134	W	A			Sprite 5 pointer (High 5 bits)
SPR5PTL	+	136	W	A			Sprite 5 pointer (Low 15 bits)
SPR6PTH	+	138	W	A			Sprite 6 pointer (High 5 bits)

SPR6PTL	+	13A	W	A		Sprite 6 pointer (Low 15 bits)
SPR7PTH	+	13C	W	A		Sprite 7 pointer (High 5 bits)
SPR7PTL	+	13E	W	A		Sprite 7 pointer (Low 15 bits)
SPR0POS	%	140	W	A	M	Sprite 0 Vert-Horiz start position data
SPR0CTL	%	142	W	A	M	Sprite 0 position and control data
SPR0DATA	&	144	W		M	Sprite 0 image data register A
SPR0DATB	&	146	W		M	Sprite 0 image data register B
SPR1POS	%	148	W	A	M	Sprite 1 Vert-Horiz start position data
SPR1CTL	%	14A	W	A	M	Sprite 1 position and control data
SPR1DATA	&	14C	W		M	Sprite 1 image data register A
SPR1DATB	&	14E	W		M	Sprite 1 image data register B
SPR2POS	%	150	W	A	M	Sprite 2 Vert-Horiz start position data
SPR2CTL	%	152	W	A	M	Sprite 2 position and control data
SPR2DATA	&	154	W		M	Sprite 2 image data register A
SPR2DATB	&	156	W		M	Sprite 2 image data register B
SPR3POS	%	158	W	A	M	Sprite 3 Vert-Horiz start position data
SPR3CTL	%	15A	W	A	M	Sprite 3 position and control data
SPR3DATA	&	15C	W		M	Sprite 3 image data register A
SPR3DATB	&	15E	W		M	Sprite 3 image data register B
SPR4POS	%	160	W	A	M	Sprite 4 Vert-Horiz start position data
SPR4CTL	%	162	W	A	M	Sprite 4 position and control data
SPR4DATA	&	164	W		M	Sprite 4 image data register A
SPR4DATB	&	166	W		M	Sprite 4 image data register B
SPR5POS	%	168	W	A	M	Sprite 5 Vert-Horiz start position data
SPR5CTL	%	16A	W	A	M	Sprite 5 position and control data
SPR5DATA	&	16C	W		M	Sprite 5 image data register A
SPR5DATB	&	16E	W		M	Sprite 5 image data register B
SPR6POS	%	170	W	A	M	Sprite 6 Vert-Horiz start position data
SPR6CTL	%	172	W	A	M	Sprite 6 position and control data
SPR6DATA	&	174	W		M	Sprite 6 image data register A
SPR6DATB	&	176	W		M	Sprite 6 image data register B
SPR7POS	%	178	W	A	M	Sprite 7 Vert-Horiz start position data
SPR7CTL	%	17A	W	A	M	Sprite 7 position and control data
SPR7DATA	&	17C	W		M	Sprite 7 image data register A
SPR7DATB	&	17E	W		M	Sprite 7 image data register B
COLOR00		180	W		M	Color table 00
COLOR01		182	W		M	Color table 01
COLOR02		184	W		M	Color table 02
COLOR03		186	W		M	Color table 03
COLOR04		188	W		M	Color table 04
COLOR05		18A	W		M	Color table 05
COLOR06		18C	W		M	Color table 06
COLOR07		18E	W		M	Color table 07
COLOR08		190	W		M	Color table 08
COLOR09		192	W		M	Color table 09
COLOR10		194	W		M	Color table 10
COLOR11		196	W		M	Color table 11
COLOR12		198	W		M	Color table 12
COLOR13		19A	W		M	Color table 13
COLOR14		19C	W		M	Color table 14
COLOR15		19E	W		M	Color table 15
COLOR16		1A0	W		M	Color table 16
COLOR17		1A2	W		M	Color table 17
COLOR18		1A4	W		M	Color table 18
COLOR19		1A6	W		M	Color table 19
COLOR20		1A8	W		M	Color table 20
COLOR21		1AA	W		M	Color table 21
COLOR22		1AC	W		M	Color table 22
COLOR23		1AE	W		M	Color table 23
COLOR24		1B0	W		M	Color table 24
COLOR25		1B2	W		M	Color table 25
COLOR26		1B4	W		M	Color table 26

COLOR27		1B6	W		M	Color table 27
COLOR28		1B8	W		M	Color table 28
COLOR29		1BA	W		M	Color table 29
COLOR30		1BC	W		M	Color table 30
COLOR31		1BE	W		M	Color table 31
HTOTAL	h	1C0	W	A		Highest number count in horiz line (VARBEAMEN=1)
HSSTOP	h	1C2	W	A		Horiz line pos for HSYNC stop
HBSTRT	h	1C4	W	A		Horiz line pos for HBLANK start
HBSTOP	h	1C6	W	A		Horiz line pos for HBLANK stop
VTOTAL	h	1C8	W	A		Highest numbered Vertical line (VARBEAMEN=1)
VSSTOP	h	1CA	W	A		Vert. line pos for VSYNC stop
VBSTRT	h	1CC	W	A		Vert line for VBLANK start
VBSTOP	h	1CE	W	A		Vert line for VBLANK stop
		1D0				
		1D2				
		1D4				
		1D6				
HHPOSW	h	1D8	W	A		DUAL mode hires H beam counter write
HHPOSR	h	1DA	R	A		DUAL mode hires H beam counter read
BEAMCON0	h	1DC	W	A		Beam counter control register(SHRES,UHRES,PAL)
HSSTRT	h	1DE	W	A		Horizontal Sync start (VARHSY)
VSSTRT	h	1E0	W	A		Vertical Sync start (VARVSY)
HCENTER	h	1E2	W	A		Horizontal position for Vsync on interlace
DIWHIGH	h	~1E4	W	A	M	Display window- upper bits for start,stop
		1E6				
		1E8				
		1EA				
		1EC				
		1EE				
AUD4DAT	n	%1F0	W		P	Audio Channel 4 16 bit data register
AUD5DAT	n	%1F2	W		P	Audio Channel 5 16 bit data register
AUD6DAT	n	%1F4	W		P	Audio Channel 6 16 bit data register
AUD7DAT	n	%1F6	W		P	Audio Channel 7 16 bit data register
RESERVED		1F8-1FC				
NO-OP(NULL)		1FE				Usually indicates refresh cycle, or processor RAM access cycles.

32 Bit Control Registers

NAME	ADD	R/W	CHIP		FUNCTION
AUDLEFTR	~200	R		P	Read left audio channel
VHPOSRX	~204	R	A		Extended Read Vert and Hori Position of beam.
DSKDATRX	&~208	DR		P	Disk DMA Data read
DMACONRX	~20C	R	A		Extended DMA Control read
ADKCONRX	~210	R		P	Audio, Disk, UART, Control read
DSKBYTRX	~214	R		P	Disk DATA word and status read
INTREQRX	~218	R		P	Interrupt Request bits (read)
INTENARX	~21C	R		P	Interrupt Enable bits (read)
DSKPLLP	~220	W		P	Disk Phase Lock Loop Phase Adjust.
DSKPLLF	~224	W		P	Disk Phase Lock Loop Frequency Adjust.
DSKDATX	&~228	W		P	Disk DMA Data write
VHPOSWX	~22C	W	A		Extended Write Vert and Hori Position of beam.
DSKPLLR	~230	R		P	Disk Phase Lock Loop status read register(test).
DSKPTX	~234	W	A		Extended Disk Pointer (address bits 23-0)
BLTLLENX	~238	R	A		Blitter line length read register.
BLTPATRNR	~23C	R	A		Blitter line draw pattern read register.
	~240				
BLTPMSKX	~244	W	A		Plane Mask.
	~248				
BLTASRCPTX	~24C	W	A		Bit Map A Pointer.

BLTBSRCPTX	~250	W	A			Bit Map B Pointer.
BLTDSRCPTX	~258	W	A			Bit Map D Pointer.
BLTASRCWDX	~25C	W	A			Bit Map A Width.
BLTBSRCWDX	~260	W	A			Bit Map B Width.
BLTDSRCWDX	~264	W	A			Bit Map D Width.
BLTSIZEX	~268	W	A			Blit Size.
BLTAORGX	~26C	W	A			Blit A Origin.
BLTBORGX	~270	W	A			Blit B Origin.
BLTDORGX	~274	W	A			Blit D Origin.
BLTFUNCX	~278	W	A			Blit Function and Start Register.
DSKSYNCX	~27C	W			P	Longword disk sync pattern for disk read.
COP1LC	280	W	A			Extended coprocessor first location reg.
COP2LC	284	W	A			Extended coprocessor second location reg.
	288					
ADKCONX	28C	W			P	Audio, Disk, UART, Control write
DMACONX	290	W	A			Extended DMA control write
INTENAX	294	W			P	Interrupt Enable bits (clear or set)
CLXCONX	298	W		M		Extended Collision Control
INTREQX	29C	W			P	Interrupt Request bits (clear or set)
AUD0LCX	2A0	W			P	Extended Audio channel 0 backup pointer
AUD0LENX	2A4	W			P	Extended Audio channel 0 length
AUD0PERX	2A8	W			P	Extended Audio channel 0 period
AUD0VOLX	2AC	W			P	Extended Audio channel 0 volume
AUD0DATX	% 2B0	W			P	Extended Audio channel 0 data
AUD0CTL	2B4	W			P	Audio channel 0 control bits
AUD1LCX	2B8	W			P	Extended Audio channel 1 backup pointer
AUD1LENX	2BC	W			P	Extended Audio channel 1 length
AUD1PERX	2C0	W			P	Extended Audio channel 1 period
AUD1VOLX	2C4	W			P	Extended Audio channel 1 volume
AUD1DATX	% 2C8	W			P	Extended Audio channel 1 data
AUD1CTL	2CC	W			P	Audio channel 1 control bits
AUD2LCX	2D0	W			P	Extended Audio channel 2 backup pointer
AUD2LENX	2D4	W			P	Extended Audio channel 2 length
AUD2PERX	2D8	W			P	Extended Audio channel 2 period
AUD2VOLX	2DC	W			P	Extended Audio channel 2 volume
AUD2DATX	% 2E0	W			P	Extended Audio channel 2 data
AUD2CTL	2E4	W			P	Audio channel 2 control bits
AUD3LCX	2E8	W			P	Extended Audio channel 3 backup pointer
AUD3LENX	2EC	W			P	Extended Audio channel 3 length
AUD3PERX	2F0	W			P	Extended Audio channel 3 period
AUD3VOLX	2F4	W			P	Extended Audio channel 3 volume
AUD3DATX	% 2F8	W			P	Extended Audio channel 3 data
AUD3CTL	2FC	W			P	Audio channel 3 control bits
BPL1DATX	& 300	W		M		Extended Bitplane data register 1
BPL2DATX	& 304	W		M		Extended Bitplane data register 2
BPL3DATX	& 308	W		M		Extended Bitplane data register 3
BPL4DATX	& 30C	W		M		Extended Bitplane data register 4
BPL5DATX	& 310	W		M		Extended Bitplane data register 5
BPL6DATX	& 314	W		M		Extended Bitplane data register 6
	318					
	31C					
SPR0PTX	320	W	A			Extended SPRITE 0 pointer register
SPR1PTX	324	W	A			Extended SPRITE 1 pointer register
SPR2PTX	328	W	A			Extended SPRITE 2 pointer register
SPR3PTX	32C	W	A			Extended SPRITE 3 pointer register
SPR4PTX	330	W	A			Extended SPRITE 4 pointer register
SPR5PTX	334	W	A			Extended SPRITE 5 pointer register
SPR6PTX	338	W	A			Extended SPRITE 6 pointer register
SPR7PTX	33C	W	A			Extended SPRITE 7 pointer register
SPR0DATAX	& 340	W		M		Extended Sprite 0 image data register B
SPR0DATBX	& 344	W		M		Extended Sprite 0 image data register A
SPR1DATAX	& 348	W		M		Extended Sprite 1 image data register B

SPR1DATBX	& 34C	W		M	Extended Sprite 1 image data register A
SPR2DATAX	& 350	W		M	Extended Sprite 2 image data register B
SPR2DATBX	& 354	W		M	Extended Sprite 2 image data register A
SPR3DATAX	& 358	W		M	Extended Sprite 3 image data register B
SPR3DATBX	& 35C	W		M	Extended Sprite 3 image data register A
SPR4DATAX	& 360	W		M	Extended Sprite 4 image data register B
SPR4DATBX	& 364	W		M	Extended Sprite 4 image data register A
SPR5DATAX	& 368	W		M	Extended Sprite 5 image data register B
SPR5DATBX	& 36C	W		M	Extended Sprite 5 image data register A
SPR6DATAX	& 370	W		M	Extended Sprite 6 image data register B
SPR6DATBX	& 374	W		M	Extended Sprite 6 image data register A
SPR7DATAX	& 378	W		M	Extended Sprite 7 image data register B
SPR7DATBX	& 37C	W		M	Extended Sprite 7 image data register A
SPR0POSX	% 380	W	A	M	Extended Sprite 0 Vert start and stop position data
SPR0CTLX	% 384	W		M	Extended Sprite 0 position and control data
SPR1POSX	% 388	W	A	M	Extended Sprite 1 Vert start and stop position data
SPR1CTLX	% 38C	W		M	Extended Sprite 1 position and control data
SPR2POSX	% 390	W	A	M	Extended Sprite 2 Vert start and stop position data
SPR2CTLX	% 394	W		M	Extended Sprite 2 position and control data
SPR3POSX	% 398	W	A	M	Extended Sprite 3 Vert start and stop position data
SPR3CTLX	% 39C	W		M	Extended Sprite 3 position and control data
SPR4POSX	% 3A0	W	A	M	Extended Sprite 4 Vert start and stop position data
SPR4CTLX	% 3A4	W		M	Extended Sprite 4 position and control data
SPR5POSX	% 3A8	W	A	M	Extended Sprite 5 Vert start and stop position data
SPR5CTLX	% 3AC	W		M	Extended Sprite 5 position and control data
SPR6POSX	% 3B0	W	A	M	Extended Sprite 6 Vert start and stop position data
SPR6CTLX	% 3B4	W		M	Extended Sprite 6 position and control data
SPR7POSX	% 3B8	W	A	M	Extended Sprite 7 Vert start and stop position data
SPR7CTLX	% 3BC	W		M	Extended Sprite 7 position and control data
HTOTALX	3C0	W	A		Extended horiz line count register
DIWSTRTX	3C4	W	A	M	Extended Display Window Start
DIWSTOPX	3C8	W	A	M	Extended Display Window Stop
HBSTOPX	3CC	W	A		Extended horiz blank stop
HBSTRTX	3D0	W	A		Extended horiz blank start
HCENTERX	3D4	W	A		Extended horiz line center
HSSTOPX	3D8	W	A		Extended horiz sync stop
HSSTRTX	3DC	W	A		Extended horiz sync start
BPL1PTX	3E0	W	A		Extended Bitplane 1 pointer
BPL2PTX	3E4	W	A		Extended Bitplane 2 pointer
BPL3PTX	3E8	W	A		Extended Bitplane 3 pointer
BPL4PTX	3EC	W	A		Extended Bitplane 4 pointer
BPL5PTX	3F0	W	A		Extended Bitplane 5 pointer
BPL6PTX	3F4	W	A		Extended Bitplane 6 pointer
	3F8				
	3FC				
RAMATRN	~400	R	A		RAM Bank Attributes read register
COINTRER	~404	R	A		Coprocessor interrupt request/enable read register.
PRITEST	~408	R	A		Read Processor and Blitter priority control bits.
BLTROMD	~40C	R	A		Blitter ROM data.
AUDTST0R	~410	R		P	Read audio cycle addr, ctrl state, and data
AUDTST1R	~414	R		P	Read audio cycle control and test control bits
AUDRIGHTR	~418	R		P	Read right audio channel
SERDATR2N	~41C	R		P	Serial Port 2 Data and Status read
AUDTST0W	~420	W		P	Write audio cycle addr, ctrl state, and data
GENERALR	~424	R	A		General purpose register, read.
VPPOSRN	~428	R	A		Read Primary vertical position of beam.
BLITENRN	~42C	R	A		Blitter Enable register read.
PRISET	~430	W	A		Write Processor and Blitter priority control bits.
BLITEN	~434	W	A		Blitter Enable register.
AUDTST1W	~438	W		P	Write audio cycle control and test control bits
BLTLCLPAX	~43C	W	A		Blitter Clip Rectangle Coordinates, Source A.
BLTLCLPBX	~440	W	A		Blitter Clip Rectangle Coordinates, Source B.

BLTLCOLORX	~444	W		A		Blitter Line Color Register.
BLTLPATRNX	~448	W		A		Blitter Line Pattern Register.
BLTLPMSKX	~450	W		A		Blitter Line Plane Mask Register.
BLTLMAPPTX	~454	W		A		Blitter Line Source Pointer Register.
BLTLMAPWDX	~458	W		A		Blitter Line Width Register.
BLTLEND1X	~45C	W		A		Blitter Line End Point 1 Register.
BLTLEND2X	~460	W		A		Blitter Line End Point 2 Register.
BLTLFUNCX	~464	W		A		Blitter Line Function and Start Register.
BLTADATX	&~468	W		A		Blitter Source A Data Register.
BLTBDATX	&~46C	W		A		Blitter Source B Data Register.
BLTCDATX	&~470	W		A		Blitter Source C Data Register.
BLTDDATX	&~474	DR		A		Blitter Source D Data Register.
BLTEDATX	&~478	DR		A		Blitter Source E Data Register.
RAMATWN	~47C	W		A		RAM Bank Attributes write register
BLTTST	480	W		A		Blitter ROM test.
SERDAT2N	484	W			P	Serial Port 2 Data and stop bits write
MONITORID	488	R	External Hardware Monitor ID.			
SERPERN	48C	W			P	Serial Port 2 Period and control
BPLOFFR	490	R		M		Bitplane and Sprite LUT Address Offset read reg
CLXCONOEN	494	W		M		Chunky Pixel collision control; enable bits.
CLXCONODN	498	W		M		Chunky Pixel collision control; match bits.
MONTEST	49C	W		M		MONICA test register.
AUD4LCN	4A0	W			P	Extended Audio channel 4 backup pointer
AUD4LENN	4A4	W			P	Extended Audio channel 4 length
AUD4PERN	4A8	W			P	Extended Audio channel 4 period
AUD4VOLN	4AC	W			P	Extended Audio channel 4 volume
AUD4DATN	% 4B0	W			P	Extended Audio channel 4 data
AUD4CTL	4B4	W			P	Audio channel 4 control bits
AUD5LCN	4B8	W			P	Extended Audio channel 5 backup pointer
AUD5LENN	4BC	W			P	Extended Audio channel 5 length
AUD5PERN	4C0	W			P	Extended Audio channel 5 period
AUD5VOLN	4C4	W			P	Extended Audio channel 5 volume
AUD5DATN	% 4C8	W			P	Extended Audio channel 5 data
AUD5CTL	4CC	W			P	Audio channel 5 control bits
AUD6LCN	4D0	W			P	Extended Audio channel 6 backup pointer
AUD6LENN	4D4	W			P	Extended Audio channel 6 length
AUD6PERN	4D8	W			P	Extended Audio channel 6 period
AUD6VOLN	4DC	W			P	Extended Audio channel 6 volume
AUD6DATN	% 4E0	W			P	Extended Audio channel 6 data
AUD6CTL	4E4	W			P	Audio channel 6 control bits
AUD7LCN	4E8	W			P	Extended Audio channel 7 backup pointer
AUD7LENN	4EC	W			P	Extended Audio channel 7 length
AUD7PERN	4F0	W			P	Extended Audio channel 7 period
AUD7VOLN	4F4	W			P	Extended Audio channel 7 volume
AUD7DATN	% 4F8	W			P	Extended Audio channel 7 data
AUD7CTL	4FC	W			P	Audio channel 7 control bits
GRAPHICS	& 500	W	L			Graphics fetch cycle
BPLOFFN	504	W		M		Bitplane and Sprite offset register.
BPL7DATN	& 508	W		M		New Bit plane 7 data (Parallel to serial convert)
BPL8DATN	& 50C	W		M		New Bit plane 8 data (Parallel to serial convert)
BPL9DATN	& 510	W		M		New Bit plane 9 data (Parallel to serial convert)
BPL10DATN	& 514	W		M		New Bit plane 10 data (Parallel to serial convert)
BPL11DATN	& 518	W		M		New Bit plane 11 data (Parallel to serial convert)
BPL12DATN	& 51C	W		M		New Bit plane 12 data (Parallel to serial convert)
BPL13DATN	& 520	W		M		New Bit plane 13 data (Parallel to serial convert)
BPL14DATN	& 524	W		M		New Bit plane 14 data (Parallel to serial convert)
BPL15DATN	& 528	W		M		New Bit plane 15 data (Parallel to serial convert)
BPL16DATN	& 52C	W		M		New Bit plane 16 data (Parallel to serial convert)
BPLOVRDAT	& 530	W		M		Overlay Bitplane data (Parallel to serial convert)
COPWAIT	534	W		A		Coprocessor wait interrupt routine start address.
COPJMP3	538	W		A		Coprocessor restart at third location.
COPJMP4	53C	W		A		Coprocessor restart at fourth location.

COP3LC	540	W	A		Coprocessor third jump location register.
COP4LC	544	W	A		Coprocessor fourth jump location register.
COPBLITLC	548	W	A		Blitter Finished interrupt starting addr.
COPRET	54C	S	A		Coprocessor return from interrupt strobe.
COINTREW	550	W	A		Coprocessor interrupt request/enable write reg.
COPRETC	554	S	A		Coprocessor return from blitter interrupt strobe.
	558				
	55C				
GENERALW	560	W	A		General purpose register, write.
	564				
BRSTSTRT	568	W	A		Burst start.
BRSTSTOP	56C	W	A		Burst stop.
EQU1STOP	570	W	A		First equalization pulse stop position.
EQU2STOP	574	W	A		Second equalization pulse stop position.
SER1STOP	578	W	A		First serration pulse stop position.
SER2STOP	57C	W	A		Second serration pulse stop position.
VEQUSTOP	580	W	A		Vertical equalization stop line.
CLCNTR	584	W	A		Composite Line Center.
MLSYNC	588	W	A		Mid Line Sync.
	58C				
DSKCRCI	590	W		P	Initial CRC value register.
DSKGAP	594	W		P	Disk gap timer register.
DSKLENN	598	W	A	P	New mode length register.
DSKHDPT	59C	W	A		New disk DMA header pointer.
DSKBKPT	5A0	W	A		New disk DMA backup pointer.
DSKTST	5A4	W		P	Disk test register.
	5A8				
	5AC				
	5B0				
	5B4				
	5B8				
	5BC				
	5C0				
	5C4				
	5C8				
	5CC				
	5D0				
	5D4				
BPL7PT	5D8	W	A		New bit plane 7 pointer
BPL8PT	5DC	W	A		New bit plane 8 pointer
BPL9PT	5E0	W	A		New bit plane 9 pointer
BPL10PT	5E4	W	A		New bit plane 10 pointer
BPL117PT	5E8	W	A		New bit plane 11 pointer
BPL12PT	5EC	W	A		New bit plane 12 pointer
BPL13PT	5F0	W	A		New bit plane 13 pointer
BPL14PT	5F4	W	A		New bit plane 14 pointer
BPL15PT	5F8	W	A		New bit plane 15 pointer
BPL16PT	5FC	W	A		New bit plane 16 pointer
	600-7FC				reserved.
COLORnRX	800- thru BFC	R		M	Extended color register read addresses.
COLORnWX	C00- thru FFC	W		M	Extended color register write addresses.

APPENDIX F

RGA REGISTER BITS.

REGBITS.DOC (13-Apr-90) New AMIGA chip extended registers (jwr)

n= new register
x= new 32 bit extended register

ADKCON 09E W P Audio, Disk, Uart, Control write
ADKCONR 010 R P Audio, Disk, Uart, Control read

BIT#	USE	
15	SET/CLR	Set/Clear control bit. Determines if bits written with a 1 get set or cleared. Bits written with a zero are always unchanged.
14-13	PRECOMP 1-0	CODE PRECOMP VALUE 00 none 01 140 ns 10 280 ns 11 560 ns
12	MFMPREC	(1=MFM precomp 0=GCR precomp)
11	UARTBRK	Forces a UART break (clears TXD) if true
10	WORDSYNC	Enables disk read synchronizing on a word equal to DISK SYNC CODE, located in address DSKSYNC (7E).
09	MSBSYNC	Enables disk read synchronizing on the MSB (most signif bit). Appl type GCR
08	FAST	Disk data clock rate control 1=fast(2us) 0=slow(4us) (fast for MFM or 2us GCR, slow for 4us GCR)
07	USE3PN	Use audio channel 3 to modulate nothing
06	USE2P3	Use audio channel 2 to modulate period of channel 3
05	USE1P2	Use audio channel 1 to modulate period of channel 2
04	USE0P1	Use audio channel 0 to modulate period of channel 1
03	USE3VN	Use audio channel 3 to modulate nothing
02	USE2V3	Use audio channel 2 to modulate volume of channel 3
01	USE1V2	Use audio channel 1 to modulate volume of channel 2
00	USE0V1	Use audio channel 0 to modulate volume of channel 1

NOTE If both period and volume are modulated on the same channel, the period and volume wil be alternated. First AUDxDAT word is used for V6-V0 of AUDxVOL Second AUDxDAT word is used for P15-P0 of AUDxPER this alternating sequence is repeated. When any of the disk control bits are written, all the new mode disk bits are cleared (SHORTPULSE, RLLPREC, SYNCEN1, LSBFIRST, INCSYNC, MODE1, MODE0, CODE1, CODE0). SYNCEN0 is the same bit as WDSYNCEN, and is not cleared.

ADKCONX x 28C W P Extended Disk, Control write
ADKCONRX x 210 R P Extended Disk, Control read

BIT#	USE	
31	SET/CLR	Set/Clear control bit. Determines if bits written with a 1 get set or cleared. Bits written with a zero are always unchanged.
30	x	Should always be set to 0.
29	SHORTPULSE	Make output pulse on write 1 instead of 2 CPLL ticks CPLL- 35ns @FASTB==1, 140ns @FASTB==0;
28	UART2BRK	Forces a break on UART 2(clears TXD2) if true
27	UARTBRK	Forces a UART break (clears TXD) if true
26	AUDMONO	Global audio mono. When this bit is set

all active audio channels are summed and output to both left and right output pins (including both analog and digital outputs) disregarding the LEFT/RIGHT assignment bits in AUDxCTL. When this bit is reset, left and right channel assignments revert back to the original settings. This bit allows a user with one speaker to always hear the audio.

25-24 AUDSCALE1-0 These control bits govern scaling of the audio channels to permit summation of varying number of channels without clipping, yet close to full volume.

CODE	MEANING
10	Normal. Max volume achieved with 8 channels a side.
01	Times 2. Max volume achieved with 4 channels a side.
00	Times 4. Max volume achieved with 2 channels a side.

(Note: 'side' is one of left/right. Power up state is 00.)

23-16 unused Should always be set to 0.

15 RLLPREC Precomp on bits 3 away from center (for RLL(2,7))

14 PRECOMP1 Amount of precomp

13 PRECOMP0 Amount of precomp

PRECOMP1,0	FASTB=0	FASTB=1
00	0	0
01	140ns	35ns
10	280ns	70ns
11	560ns	140ns

(PRECOMP 1-0 are same bits as PRECOMP 1-0 of ADKCON.)

12 MFMPREC precomp on bits 2 away from center (for MFM) if RLLPREC=0 && MFMPREC==0, precomp is on 1 bit from center (GCR)
(Same bit as MFMPREC of ADKCON.)

11 SYNCEN1 Length of data to sync on

10 SYNCEN0

SYNCEN1	SYNCEN0	length
0	0	no sync
0	1	16 bits (lower 16 bits of sync reg)
1	0	32 bits
1	1	8 bits (lower 8 bits of sync reg)

09 MSBSYNC Enables disk read synchronizing on the MSB (most signif bit). Appl type GCR
(Same bit as MSBSYNC of ADKCON.)

08 FASTA 2* data sep speed (count around in half as many ticks)

07 FASTB 4* data sep speed (clock speed change-28M vs 7M)
(FASTA is same bit as FAST of ADKCON.)
(both fastb and fasta can be set at once)

06 BOTHEDGES Use NRZ encoding for data in/out (CD stuff) (an edge rather than a pulse signifies a '1')

05 LSBFIRST Send the LSB of data out/in first (CD stuff)

(CD format sends/receives the LSB of data first) (this saves a lot of bit mirroring in software)

04 INCSYNC — Sync data is included in the data input stream after a sync mark occurs. This allows alignment of CD data on input/output to be the same.

03 MODE1
02 MODE0 — Disk controller major modes of operation

MODE1	MODE0	
0	0	Track mode. This is the old controller mode One chunk of data is read/written (DSKLEN*2 bytes long), with a DSKBLKIR at the end DSKPTX is used as data pointer. If SYNCEN not zero, it waits for a sync mark to start the dma. (DSKLEN on dsken,LEN3 on dskenx)
0	1	Sector mode. Data from DSKHDPTX in memory is matched against the disk. When it matches data is read/written using DSKPTX pointer and LEN3 for that sector. SECTCNT determines how many sectors are done in a row. After a sync mark, data from memory (DSKHDPTX) is matched with the raw address header on the disk. If no match, it waits for the next sync mark & tries again indefinitely. (LEN1*2 of data is matched) Next, on read, a second string of data is matched starting at a sync mark. (data mark) On match failure, NOTFOUND is set (DSKBYTR) and the controller stops. NOTFOUND is also set if it takes longer than GAPLEN*2 raw bytes to find the header. LEN2*2 raw bytes are matched. After this, LEN3*2 bytes of decoded data are stored in memory at DSKPTX pointer, and the SECTCNT is decremented. If zero, a DSKBLKIR is generated, else it goes back to look for another sector header. CRC is also checked at the end of the read data. If CRC failure, the controller continues, but sets the CRCERR bit. Next, on write, GAPLEN*2 raw bytes are waited for, then raw data is written for LEN2*2 bytes (DSKHDPTR), then data from memory (DSKPTX) is encoded for LEN3*2 bytes & sent to the disk, then a CRC is generated and sent to the disk, followed by bits 15-8 of data from DSKHDPTX. The SECTCNT is decremented, and if not zero, a new header is looked for. A DSKBLKIR is sent out after all sectors finished. At the beginning of the first sector or match failure, DSKHDPTX=DSKBKPTX so restart of match occurs at the correct spot. At the end of a sector DSKBKPTX=DSKHDPTX so that once a sector is finished, the restart will be to the beginning of the next sector.
1	0	CD/DAT mode. Reads/writes a number of blocks of data (SECTCNT) and gives a block interrupt every sector, at a time when the DSKBKPTX can be rewritten. This allows double buffering for continuous output/input. On startup, DSKPTX should be written to the address of the first buffer, DSKBKPTX should have the address of the 2nd. This mode does strange things with the sync on write. The lower 4 bits are not encoded directly, they are used to generate the 3

necessary sync marks:

lower 4 bits	sent out (raw)
0	bits 31-24 of DSKSYNCX
1	31-28, then 23-20
2	31-28, then 19-16
3	same as 2

The DSKGAPX register can be written with a >1 sector number at the same time as the DSKBKPTX is written to keep things going indefinitely.

1 1 TRACKPLUS mode. Designed as a limited format mode for transfers at double the speed of SECTOR mode for the same dma bandwidth

write, for SECTCNT sectors:
1) write LEN1*2 bytes of data from DSKHDRPTX (encoded)
2) write LEN2*2 bytes of data from DSKHDRPTX (even LEN2 only)
first 48 bits of raw data are 3 copies of the lower 16 bits of DSKSYNCX. The last 16 are encoded version of the lower 8 bits of the data at DSKHDRPTX
3) write LEN3*2 bytes of data from DSKPTX (encoded)
4) write CRC for DSKPTX data, followed by bits 15-0 of DSKHDRPTX data, encoded.

read, for SECTCNT sectors:
1) wait for sync mark
2) read in LEN3*2 bytes of data
3) check CRC

In all modes, there is a chance the hardware will hang forever under some conditions. It is software responsibility to enable a max global timeout (see INTREQX for how to stop controller)

01 CODE1
00 CODE0 Type of encode/decode to use

CODE1	CODE0	type of encode/decode
0	0	none- raw bits in/out
0	1	MFM
1	0	RLL(2,7)
1	1	Biphase/Mark (CD/DAT)

For header raw data generation/matching, RLL.C has the encode/decode algorithm for RLL(2,7).

AUDnLCH 0A0 W A Audio channel n location (High 5 bits)
AUDnLCL 0A2 W A Audio channel n location (Low 16 bits, A0 forced low)

This pair of registers contains the 20 bit starting word address (location) of Audio channel n (n=0,1,2,3) DMA data. This is not a pointer register and therefore only needs to be reloaded if a different memory location is to be output. If AUDnEN of DMACON is enabled, 16 bit word transfers from the address pointed to by this register will be done with each AUDn DMA request. If AUDnENX of DMACONX is enabled, 32 bit longword transfers from the address pointed to by AUDnLC(A1,0 forced low) will be done with each AUDn DMA request.

AUDnLC nx 2A0, 2B8, 2D0, 2E8, 4A0, W A Audio channel n location (24 bits, A1,0 forced low)

n=0-7. When n=0-3, these registers are the same as the AUDnLCH or AUDnLCL address registers except they include 3 extra address bits A23-A21. When AUDnLCH or AUDnLCL above are written bits A23-A21 are cleared.

4B8,
4D0,
4E8

AUDxLEN 0A4, W P Audio Channel x length
0B4, This register contains the length (number of 16 bit words) of
0C4, Audio Channel x DMA data.
0D4

AUDnLENX 2A4, W P Audio Channel n length, n = 0 to 3.
2BC, These registers map directly into AUDxLEN. Data should
2D4, be presented to data bus bits 15-0. As in old mode audio,
2EC this length is in 2 byte increments.

AUDnLENN 4A4, W P Audio Channel n length, n = 4 to 7.
4BC, These registers have the same function as AUDnLENX, except
4D4, they control audio channel 4 thru 7 length.
4EC

AUDxPER 0A6, W P Audio channel x Period
0B6, This register contains the Period (rate) of Audio channel
0C6, x DMA data transfer in 280ns increments.
0D6 Because MARY DMA requests are handled differently than previous versions of the chip(DMA requests are sent serially to ANDREA whenever a channel needs serviced), the theoretical minimum period is 8 color clocks. This assumes no higher priority channel needs service when the request is made, and corresponds to a sample frequency of 447.4Khz. Practically, the smallest number which should be placed in this register is 70 which corresponds to a sample frequency of 51Khz.

AUDnPERX 2A8, W P Audio channel n period, n=0 to 3.
2C0, These registers map directly into AUDxPER with the exception
2D8, that data is presented to data bus bits 31-16.
2F0 When AUDPERF(AUDnCTL) is 0, the lsb (bit 16) specifies 279.3651 nsec.
When AUDPERF(AUDnCTL) is 1, the lsb (bit 16) specifies 4.3650797 nsec.

AUDnPERN 4A8, W P Audio Channel n period, n = 4 to 7.
4C0, These registers have the same function as AUDnPERX, except
4D8, they control audio channel 4 thru 7 length.
4F0

AUDxVOL 0A8, W P Audio Channel x Volume
0B8,
0C8,
0D8

This register contains the Volume setting for Audio Channel x. Bits 6,5,4,3,2,1,0 specify 65 linear volume levels as shown below.

BITS	USE
15-07	Not used
06	Forces volume to max (64 ones, no zeros)
05-00	Sets one of 64 levels (000000=no output (111111=63 Ones, one zero)

AUDnVOLX 2AC, W P Extended Audio channel n volume(n=0 to 3).
2C4, These registers contain the new mode volume setting for

2DC, 2F4 — Audio channel n. New mode volume permits signed 2's complement volume settings as specified below.

BITS	USE
31	Sign bit.
30-20	Volume Magnitude. When AUDSCALE of ADKCONX is 0, bit 30 permits drive of 1/2 full scale output 1, bit 30 permits drive of 1/4 full scale output 2, bit 30 permits drive of 1/8 full scale output Note that when AUDxVOL is written, if bit 6 is set bit 31 of AUDnVOLX is reset, and bits 30 thru 20 are set. If bit 6 is 0, then bit 31 of AUDnVOLX is reset, bits 24 to 20 are reset, and bits 5 to 0 of AUDxVOL are loaded into bits 30 to 25 of AUDnVOLX.

AUDnVOLN 4AC, 4C4, 4DC, 4F4 W P — New Audio channel n volume(n=4 to 7). These registers have the same function as AUDnVOLX except they control channels 4 thru 7.

AUDxDAT 0AA, 0BA, 0CA, 0DA W P — Audio channel x Data
This register is the Audio channel x (x=0,1,2,3) DMA data buffer. It contains 2 samples of data (each sample is a 8 bit twos complement signed integer) that are output sequentially (with digital to analog conversion) to the audio output pins.
The audio DMA channel controller automatically transfers data to this register from RAM. The Processor can also write directly to this register. When the DMA data is finished (words output=Length) and the data in this register has been used, an audio channel interrupt request is set. DMA transfers are made to this register instead of to AUDxDATX when AUDxEN of DMACON has been set.

AUDnDAT n 1F0, 1F2, 1F4, 1F6 W P — Audio channel n data, n=4 to 7.
These registers are the same as AUDxDAT above, except they apply to the new audio channels 4 to 7.

AUDnDATX x 2B0, 2C8, 2E0, 2F8 W P — Extended Audio channel n Data n=0-3
This register is the extended Audio channel x (x=0,1,2,3) DMA data buffer. It contains 2 or 4 samples of twos complement data that are output sequentially to the audio output pins. DMA transfers are made to this register instead of to AUDxDAT when AUDxENX of DMACONX has been set. When 16 bit samples are used, bits 31-16 contain sample 0 and bits 15-0 contain sample 1. When 8 bit samples are chosen, bits 31-24 contain sample0, 23-16 contain sample1, 15-8 contain sample2, and 7-0 contain sample 3. Note that since AUDnLENX counts the number of words transferred, bits 31-16 may be used as the last sample(s) instead of bits 15-0.

AUDnDATN n 4B0, 4C8, 4E0, 4F8 W P — New Audio channel n Data n=4-7
These registers have the same function as AUDnDATX except they apply to channels 4 thru 7.

AUDnCTL n 2B4, 2CC, 2E4, 2FC, W P — Audio channel n control bits n=0-7
These bits are only active when the audio circuits are in 'new mode'. Note that channels 4-7 are always in new mode and only channels 0-3 can be in old mode.

4B4, 4CC, 4E4, 4FC

Old mode is provided to support old audio software. Channels 0-3 are in old mode after power up, whenever an audio DMACON bit is set, and when a write is done to one of the AUD0-3DAT registers. Audio is in new mode when an audio DMACONX bit is set, or when one of the AUDnDATX or AUDnDATN registers is written.

BIT	FUNCTION
31-8	x
7	AUDTURBO
6	AUDSTOP
5	AUDPERF
4	AUD16
3	AUDAP
2	AUDAV
1	LEFT
0	RIGHT

x's should be written with 0's

AUDTURBO Audio Turbo mode. Volume resolution is sacrificed but sampling rates to 110Khz are possible. AUDnVOLX bits 25 thru 20 must be zero when this bit is set. When this bit is clear, the max sampling rate possible is 64Khz.

AUDSTOP Audio Stop. When the audio channel is being dma driven and this bit is set, the audio channel automatically stops after one pass thru the DMA samples(when length times out). When the audio channel dma is not enabled (ie, interrupt driven) and this bit is set, the audio channel automatically stops after a pair of samples is processed. Normally, this bit should be set when the dma is required to end after the end of the next buffer (it can be set after the buffer end intreq). It must be reset for >= 8 CCKs (2.24us) in order to allow the dma to start again if so required. The Overrun intreq can be used to time this reset.

AUDPERF Audio Period Fine. Allows finer period resolution than can normally achieved, but with a small amount of induced distortion to the audio. When this bit is clear, the lsb of the AUDnPERX register signifies 279.3651 nsec increments. When this bit is set, the lsb of the AUDnPERX register signifies 4.3650797 nsec increments. See manual for a detailed description of this mode.

AUD16 Audio channel uses 16 bit samples. When set, 16 bit samples. When clear, 8 bit samples. Reset on power up.

AUDAP Attach period. In channels 0-3 and in old mode, these bits are ignored(USE0P1 thru USE3PN of ADKCON are used instead.) In new mode, these bits control whether or not this channels data is used to modulate the period of the next sequential channel. Reset on power up.

AUDAV Attach volume. In channels 0-3 and in old mode, these bits are ignored(USE0V1 thru USE3VN of ADKCON are used instead.) In new mode, these bits control whether or not this channels data is used to modulate the volume of the next sequential channel. Reset on power up.

LEFT When this bit is set, this channels output is summed onto the left speaker channel. This bit is set for channels 0 and 3 on power on reset,

and reset for all others.

RIGHT When this bit is set, this channels output is summed onto the right speaker channel. This bit is set for channels 1 and 2 on power on reset, and reset for all others.

AUDLEFTR 200 R P Read left audio channel. This register is for test purposes only.

BITS	FUNCTION
31-21	x
20-0	Output to left DAC

AUDRIGHTR 418 R P Read right audio channel. This register is for test purposes only.

BITS	FUNCTION
31-21	x
20-0	Output to right DAC

AUDTST0W 420 W P Write audio cycle addr, ctrl state, and data. This register is for test purposes only and should never be written by user software.

BITS	FUNCTION
31-26	Address to main calculation loop (read address)
25-18	New control state for main loop (@ readaddr-3)
17-0	New data state (result of calculations)

AUDTST0R 410 R P Read audio cycle addr, ctrl state, and data This register is for test purposes only.

BITS	FUNCTION
31-26	Address to main loop read/write (read address)
25-18	Control state for main loop (newctl)
17-0	Data created by main loop data and control

AUDTST1W 438 W P Write audio cycle control and test control bits.
AUDTST1R 414 W P Read audio cycle control and test control bits. These registers are for test purposes only. Write of this register writes the ones enabled by the UTESTC, read reads the ones going to the data path.

BITS	FUNCTION
31-21	x (write to 0)
20	ANDU
19	ANDM
18	ANDL
17	SH3
16	SH2
15	SH1
14	M7
13	M6
12	M5
11	M4
10	M3
9	M2
8	M1
7	SUB
6	NEWLYDONE
5	ATTACHW
4	x(write to 0)

3	UTESTA	use test addr (fm AUDTST0W)
2	UTESTCS	use test control state (fm AUDTST0W)
1	UTESTD	use test data (fm AUDTST0W)
0	UTESTC	use test control- use above bits(5-18) for data path control

BEAMCON0 1DC W A M Beam Counter control bits

Bit	Function
15	COPRSEL
14	x
13	LPENDIS
12-11	x
10	BEAMLCK
09-07	x
6	NTSC
5	PAL
4	BEAMEN
3	x
2	CSYTRUE
1	VSYTRUE
0	HSYTRUE

Warning- None of the bits in this register should be written by the user. Appropriate system software calls should be made to invoke the desired display mode. Monica is only concerned with the CSYTRUE bit in this register.

COPRSEL= enable the vertical primary counter to do copper wait position comparisons rather than the secondary counter. COPRSEL = 1, use primary counter outputs when making copper wait instruction comparisons. COPRSEL = 0, use secondary counter outputs when making copper wait instruction comparisons.

LPENDIS= When this bit is a low and LPE (BPLCON0,BIT 3) is enabled, the light-pen latched value (beam hit position) will be read by VHPOSR, VPOSR and HHPOSR. When the bit is a high the light-pen latched value is ignored and the actual beam counter position is read by VHPOSR, VPOSR and HHPOSR.

BEAMLCK= BEAMEN LOCK. When this bit is set, the system reset signal will not affect the BEAMEN bit of this register, or the HTOTAL register. When this bit is clear, the system reset signal is permitted to clear the BEAMEN bit and preset the HTOTAL register. This bit is cleared at power on time. System software should set this bit after programming the display control registers (HTOTAL, HSSTRT, HSSTOP, etc) for the monitor connected to the system, thus permitting the retention of display control register values after a system reset sequence. This bit is cleared at power on time, thus enabling the clear of BEAMEN and HTOTAL at power on time. The system reset signal does not affect this bit.

NTSC= Enable appropriate decodes for NTSC.

PAL= Enable appropriate decodes for PAL.

PAL and NTSC bits enable specific graphics circuit functionality and should only be used when the system is driving real NTSC or PAL monitors/TV's, and when NTSC or PAL GENLOCK devices are being used in the system. When either of the NTSC or PAL bits is set, the correct pixel frequency must be present, and HBSTRTX, HBSTOPX, HCENTERX, HSSTRTX, HSSTOPX, BRSTSTRT, BRSTSTOP, EQU1STOP, EQU2STOP, SER1STOP, SER2STOP, VBSTRT, VBSTOP, VSSTRT, VSSTOP, VTOTAL,

CLCNTR, and MLSYNC must be programmed to the appropriate fixed settings.
BEAMEN= beam enable. When this bit is cleared, the VSYNC and HSYNC signals are disabled, thus disabling the video display. This bit should be set to enable video after the display control registers have been set up for the monitor currently plugged into the system. When the BEAMLCK bit in this register is clear, system reset will clear this bit (BEAMEN is cleared at power on time). When the BEAMLCK bit is set, only writes to this register can affect BEAMEN.
HSYTRUE,VSYTRUE,CSYTRUE= These change the polarity of the HSY*, VSY*, & CSY* pins to HSY, VSY,& CSY respectively for input & output.

BLTxPTH 050 W A Blitter Pointer to x (High 5 bits)
BLTxPTL 052 W A Blitter Pointer to x (Low 16 bits, bit 0 forced low)
This pair of registers contains the 20 bit word address of Blitter source (x=A,B,C) or dest. (x=D) DMA data. This pointer must be preloaded with the starting address of the data to be processed by the blitter. After the Blitter is finished it will contain the last data address (plus increment and modulo). If BLTEN of DMACON is enabled, 16 bit word transfers from the address pointed to by this register will be done with each BLIT transfer. If BLTENX of DMACONX is enabled, 32 bit longword transfers from the address pointed to by this register(A1,0 forced low) will be done. If BLTENX of DMACONX is enabled, and BBRSTENX of DMACONX are enabled, then longword burst transfers from the address pointed to by this register will be done. Note BLITTER BURST mode differs slightly form processor burst mode. No modulo 4 counting mechanism will inhibit A23-A4 from incrementing when A3,A2 goes from 11 to 00. A counter inhibit mechanism is used when ANDREA is servicing a host processor burst request. Another difference is that blitter burst mode cycles may fetch less than 4 longwords because blit fetches can start at any longword address. Less than 4 longwords are burst fetched when a RAM page boundary is crossed, and possibly at the end of a blit line fetch.

BLTxMOD 064 W A Blitter Modulo x
This register contains the Modulo for Blitter source(x=A,B,C) or Dest (x=D). A Modulo is a number that is automatically added to the address at the end of each line, in order that the address then points to the start of the next line. Each source or destination has it's own Modulo, allowing each to be a different size, while an identical area of each is used in the Blitter operation.

BLTAFWM 044 W A Blitter first word mask for source A
BLTALWM 046 W A Blitter last word mask for source A
The patterns in these two registers are "anded" with the first and last words of each line of data from Source A into the Blitter. A zero in any bit overrides data from Source A. These registers should be set to all "ones" for fill mode or for line drawing mode.

BLITENRN n 42C R A Blitter Enable register read
BLITEN n 434 W A Blitter Enable register
This register holds the external blitter bits which are used to enable external blit chips in the system. All external blit chips are programmed with the same RGA addresses that are used to program the blitter on ANDREA. These blitter

enable bits are used in conjuction with the normal RGA address to specify which BLIT chips are to respond to the RGA address. The value in this register is driven onto bits 23-16 of the AD bus during the RGA portion of the memory cycle for use by the external BLIT chip decoders. This register is cleared on power up. When this register is zero, the blitter on ANDREA is being accessed. It is not possible to use the blitter on ANDREA at the same time an external blit is accessed. When external blits are underway, the ANDREA blitter is used to 'shadow' the operation so as to put out the proper addresses/commands at the proper times. The system is a single instruction, multiple data machine.

Bit Posi.	Meaning
31-24	x- don't care. Should be set to 0.
23	BLIT chip at RAM slot 7 enabled
22	BLIT chip at RAM slot 6 enabled
21	BLIT chip at RAM slot 5 enabled
20	BLIT chip at RAM slot 4 enabled
19	BLIT chip at RAM slot 3 enabled
18	BLIT chip at RAM slot 2 enabled
17	BLIT chip at RAM slot 1 enabled
16	BLIT chip at RAM slot 0 enabled
15-0	x- don't care. Should be set to 0.

BLTnDAT 074, W A Blitter source n data reg
072,
070
This register holds Source n (n=A,B,C) data for use by the Blitter. It is normally loaded by the Blitter DMA channel, however it may also be preloaded by the microprocessor.

BLTxDATX 468 W A Blitter data registers
46C
470
These extended 32 bit register addresses are generated by the blitter during DMA operations, where x is one of A, B, or C. The least significant word of these registers is the same as the corresponding BLTnDAT register meaning old blitter data will change as the result of a new mode blitter operation.

BLTDDAT 000 W A Blitter destination data register
BLTdDATX x 474 W A Extended Blitter destination data register
478
These registers hold the data resulting from each word of Blitter operation until it is sent to a RAM destination, where d is one of D or E (E is used in the sort operation). These are dummy addresses and cannot be read by the micro. The transfer is automatic during Blitter operation. The BLTDDAT register occupies the least significant word of the BLTDDATX register which is used when 32 bit BLITs are being done.

BLTCON0 040 W A Blitter control register 0
BLTCON0L 05A W A Blitter control register 0 (write lower 8 bits only)
This is to speed up software-- the upper bits are often the same.
BLTCON1 042 W A Blitter control register 1
These two control registers are used together to control Blitter operations.There are 2 basic modes, area and line, which are selected by bit 0 of BLTCON1, as shown below.

	AREA MODE ("normal")		LINE MODE (line draw)	
BIT#	BLTCON0	BLTCON1	BLTCON0	BLTCON1
15	ASH3	BSH3	ASH3	BSH3
14	ASH2	BSH2	ASH2	BSH2
13	ASH1	BSH1	ASH1	BSH1
12	ASA0	BSH0	ASH0	BSH0
11	USEA	0	1	0
10	USEB	0	0	0
09	USEC	0	1	0
08	USED	0	1	0
07	LF7	DOFF	LF7	DOFF
06	LF6	0	LF6	SIGN
05	LF5	0	LF5	OVF
04	LF4	EFE	LF4	SUD
03	LF3	IFE	LF3	SUL
02	LF2	FCI	LF2	AUL
01	LF1	DESC	LF1	SING
00	LF0	LINE(=0)	LF0	LINE(=1)

ASH3-0 Shift value of A source
BSH3-0 Shift value of B source and line texture
USEA Mode control bit to use Source A
USEB Mode control bit to use Source B
USEC Mode control bit to use Source C
USED Mode control bit to use Destination D
LF7-0 Logic function minterm select lines
EFE Exclusive fill enable
IFE Inclusive fill enable
FCI Fill carry input
DESC Descending (decreasing address) control bit
LINE Line mode control bit
SIGN Line draw sign flag
OVF Line draw r/l word overflow flag
SUD Line draw, Sometimes Up or Down (=AUD*)
SUL Line draw, Sometimes Up or Left
AUL Line draw, Always Up or Left
SING Line draw, Single bit per horiz. line
DOFF Disables the D output- for external ALUs
The cycle occurs normally, but the data bus is tristate. (hires chips only)

BLTLFUNCX 464 W A Blit line function and start register.
BLTFUNCX 278 W A Blit function and start register.

This extended 32 bit register is written to start the blitter and contains control information. Once this register has been written, no other blitter register may be written until the blitter has completed its operation. Writing any blitter register before then will cause undefined results.

Bit#	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	B7	B6	B5	B4	B3	B2	B1	B0	C2	C1	C0	CT			SG	O4
									EF	IF	FE	CK				

Bit#	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
	O3	O2	O1	O0	UD	UC	UB	UA	M7	M6	M5	M4	M3	M2	M1	M0

B7-B0 The number of bits per pixel (Upper 5 bits specify the number of bits; the lower 3 bits specify the power of two: i.e. 2**n).

The number of bits per pixel must be a power of two. Therefore, the values of B[4-0]N[2-0] may only be:

Bits/pixel	B[7-0] (hex)
1	08
2	11
4	22
8	43
16	84
32	05

C2-C0 Clipping information

C2-C0	Definition
0xx	No Clipping
100	Clip internal inclusive
101	Clip internal exclusive
110	Clip external exclusive
111	Clip external inclusive

Where internal is defined as writing the pixels of a line contained inside the clipping rectangle; Inclusive is defined as including the clip rectangle edge as part of the clip rectangle.

CT - Continue bit - Set (in new line mode) if line is a continuation of a previous line.

FE - Fill carry input

IF - Inclusive fill enable

EF - Exclusive fill enable

CK - 'CHUNKY' pixel style addition. When set, causes the hardware carry chain to be broken at 5-bit boundaries in lower and upper words to match the storage format of CHUNKY (16-bit) pixels.

SG - Single bit - Set (in new line mode) to draw a single bit per horizontal raster. Used to create fill patterns.

O4-O0 Operation

Bit O4 distinguishes between traditional blitter use and special modes.

O3-O0 (hex)	Operation
2	New Blit (1 operand)
3	New Blit (2 operand)
4	New Blit (3 operand)
5	New Reverse Blit (1 operand)
6	New Reverse Blit (2 operand)
7	New Reverse Blit (3 operand)
8	New line
9	Place holder for old mode blit (not set by user)

a	Place holder for old mode line (not set by user)
b	Tally
c	Sort1 (A source is sorted)
d	Sort2 (A source is sorted, B source follows)

UD-UA DMA channels used

M7-M0 Minterms or operation

The minterms to be applied during traditional blitting, or the operation to be applied during special blitting.

During special blitting, the following applies:

M7-M0	Operation
1	Add2
2	Add3
3	Sub2
4	Add2 Saturated
5	Add3 Saturated
6	Sub2 Saturated
7	Add2 Averaged

BLTLCLPxX 43C W A Blitter Clip Rectangle Coordinates Registers.
440 These extended 32 bit registers contain X and Y coordinate values of the upper left hand and lower right hand corners of the clip rectangle relative to the base origin. Clip point A must be closer to the origin than clip point b.

Bit#	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Bit#	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

BLTLCOLORX 444 W A Blitter Linedraw Color Register.
This register contains the color which will be used to paint each pixel of the line being drawn. The color must be repeated for each possible pixel location within a 32 bit word. (See BLTPMSKX.)

BLTLLENX 238 R A Blitter line length read register. Can be read to obtain the number of pixels in the last line processed. This is not necessarily the number of pixels drawn if the last line processed was in clip-rec mode.

BLTLENDxX 45C W A Blitter Line End Point Registers.
460 These extended 32 bit registers contain X and Y coordinate values of the two ends of the line to be drawn. The values are signed 14 bit integers which have been extended to 16 bits. (Uses same bit positions as BLTLCLPxX.)

BLTLMAPPTX 454 W A Blitter Linedraw Bit map pointer register.

BLTxSRCPTX 24C W A Blitter Bit map x pointer.
250 These extended 32 bit registers contain the bit address
258 of the bitmaps which are being operated on by the blitter. They contain the bit address (of the most significant bit)

of the pixel in the upper left hand corner of the bitmap. (Lower right hand corner for reverse blits). x may be one of A, B, or D. When three sources are used in an operation, the third source (C) is the same as D.

BLTLMAPWDX 458 W A Blitter Linedraw Bitmap Width Register.
BLTxSRCWDX 25C W A Bit map x width
260
264 These extended 32 bit registers contain the width of the specified bit map in pixels where x is one of A, B, or D. The blitter will adjust for pixel size. This must be a positive integer of no more than 14 bits.

BLTLPATRNR 23C R A Blitter Linedraw Pattern Read Register. This register can be polled at the completion of a patterned line draw operation to determine the terminal position of the pattern bits from the BLTPATRNX register. This value can then be re-written into the BLTPATRNX register before starting a new line draw operation so as to allow contiguous pattern line draws across two connected lines.

BLTLPATRNX 448 W A Blitter Linedraw Pattern Register.
This register contains a bit pattern which defines which pixels in the drawn line are actually painted. 1 indicates paint the pixel while 0 indicates don't paint the pixel. Normally, this register is loaded with 0xffffffff.

BLTLPMSKX 450 W A Blitter Linedraw Mask Register.
BLTPMSKX 244 W A Plane Mask Register.
These extended 32 bit registers contain masks which are applied to certain blitter functions when the blitter is being used in chunky pixel mode. The use of these masks permits "per plane" type operations to be performed on chunky data. The value loaded into the registers must be repeated for each possible pixel location within a 32 bit word. For example, if 4 bit chunky pixels (4 planes) are being processed and only the next to last plane is to be affected by the blitter operation, the value 0x22222222 should be placed in the plane mask. If 8 bit chunky pixels are being processed and only the highest plane is to be operated on, the value 0x80808080 should be placed in the plane mask. Normally, 0xffffffff is placed in this register.

BLTxORGX 26C W A Blit x Origin.
270
274 These extended 32 bit registers contain X and Y coordinate values of the upper left hand corner of the rectangles (lower right hand corners for reverse blits) to be blitted, relative to the origins specified in BLTxSRCPTX. x may be one of A, B, or D.

BLTROMD 40C R A Blitter Microcode ROM Data.
31-28 : x (not used)
27 : Flag bit (set when data is ready to be read)
26-0 : Rom output data

BLTSIZE 058 W A Blitter start and size (window width, height)
This register contains the width and height of the blitter operation (in line mode width must =2, height = line length) Writing to this register will start the Blitter, and should be done last, after all pointers and control registers have been initialized. For compatibility purposes, all zeros written to this register cause 1024 x 1024 blits. In

other words, when this register is written with all zeros, w6 of BLTSIZH below is set and all other w bits are reset and h10 of BLTSIZV below is set while all other h bits are reset.

BIT# 15,14,13,12,11,10,09,08,07,06,05,04,03,02,01,00
h9 h8 h7 h6 h5 h4 h3 h2 h1 h0,w5 w4 w3 w2 w1 w0

h=Height=Vertical lines (10 bits=1024 lines max)
w=Width =Horiz pixels (6 bits=64 words=1024 pixels max)

BLTSIZH x 05E W A Blitter H size & start (12 bit width)

Bit#	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
	x	x	x	x	x	w10	w9	w8	w7	w6	w5	w4	w3	w2	w1	w0

BLTSIZV x 05C W A Blitter V size (15 bit height)

Bit#	15	14	12	12	11	10	09	08	07	06	05	04	03	02	01	00
	x	h14	h13	h12	h11	h10	h9	h8	h7	h6	h5	h4	h3	h2	h1	h0

These are the blitter size registers for blits larger than the earlier chips could accept. The original commands are retained for compatibility. BLTSIZV should be written first, followed by BLTSIZH, which starts the blitter. BLTSIZV need not be rewritten for subsequent blits if the vertical size is the same. max size of blit 512k pixels * 1k lines using word blits and 1M pixels * 1k lines using longword blits. x's should be written to 0 for upward compatibility.

BLTSIZEX 268 W A Blit size

This extended 32 bit register contains the size of the rectangle to be blitted. Both the height and width are specified in pixels. The blitter will adjust for pixel size. Max size of blit is 16K pixels by 16K lines.

BIT#	31	30-16	15	14-0
	must be 0	h14-h0	must be 0	w14-w0

BLTTST 480 W A Blitter Microcode ROM Test.

Bit 0 : TEST MODE ENABLE
Bit 1 : LATCHED ROM TEST SELECTED
Bit 2 : LATCHED ROM TEST RESUME (Used for handshaking between tester and internal test circuitry).

Bits 0 & 1 allow for the future possibility of extending this register to control additional test modes.

BPLxDAT (x=1-6) 110 W M Bit plane x data (Parallel to serial convert)

These registers receive the DMA data fetched from RAM by the Bit Plane address pointers described above. They may also be written by either micro.(See SPRxPOS) (Warning- It is not possible to guarantee that the coprocessor will be granted the bus at exactly the same pixel time from display field to field. Therefore, if the coprocessor is being used to write data into these registers in order to creat special effects, it is likely that the screen will exhibit jitter because in some instances the BPLxDAT register write will be delayed beyond the point when data was supposed to have been written.)
These bit plabe data registers act as a 6 word parallel to serial buffer for up to 6 memory "Bit Planes". The parallel to serial conversion is triggered whenever bit plane #1 is written from either the data bus or the video data bus. This indicates the completion of all bit planes for that word (32 pic normal, 16 pic compatibility mode). The MSB is output first, and is therefore always on the left.

BPLxDATX x 300, W M Bit plane x data (Parallel to serial convert)
(x=1-6) 304, These extended 32 bit registers receive data fetched from
308, the LINe DAta buffer, or from RAM. Bits 31-16 are the same
30C, physical register bits that make up BPLxDAT above. These
310, registers act as the Least Significant part of a 16 longword
314 parallel to serial buffer for up to 16 memory "Bit Planes".
When in bitplane mode, the parallel to serial
conversion is triggered whenever BPL1DATX is written.
It is not possible to write chunky pixels into the BPLxDAT
registers from the chip data bus.

BPLxDATN n 508 W M Bit plane x data (Parallel to serial convert)
(x=7-16) These new 32 bit registers act as the most significant
part of the 16 longword parallel to serial buffer described
above. All of the even numbered BPLxDAT registers make up
playfield 2 while the odd numbered BPLxDAT registers make up
playfield 1.

BPLCON0 x 100 W A M Bit plane control reg.(misc control bits)
BPLCON1 102 W M Bit plane control reg.(horiz. scroll control)
BPLCON2 x 104 W M Bit Plane control reg.(video priority control)
BPLCON3 xn 106 W A M Bit Plane control reg.(enhanced features)
BPLCON4 10C W M Bit Plane control reg.

These registers control the operation of the Bit Planes and various aspects of the display. The bits in these registers are held until horizontal count 10 (decimal) before being transferred to working registers. Therefore, these registers can'nt be used to cause mid-line changes to the display attributes. System software calls should be made when it is necessary to write these registers to change the next line display characteristics.

BIT#	BPLCON0	BPLCON1	BPLCON2	BPLCON3	BPLCON4
15	HIRES	x	ZDBPSEL3	LINERPTEN	BPLAM7=0
14	BPU2	x	ZDBPSEL2	OBPLMODE2	BPLAM6=0
13	BPU1	x	ZDBPSEL1	OBPLMODE1	BPLAM5=0
12	BPU0	x	ZDBPSEL0	OBPLMODE0	BPLAM4=0
11	HAM	x	ZDBPEN	PIXCLKSEL2	BPLAM3=0
10	DPF	x	ZDCTEN	PIXCLKSEL1	BPLAM2=0
09	COLOR	x	KILLEHB	PIXCLKSEL0	BPLAM1=0
08	GAUD	x	x	LINERPT1	BPLAM0=0
07	HAM8	PF2H3	SOGEN	LINERPT0	ESPRM7=0
06	x	PF2H2	PF2PRI	BASEN	ESPRM6=0
05	BPU4	PF2H1	PF2P2	BRDBLNK	ESPRM5=0
04	BPU3	PF2H0	PF2P1	BRDNTRAN	ESPRM4=1
03	LPEN	PF1H3	PF2P0	DIR	OSPRM7=0
02	LACE	PF1H2	PF1P2	ZDCLKEN	OSPRM6=0
01	ERSY	PF1H1	PF1P1	LACEDIS	OSPRM5=0
00	ENBPLCN3	PF1H0	PF1P0	x	OSPRM4=1

x= don't care; but drive to 0 for upward compatibility !
HIRES=High resolution(640*200/640*400interlace) mode
BPU =Bit plane use code 00000-10000 (NONE thru 16 inclusive)
When OBPLMODE indicates BITPLANE, this field defines the number of bitplanes used.
When OBPLMODE indicates CHUNKY, PACKLUT, 4-Bit Half Chunky, HALFCHUNKY, 2-Bit Half Chunky, or PACKHY pixel modes, this field should be set to 1.
When OBPLMODE indicates HYBRID pixel mode, this field must be set to 5.

HAM=Hold and Modify mode. Provides compatible HAM mode. Bottom 16 registers are addressed by planes 1 thru 4 or most significant 4 bits of LUT color is modified.
HAM8=10 plane Hold and Modify mode. Allows planes 3 thru 10 to be used to address all color registers, or all of the chosen LUT color to be modified. Planes 1 and 2 provide the HAM control data. Not all 10 bitplanes are required to use this HAM mode. BPU specifies the number of active bitplanes. Unused planes are driven low by the hardware. If both HAM and HAM8 bits are set, then HAM is ignored.
DPF=Double playfield (PF1=odd PF2=even bit planes) (If BPU=6 and HAM=0 and DPF=0 a special mode is defined that allows bitplane 6 to cause an intensity reduction of the other 5 bitplanes. The color register output selected by 5 bitplanes is shifted to half intensity by the 6th bitplane. This is called EXTRA-HALFBRITE Mode.)
COLOR= Composite video COLOR enable
GAUD=Genlock audio enable. This level appears on the ZD pin on Monica during all blanking periods.
LPEN =Light pen enable(reset on power up)
LACE =Interlace enable(reset on power up) Should only be used in conjuction with PAL or NTSC of BEAMCON0.
ERSY =External Resync (HSYNC, VSYNC pads become inputs) (reset on power up)
ENBPLCN3= When set enables some of the new features in BPLCON3. This bit does not mask PIXCLKSEL, LINERPT, or LINERPTEN bits.
PF2Hx= Playfield 2 horizontal scroll code
PF1Hx= Playfield 1 horizontal scroll code
Scroll LSB is 1 pixel @ low res, 2 at high res
ZDBPSELx= 4 bit field which selects which Bit plane is to be used for ZD when ZDBBPEN is set;0000 selects BP1 and 1111 selects BP16.
ZDBPEN= causes ZD pin to mirror bitplane selected by ZDBPSELx bits. This does not disable the ZD mode defined by ZDCTEN, but rather is "ored" with it.
ZDCTEN= causes ZD pin to mirror bit #15 (bit #25 when COLORNn registers are used)of the active color table entry. When ZDCTEN is reset ZD reverts to mirroring color(0).
KILLEHB= disables Extra Half Brite mode.
SOGEN=Sync On Green Enable. When this signal is high, MONICA's analog green signal is merged with the composite sync input. The green voltage level is level shifted accordingly.
PF2PRI= gives Playfield 2 priority over Playfield 1.
PF2Px= Playfield 2 priority code (with resp. to sprites) Note, this field determines playfield priority with respect to sprites when dual-playfield mode is disabled.
PF1Px= Playfield 1 priority code (with resp. to sprites)
LINERPTEN= Display Line Repeat Enable.
LINERPTx= Display Line Repeat Code. When enabled, this code specifies how many times each display line is repeated. When disabled, each line is displayed once.

LINERPT1,0	ACTION
00	Each line is repeated 4 times.
01	Each line is repeated 3 times.
10	Each line is repeated 2 times.
11	Each line is displayed once.

BASEN=Enable OBPLOFF and EBPLOFF of BPLOFF. (reset on power up)
PIXCLKSELx=Select pixel clock frequency. These bits drive

ANDREA outputs FREQCTRL2,1,&0. (reset on power up) They are used to select an appropriate pixel clock for monitor emulation modes. Values in these registers are transferred to the output drivers when the horizontal counter rolls over from the count of HTOTAL. Since other graphics control bits do not take affect until count 10 decimal, there exists a 10 PCLK (low end system, 5 PCLK in the high end system) window in which a write to BPLCON3 could cause all control bits except the PIXCLKSEL bits to take affect on the current scan line and thus possible cause display glitching.
This anomoly can be witnessed under the following conditions:
DMA activity during the previous scan line was such that the COPPER write to this register did not occur until this 10 (5 in dual) PCLK window (this is an overrun condition- COPPER lists should reprogram display control registers between counts 0xA and DDFSTRT for the current scan line, and the new control bit data would take effect on the next scan line);
or if the COPPER is made to wake up at horizontal count 0 and write BPLCON3 first- in this case, there is no problem when a 32-bit DMA COPPER list is being processed (the vertical counter will not match until horizontal count 8), and there is also no problem in 16-bit DMA COPPER lists because these lists would not have known that the PIXCLKSEL field of BPLCON3 existed (NOTE: the PIXCLKSEL field has entirely different meaning to the AA chips. Currently, this problem (different from the problem defined above) must be resolved by system or user software so that a check is done on what chip (AA or AAA) is being written before writting it (ie, look before you leap).

PIXCLKSEL2,1,0	ACTION
000	Native mode. Supply highest frequency clock to PIXCLK.
001	Supply second highest freq. to PIXCLK.
010	Supply third highest freq. to PIXCLK.
011	Supply forth highest freq. to PIXCLK.
100	Supply slowest frequency to PIXCLK.

1280x1K monitor systems:
000= Native 1280x1K pixel clock.
001= Pixel clock required to emulate 1Kx768 display.
010= Pixel clock required to emulate 800x560 display.
011= Pixel clock required to emulate 640x400 display.
100= Pixel clock required to emulate 640x200 display.

1Kx768 monitor systems:
000= Native 1Kx768 pixel clock.
001= Pixel clock required to emulate 800x560 display.
010= Pixel clock required to emulate 640x400 display.
011= Pixel clock required to emulate 640x200 display.
100= Not valid, don't use.

800x560 monitor systems:
000= Native 800x560 pixel clock.
001= Pixel clock required to emulate 640x400 display.
010= Pixel clock required to emulate 640x200 display.
011= Not valid, don't use.
100= Not valid, don't use.

640x400 monitor systems:
000= Native 640x400 pixel clock.
001= Pixel clock required to emulate 640x200 display.
010= Not valid, don't use.
011= Not valid, don't use.
100= Not valid, don't use.

640x200 monitor systems:
000= Native 640x200 pixel clock.
001= Not valid, don't use.
010= Not valid, don't use.
011= Not valid, don't use.
100= Not valid, don't use.

OBPLMODE= Select odd playfield display mode. Data in these these bits is transferred to a working register when the horizontal counter goes from 0 to 1. ANDREA uses the working register to control the fetch of next line buffer data. MONICA pipelines the working register one additional HSYNC pulse and then uses the data to control the bitplane parallel to serial converters. LINDA converts PACKHY display data to HYBRID pixels, and PACKLUT display data to HALFCHUNKY pixels. Therefore, MONICA interprets PACKHY and PACKLUT to be HYBRID and HALFCHUNKY respectively. (reset on power up)

OBPLMODE2-0	meaning
000	Bitplane mode.
001	HYBRID mode.
010	CHUNKY mode.
011	PACKLUT mode.
100	4-Bit Half Chunky.
101	HALFCHUNKY mode.
110	2-Bit Half Chunky.
111	PACKHY mode.

Note, when OBPLMODE indicates HYBRID pixels, the BPU field of BPLCON0 must be set to 5, and the BPL5PTX, BPL3PTX, and BPL1PTX registers are programmed to point the red, green, and blue registers, respectively.

BRDBLNK= "border area" is black instead of color(0).

BRDNTRAN= "border area" is non-transparent(ZD pin is low when border is displayed.

DIR= "Direction". When this bit is set, the chips are in Framegrabber mode and the external video data present on the VDATA bus is captured in LINDA on a line by line basis, then shifted into the serial port of the VRAM's under the control of ANDREA.

ZDCLKEN= ZD pin outputs a derived pixel clock whose rising edge coincides with high-res video data.

When set this bit disables all other ZD functions.

LACEDIS= Interlace disable bit. When this bit is set, changes to the LACE bit of BPLCON0 are not permitted. Since old software may change the LACE bit directly, and since the new chip set has monitor emulation modes which could permit monitor damage if this bit is inadvertenty set, this bit has been provided as a way of protecting the users monitor when he runs old software in monitor emulation mode. System software should always set this bit after the monitor control registers have been properly programmed to disallow further changes to LACE.

BPLAMx= 8 bit field is XOR'ed with the 8 bit bitplane color address, therefore altering the color addresses sent to the color table. This XOR function occurs after the collision circuitry. The XOR does not occur if the HAM modes are enabled. When the value in this field is zero, the EBPLOFF and OBPLOFF fields in the BPLOFF register are used as modulo 256 additive offsets into the color table. If this field is non-zero, EBPLOFF and OBPLOFF is disabled. This field is cleared at reset.

ESPRMx= 4 bit field provides the high order color table

address additive offset bits for even sprites(SPR0,SPR2, SPR4,SPR6). These bits are the same hardware bits as ESPRM7-4 in BPLOFF. When this register is written, bits ESPRM3-0 in BPLOFF are cleared.
This field is set to '0001' at reset.

OSPRMx= 4 bit field provides the high order color table address additive offset bits for odd sprites(SPR1,SPR3, SPR5,SPR7). These bits are the same hardware bits as OSPRM7-4 in BPLOFF. When this register is written, bits OSPRM3-0 in BPLOFF are cleared.
This field is set to '0001' at reset.

BPLOFFR 490 R M Bitplane and Sprite LUT Address Offset read reg.
BPLOFFN 504 W M Bitplane and Sprite LUT Address Offset register.

This register contains offsets into the color register table for the even and odd playfield, and even and odd sprites. Note when the system is not in dual playfield mode, and when BPLAMx of BPLCON4 is zero, the EBPLAM field is used as the bitplane offset.

BIT#	FUNCTION	DESCRIPTION
31-24	ESPRM7-0	This 8 bit field provides the color table address offset bits for even sprites(SPR0, SPR2,SPR4,SPR6). Bits ESPRM 7 thru 4 are the same hardware bits as ESPRM7-4 in BPLCON4. When BPLCON4 is written, bits ESPRM3-0 are cleared. This field is set to '00010000' at reset.
23-16	OSPRM7-0	This 8 bit field provides the color table address offset bits for odd sprites(SPR1, SPR3,SPR5,SPR7). Bits OSPRM 7 thru 4 are the same hardware bits as OSPRM7-4 in BPLCON4. When BPLCON4 is written, bits OSPRM3-0 are cleared. This field is set to '00010000' at reset.
15-8	EBPLOFF7-0	This 8 bit value is added (modulo 256) to the even playfield data just before the color lookup table, but after the collision detect logic. If the BPLAM field of BPLCON4 is non-zero, this offset is not added to the address. Instead, the XOR function provided by BPLAM is generated. If a HAM mode is enabled, this offset is not added to the address. This field is cleared at reset.
7-0	OBPLOFF7-0	This 8 bit value is added (modulo 256) to the odd playfield data just before the color lookup table, but after the collision detect logic. If the BPLAM field of BPLCON4 is non-zero, this offset is not added to the address. Instead, the XOR function provided by BPLAM is generated. If a HAM mode is enabled, this offset is not added to the address. This field is cleared at reset.

BPLOVRDAT 530 W M Overlay Bitplane Data Register.

It is possible to program a DMA channel to load playfield 2 with a single bitplane while loading playfield 1 to display CHUNKY pixels from the line data buffer chip. This provides an overlay playfield for use with CHUNKY pixel displays. To do this, dual playfield mode is enabled as well as one of the

CHUNKY pixel modes. BPL2PT is used to point to the overlay bitplane, and BPL1PT is used to point to the CHUNKY plane (BPL3PT, and BPL5PT are also used when the chunky pixel mode choosen is HYBRID).
(See SPRxPOS for description of pipeline delay attribute.)

BPLnPTH 0E0 W A Bit plane n pointer (High 5 bits)
BPLnPTL 0E2 W A Bit plane n pointer (Low 16 bits, bit 0 forced low)
This pair of registers contains the 20 bit pointer to the address of Bit plane n (n=1,2,3,4,5,6) DMA data. This pointer must be reinitalized by the processor or coprocessor to point to the beginning of Bit Plane data every vertical blank time. 32 bit transfers(or 64 bit transfers if highend system) are always done from the address pointed to by this register(A1,0 (and A2 if 64 bits are being transferred) are forced low. LINDA will perform the appropiate word alignment before display.) Normally, The LINe DAta buffer serves as the destination of bitplane data transfers. However, when the display format is a CHUNKY type and dual playfields are enabled, overlay bitplane data will be written into MONICA's BPLxDAT registers. The manner by which data is transferred into the line buffer will vary depending on the type of RAM being addressed. ANDREA will modify the transfer mechanism to support the RAM being addressed. See description for RAMnATN registers.
Past revisions of the chip set permitted mid-line changes to these registers to allow horizontal split screens. Now, the chip set contains unaccessable double buffered working registers which are prebuffered at horizontal count 10 decimal, then loaded into the final working register at DDFSTRT time. Mid-screen changes made to these registers will not cause video data fetches from the location written into these registers until the following display line.

BPLnPTX x 3E0, 3E4, 3E8, 3EC, 3F0, 3F4 W A Extended Bit plane n pointer (24 bits, bits 1,0 forced low)(n=1 thru 6)These registers are the same as BPLnPTH and BPLnPTL above except they extend bitplane addressing capabilities to 16Mbytes. Whenever BPLnPTL or BPLnPTH are written, bits 23-21 of BPLnPTX are cleared.

BPLnPTN n 5D8-5FF W A New Bitplane n pointer (24 bits, bits 1,0 forced low)
These are the new bitplane pointer registers for bitplanes 7 thru 16.

BPL1MOD 108 W A Bit plane modulo (odd planes)
BPL2MOD 10A W A Bit Plane modulo (even planes)
These registers contain the Modulos for the odd and even bit planes. A Modulo is a number that is automatically added to the address at the end of each line, in order that the address then points to the start of the next line. Since they have separate modulos, the odd and even bit planes may have sizes that are different from each other, as well as different from the Display Window size. Note- These registers are double buffered. Values written into these registers are latched into a secondary holding register at horizontal count 10 decimal, then loaded into the final working register at DDFSTRT time. Therefore, data written into these registers will not effect the display until the next lines' DDFSTRT event.

BRSTSTRT 568 W A Burst start. Used in NTSC or PAL modes to program

the start of the burst interval. This register should be programmed relative to HSYNC start to obtain spec NTSC or PAL. BURST is blanked during the vertical equalization lines, and when beamen is not set. (See HTOTALX for bits.)

BRSTSTOP 56C W A Burst stop. Used in NTSC or PAL modes to program the end of the burst interval. (See HTOTALX for bits.)

CLCNTR 584 W A Compatibility mode (PAL/NTSC) line center. Same as HCENTER. Horizontal position to triggers the VHALF bit of the vertical counter. (See HTOTALX for bits.)

CLXCON 098 W M Collision control

This register controls which Bitplanes are included (enabled) in collision detection, and their required state if included. It also controls the individual inclusion of odd numbered sprites in the collision detection, by logically ORing them with their corresponding even numbered sprite. When this register is written, bits ENBP7-ENBP24 and bits MVBP7-MVBP24 of CLXCONX are cleared.

BIT#	FUNCTION	DESCRIPTION
15	ENSP7	Enable Sprite 7 (ORed with Sprite 6)
14	ENSP5	Enable Sprite 5 (ORed with Sprite 4)
13	ENSP3	Enable Sprite 3 (ORed with Sprite 2)
12	ENSP1	Enable Sprite 1 (ORed with Sprite 0)
11	ENBP6	Enable Bit Plane 6 (Match reqd. for collision)
10	ENBP5	Enable Bit Plane 5 (Match reqd. for collision)
09	ENBP4	Enable Bit Plane 4 (Match reqd. for collision)
08	ENBP3	Enable Bit Plane 3 (Match reqd. for collision)
07	ENBP2	Enable Bit Plane 2 (Match reqd. for collision)
06	ENBP1	Enable Bit Plane 1 (Match reqd. for collision)
05	MVBP6	Match value for Bit Plane 6 collision
04	MVBP5	Match value for Bit Plane 5 collision
03	MVBP4	Match value for Bit Plane 4 collision
02	MVBP3	Match value for Bit Plane 3 collision
01	MVBP2	Match value for Bit Plane 2 collision
00	MVBP1	Match value for Bit Plane 1 collision

Note; Disabled Bit Planes cannot prevent collisions. Therefore if all bitplanes are disabled, collisions will be continuous, regardless of the match values. These registers are copied to working registers at each horizontal count 10 decimal. Therefore, data written into these registers will not effect the display until the next horizontal count 10 event.

CLXCONX x 298 W M Extended Collision control

This register has the same function as CLXCON. This register is included to control collisions involving the extended bitplanes 7 thru 16 when OBPLMODE indicates bitplane mode. and to control the even playfield when OBPLMODE indicates a chunky pixel type. Note that writing CLXCON causes the ENBP7-16 and MVBP7-16 bits to be cleared. Therefore, CLXCONX should always be written after writing the ENSP bits of CLXCON.

BIT#	FUNCTION	DESCRIPTION
31-16	ENBP16-1	Enable Bit Planes 16-1 (Match reqd. for collision)
15-00	MVBP16-1	Match value for Bit Planes 16-1 collision

CLXCONOEN n 494 W M CHUNKY Pixel Collision control: Enable bits.

BIT#	FUNCTION	DESCRIPTION
23-00	ENC24-1	Enable CHUNKY bits 24-1 (Bits 24-9 should be cleared when displaying HALFCHUNKY pixels. Bits 24-17 should be cleared when displaying CHUNKY pixels.)

CLXCONODN n 498 W M CHUNKY Pixel Collision control: Match value.

BIT#	FUNCTION	DESCRIPTION
23-00	MVC24-1	Match value for CHUNKY bits 24-1.

CLXDAT 00E R M Collision data reg.(Read and clear)

This address reads (and clears) the collision detection register. The bit assignments are below.

NOTE: Playfield 1 is all odd numbered enabled bit planes. Playfield 2 is all even numbered enabled bit planes

BIT#	COLLISIONS REGISTERED
15	not used
14	Sprite 4 (or 5) to Sprite 6 (or 7)
13	Sprite 2 (or 3) to Sprite 6 (or 7)
12	Sprite 2 (or 3) to Sprite 4 (or 5)
11	Sprite 0 (or 1) to Sprite 6 (or 7)
10	Sprite 0 (or 1) to Sprite 4 (or 5)
09	Sprite 0 (or 1) to Sprite 2 (or 3)
08	Playfield 2 to Sprite 6 (or 7)
07	Playfield 2 to Sprite 4 (or 5)
06	Playfield 2 to Sprite 2 (or 3)
05	Playfield 2 to Sprite 0 (or 1)
04	Playfield 1 to Sprite 6 (or 7)
03	Playfield 1 to Sprite 4 (or 5)
02	Playfield 1 to Sprite 2 (or 3)
01	Playfield 1 to Sprite 0 (or 1)
00	Playfield 1 to Playfield 2

COLORxx 180 W M Color table xx

The old 32 word color palette has been left intact for software compatibility purposes. They contain 12 bit codes representing RED,GREEN,BLUE colors for RGB systems. Bits R3-R0, G3-G0, and B3-B0 are the same as bits R7-R4, G7-G4, and B7-B4 of the first 32 entries of COLORN. When one of these registers is written, bits R3-R0, G3-G0, and B3-B0 of the corresponding entry of COLORN are duplicated. (Warning- It is not possible to guarantee that the coprocessor will be granted the bus at exactly the same pixel time from display field to field. Therefore, if the coprocessor is being used to write data into these registers in order to display more colors, it is likely that the screen will exhibit jitter because in some instances the COLOR register write will be delayed beyond the point when data was supposed to have been written.)

The table below shows the color register bit usage.

BIT#	15,14,13,12,	11,10,09,08,	07,06,05,04,	03,02,01,00
RGB	T X X X	R3 R2 R1 R0	G3 G2 G1 G0	B3 B2 B1 B0

T=Transparency, B=blue, G=green, R=red.

COLORNxxx n C00 W M Color table xx

These registers make up a 256 entry Color Lookup Table.

(See above description of the COLOR registers for the mapping from COLOR registers to COLORN registers.) These registers contain 24 bit codes representing RED,GREEN, and BLUE colors for RGB systems. When bitplane data or HALFCHUNKY pixels emerge from the video prioritizer circuits, it is used to address one of these registers for presentation at the RGB video outputs. When CHUNKY pixels emerge from the video prioritizer, the COLORN registers are bypassed and the pixel information is presented directly to the RGB outputs.

The table below shows the color register bit usage.

BIT#	31,	30-24,	23-16,	15-08,	07-00
RGB	T	X	R7-R0	G7-G0	B7-B0

T=Transparency, B=blue, G=green, R=red.

COLORNxxx n 800 R M Color table xx
These registers are the same as COLORNxxx, except provide read addresses for the color registers.

COPCON 02E W A Coprocessor control register
This is a 1 bit register that when set true, allows the Coprocessor to access the Blitter hardware. This bit is cleared by power on reset, so that the Coprocessor cannot access the Blitter hardware.

BIT#	NAME	FUNCTION
01	CDANG	Coprocessor danger mode. Allows coprocessor access to all RGA registers if true. (if 0, access to RGA > 07E < 200, RGA > 27C < 400, and RGA > 48C)

COPJMP1 088 S A Coprocessor restart at first location
COPJMP2 08A S A Coprocessor restart at second location
COPJMP3 n 538 S A Coprocessor restart at third location
COPJMP4 n 53C S A Coprocessor restart at forth location

These addresses are strobe addresses, that when written to cause the Coprocessor to jump indirect using the address contained in the First or Second Location registers described below. The Coprocessor itself can write to these addresses, causing it's own jump indirect.

COP1LCH 080 W A Coprocessor first location reg (High 5 bits)
COP1LCL 082 W A Coprocessor first location reg. (Low 16 bits,A0 forced low)
COP2LCH 084 W A Coprocessor second location reg. (High 5 bits)
COP2LCL 086 W A Coprocessor second location reg(Low 16 bits,A0 forced low)
COP1LCX x 280 W A Extended Coprocessor first location reg. (24 bits, A1 and A0 forced low)
COP2LCX x 284 W A Extended Coprocessor second location reg. (24 bits, A1 and A0 forced low)
COP3LC n 540 W A Coprocessor third location reg(24 bits, A1, A0 forced low)
COP4LC n 544 W A Coprocessor forth location reg(24 bits, A1, A0 forced low)

These registers contain the jump addresses described above. Note when COP1LCH or L or COP2LCH or L are written, address bits 23 thru 21 of the corresponding COPnLCX registers are cleared.

COPBLITLC n 548 W Coprocessor Blitter interrupt routine starting address. Must start on even longword boundary.

COPRETC n 554 S A Coprocessor return from blitter interrupt strobe.
COPRET n 54C S A Coprocessor Return from Interrupt
When written, these strobe addresses cause the Coprocessor to execute a return from interrupt sequence by popping program counter, status, COP3LC, and COP4LC register data from the Coprocessor stack. This information is initally pushed onto the stack when a coprocessor interrupt is recognized. All Coprocessor lists should finish by writting to this address. The COPRETC address differs from the COPRET address in that it also clears the BLIT bit of COINTRER when strobed.

COPWAIT n 534 W A Coprocessor wait interrupt routine starting address.
This register is automatically updated during normal copper operation. It is used to hold the address of the intruction immediately following the last wait instruction executed. The processor should never write this register.

COINTREW n 550 W A Coprocessor Interrupt Request/Enable Write register.
COINTRER n 404 R A Coprocessor Interrupt Request/Enable Read register.
This register contains coprocessor interrupt request bits. It may be read and written by the processor. System events may set request bits within this register and if enabled by the corresponding enable bits, they may cause coprocessor interrupts.

BIT#	FUNCT	LEVEL	DESCRIPTION
31	SET/CLR		Set/Clear control bit. Determines if bits written with a 1 get set or cleared. Bits written with a 0 are always unchanged.
30-16	x	-	Should be set low.
16	BLITRQ	1	Blitter Finished Interrupt Request Flag.
15-09	x	-	Should be set low.
08	WAITF		Indicates a WAIT 'forever' instruction is being executed by the coprocessor. (This bit cannot be written with COINTRQW.)
07-01	x	-	Should be set low.
00	BLIT	1	Blitter Finished.

COPINS x 08C W A Coprocessor inst. fetch identify
This is a dummy address that is generated by the Coprocessor whenever it is loading instructions into it's own instruction register.

MOVE Move immediate to dest
WAIT Wait until beam counter is equal to, or greater than. (keeps Coprocessor off of bus until beam position has been reached)
SKIP Skip if beam counter is equal to, or greater than. (skips following instruction if beam position has been reached)

Instructions executed when COPEN of DMACON is set

	MOVE		WAIT UNTIL		SKIP IF	
BIT#	IR1	IR2	IR1	IR2	IR1	IR2
15	X	RD15	VP7	BFD *	VP7	BFD *
14	X	RD14	VP6	VE6	VP6	VE6

13	X	RD13	VP5	VE5	VP5	VE5
12	X	RD12	VP4	VE4	VP4	VE4
11	DA11	RD11	VP3	VE3	VP3	VE3
10	DA10	RD10	VP2	VE2	VP2	VE2
09	DA9	RD09	VP1	VE1	VP1	VE1
08	DA8	RD08	VP0	VE0	VP0	VE0
07	DA7	RD07	HP8	HE8	HP8	HE8
06	DA6	RD06	HP7	HE7	HP7	HE7
05	DA5	RD05	HP6	HE6	HP6	HE6
04	DA4	RD04	HP5	HE5	HP5	HE5
03	DA3	RD03	HP4	HE4	HP4	HE4
02	DA2	RD02	HP3	HE3	HP3	HE3
01	DA1	RD01	HP2	HE2	HP2	HE2
00	0	RD00	1	0	1	1

Instructions executed when COPENX of DMACONX is set
(These instructions must be on an even word address.)

	MOVE		WAIT UNTIL		SKIP IF	
BIT#	IR1	IR2	IR1	IR2	IR1	IR2
31	X	RD31	X	GTDIS	X	X
30	X	RD30	X	X	X	X
29	X	RD29	X	X	X	X
28	X	RD28	X	X	X	X
27	X	RD27	X	X	X	X
26	X	RD26	VP10	VE10	VP10	VE10
25	X	RD25	VP9	VE9	VP9	VE9
24	X	RD24	VP8	VE8	VP8	VE8
23	MM7	RD23	VP7	VE7	VP7	VE7
22	MM6	RD22	VP6	VE6	VP6	VE6
21	MM5	RD21	VP5	VE5	VP5	VE5
20	MM4	RD20	VP4	VE4	VP4	VE4
19	MM3	RD19	VP3	VE3	VP3	VE3
18	MM2	RD18	VP2	VE2	VP2	VE2
17	MM1	RD17	VP1	VE1	VP1	VE1
16	MM0	RD16	VP0	VE0	VP0	VE0
15	X	RD15	X	X	X	BFD *
14	X	RD14	X	X	X	X
13	X	RD13	X	X	X	X
12	X	RD12	X	X	X	X
11	DA11	RD11	X	X	X	X
10	DA10	RD10	HP9	HE9	HP9	HE9
09	DA9	RD09	HP8	HE8	HP8	HE8
08	DA8	RD08	HP7	HE7	HP7	HE7
07	DA7	RD07	HP6	HE6	HP6	HE6
06	DA6	RD06	HP5	HE5	HP5	HE5
05	DA5	RD05	HP4	HE4	HP4	HE4
04	DA4	RD04	HP3	HE3	HP3	HE3
03	DA3	RD03	HP2	HE2	HP2	HE2
02	DA2	RD02	X	X	X	X
01	DA1	RD01	X	X	X	X
00	0	RD00	1	0	1	1

IR1=First instruction register
IR2=Second instruction register
DA =Destination Address for MOVE instruction. Fetched during IR1 time, used during IR2 time to address register.
MM =Multiple Move. This field defines the number of longwords plus 1 following this instruction to be moved to the address sequential registers starting with the register pointed to by DA.

RD =RAM Data moved by MOVE instruction at IR2 time directly from RAM to the address given by the DA field.
VP =Vertical Beam Position comparison bit
HP =Horizontal Beam Position comparison bit
VE =Enable comparison(mask bit)
HE =Enable comparison(mask bit)
GTDIS =Greater Than comparison disable. If this bit is set, then wait instruction comparisons will only occur when the masked compare value equals the line counters.
* NOTE BFD=Blitter finished disable. When this bit is true, the Blitter Finished flag will have no effect on the Coprocessor. When this bit is zero the Blitter Finished flag must be true (in addition to the rest of the bit comparisons) before the Coprocessor can exit from its wait state, or skip over an instruction.

The Coprocessor is basically a 2 cycle machine. It has priority over only the Blitter and Micro.

There are only three types of instructions, MOVE immediate, WAIT until ,and SKIP if. All instructions require 2 bus cycles however, since the Coprocesser has lower bus priority than bitplane, sprite, disk, and audio DMA, a variable number of cycles may be required to complete the execution of an instruction.

There are two indirect jump registers COP1LC and COP2LC. These are 23 bit pointer registers whose contents are used to modify the program counter for initalization or jumps. They are transfered to the program counter whenever strobe addreses COPJMP1 or COPJMP2 are written. In addition COP1LC is automatically used at the beginning of each vertical blank time.

It is important that one of the jump registers be initalized and it's jump strobe address hit, after power up but before Coprocessor DMA is initalized. This insures a determined startup address, and state.

When using the COPPER to modify display control registers mid-screen, avoid programming the related WAIT instruction to wait for horizontal count 0 of the scan line when the changes are to be made. This count should be avoided for several reasons: 1)in genlock mode, horizontal count 0 sometimes does not occur (count 0 is skipped and count 4 is the first count of the line); 2)the vertical counter does not increment at horizontal count 0. It increments at count 8 instead (count 2 in previous chip sets, (given at the resolution available to the previous chip sets); 3)it would be possible to update 1 or 2 control registers which will take effect on the current scan line, but the remainder control register writes would not take effect until the next scan line (most display control registers are transferred to working registers at horizontal count 0xa.

DDFSTRT x 092 W A Display data fetch start(Horiz.Position)
DDFSTOP x 094 W A Display data fetch stop(Horiz.Position)

These registers control the horizontal timing of the beginning and end of the Bit Plane DMA display data fetch. The vertical Bit Plane DMA timing is identical to the Display windows described above.

The Bit Plane Modulos are dependent on the Bit Plane

horizontal size, and on this data fetch window size.

Register bit assignment

```
BIT#  15,14,13,12,11,10,09,08,07,06,05,04,03,02,01,00
USE   H01  X  X  X  X  X    H9 H8 H7 H6 H5 H4 H3 H2 H1
```

(X bits should always be driven with 0 to maintain upward compatability)

The tables below show the start and stop timing for different register contents when the programmed display mode is bitplane, less than 9 planes.

DDFSTRT(Left edge of display data fetch)

PURPOSE	H9	H8	H7	H6	H5	H4
Extra wide (max) *	0	0	0	1	0	1
wide	0	0	0	1	1	0
normal	0	0	0	1	1	1
narrow	0	0	1	0	0	0

DDFSTOP(Right edge of display data fetch)

PURPOSE	H9	H8	H7	H6	H5	H4
narrow	0	1	1	0	0	1
normal	0	1	1	0	1	0
wide (max)	0	1	1	0	1	1

Note that these numbers will vary with chosen display resolution and pixel mode. Chunky pixel modes are always programmed at DIWSTRT - 40. Also note that these registers are copied to working registers before affecting the display. The new values are loaded into working registers at horizontal count 10 decimal for use in providing the LINDA chip with pixel data output timing. Addressing circuit DDF registers are prebuffered at horizontal count 10 decimal, then transferred to working registers at DDFSTRT time. Therefore, data written into these registers will not effect the display until the appropriate time of the next horizontal scan line.

DIWSTRT 08E W A M Display Window Start (upper left vert-hor pos)
DIWSTOP 090 W A M Display Window Stop (lower right vert-hor pos)

These registers control the Display window size and position, by locating the upper left and lower right corners.

```
BIT#  15,14,13,12,11,10,09,08,07,06,05,04,03,02,01,00
USE   V7 V6 V5 V4 V3 V2 V1 V0 H7 H6 H5 H4 H3 H2 H1 H01
```

(Note: in older versions of the chip set, H01 was referred to as H0.)

DIWSTRT is vertically restricted to the upper 2/3 of the display (V8=0), and horizontally restricted to the left 3/4 of the display(H8=0).*

DIWSTOP is vertically restricted to the lower 1/2 of the display (V8=/=V7), and horizontally restricted to the right 1/4 of the display(H8=1).*

* Poof.. (See DIWHIGH for exceptions)

Note- These registers are copied to working registers at each horizontal count 10 decimal. Therefore, data written into these registers will not effect the display until the next horizontal count 10 event.

DIWHIGH x 1E4 W A M Display Window upper bits for start,stop

This is an added register for the Hires chips, and allows larger start & stop ranges. If it is not written, the above

(DIWSTRT,STOP) description holds. If this register is written last in a sequence of setting the display window, it sets direct start & stop positions anywhere on the screen.

```
Bit#  15 14 13 12 11  10 09 08  07 06 05 04 03  02 01 00
       x  x H8  x  x V10 V9 V8   x  x H8  x  x V10 V9 V8
             (stop)        |          (start)
```

Don't care (x) bits should always be written to 0 to maintain upward compatibility.

DIWSTRTX 3C4 W A M Extended Display Window Start (upper left vert-hor pos)
DIWSTOPX 3C8 W A M Extended Display Window Stop (lower right vert-hor pos)

These extended registers are the same as DIWSTRT, DIWSTOP, and DIWHIGH above except they have been extended to give a clean software method of programming this function. When any of DIWSTRT, DIWSTOP, or DIWHIGH are written, H9 and H00 are reset.

bit31-17	x
bit26-16	V10-V0
bit15-11	x
bit10-0	H9-H1,H01,H00

DMACON 096 W A M P DMA control write(clear or set)
DMACONR 002 R A DMA control (and blitter status) read

This register controls all of the 16 bit DMA channels, and contains Blitter DMA status bits.

BIT#	FUNCTION	DESCRIPTION
15	SET/CLR	Set/Clear control bit. Determines if bits written with a 1 get set or cleared. Bits written with a zero are unchanged.
14	BBUSY	Blitter busy status bit (read only)
13	BZERO	Blitter logic zero status bit.(read only)
12	X	
11	X	
10	BLTPRI	Blitter DMA priority(over CPU micro) (also called "Blitter Nasty")
09	DMAEN	Enable all DMA below
08	BPLEN	Bit Plane DMA enable
07	COPEN	Coprocessor DMA enable
06	BLTEN	Blitter DMA enable
05	SPREN	Sprite DMA enable
04	DSKEN	Disk DMA enable
03	AUD3EN	Audio channel 3 DMA enable
02	AUD2EN	Audio channel 2 DMA enable
01	AUD1EN	Audio channel 1 DMA enable
00	AUD0EN	Audio channel 0 DMA enable

DMACONX x 290 W A M P DMA control write(clear or set)
DMACONRX x 20C R A DMA control (and blitter status) read

This extended register controls all of the DMA channels, and contains Blitter DMA status bits. Unlike most of the other extended registers, not all like bits within this register map to the same hardware bits that are in DMACON. When DMA channels are enabled with this register, 32 bit DMA transfers will be done instead of the 16 bit transfers enabled by DMACON. The exception to this rule are bits 23 thru 20 which specify 16 bit audio transfers. Also, bitplane transfers controlled by BPLEN in DMACON are always at least 32 bit transfers depending on the system and RAM type, therefore the BLTEN bits map to the same register. The programmer must assure that all DMA addresses controlled by this register are even longword addresses. Unpredictable results will occur if a 16 bit DMA channel is enabled at the

same time that the same 32 bit DMA channel is enabled.

BIT#	FUNCTION	DESCRIPTION
31	SET/CLR	Set/Clear control bit. Determines if bits written with a 1 get set or cleared. Bits written with a zero are unchanged.
30-25	X	Not used.
24	COPPRI	Copper DMA priority. Also called copper nasty.(Reset on power up, and whenever copper interrupt routine is processed.)
23	AUD7EN	Audio channel 7 16 bit DMA enable
22	AUD6EN	Audio channel 6 16 bit DMA enable
21	AUD5EN	Audio channel 5 16 bit DMA enable
20	AUD4EN	Audio channel 4 16 bit DMA enable
19	AUD7ENX	Audio channel 7 32 bit DMA enable
18	AUD6ENX	Audio channel 6 32 bit DMA enable
17	AUD5ENX	Audio channel 5 32 bit DMA enable
16	AUD4ENX	Audio channel 4 32 bit DMA enable
15	OVRLAYEN	Overlay Graphics DMA enable
14	BBUSY	Blitter busy status bit (read only, same hardware bit as BBUSY of DMACON)
13	BZERO	Blitter logic zero status bit.(read only, same hardware bit as BZERO of DMACON)
12	PROPRI	Processor priority(over blitter)(set on power up.)
11	BBRSTENX	Enable burst mode fetches for Blitter DMA
10	BLTPRI	Blitter DMA priority(over CPU micro) (Same hardware bit as BLTPRI of DMACON) (also called "Blitter Nasty")(reset on power up.)
09	DMAENX	Enable all DMA enable bits in this register.
08	BPLEN	Graphics data DMA enable
07	COPENX	Coprocessor DMA enable
06	BLTEN	Blitter DMA enable
05	SPRENX	Sprite DMA enable
04	DSKENX	Disk DMA enable
03	AUD3ENX	Audio channel 3 32 bit DMA enable
02	AUD2ENX	Audio channel 2 32 bit DMA enable
01	AUD1ENX	Audio channel 1 32 bit DMA enable
00	AUD0ENX	Audio channel 0 32 bit DMA enable

Meaning of settings of DMA priority bits:

BLTPRI	PROPRI	Resulting DMA priority
0	0	Relative priority between processor and blitter toggles each time the current high priority channel is serviced. Allows bus to be time shared between processor and blitter. (Note the toggle will not occur when current low priority channel is serviced because the current high priority channel was not requesting the bus.)
0	1	Processor has priority over blitter. Chip resets to this state.
1	0	Blitter has priority over processor.
1	1	Blitter has priority over processor.

When the COPPRI (copper nasty) bit is set, the copper will take every cycle it needs and can get. When the COPPRI bit is clear, the copper bus request line is disabled every other cycle, thereby allowing the copper

access to only the 'odd' bus cycles.

DSKBKPT 5A0 W A New disk DMA backup pointer.
Bits 23-0 hold initial pointer for disk dma.

DSKBYTR 01A R P Disk Data byte and status read
This register is the Disk-Microprocessor data buffer. Data from the disk (in read mode) is loaded into this register one byte at a time, and bit 15 (DSKBYT) is set true.

BIT#		
15	DSKBYT	Disk byte ready (reset on read)
14	DMAON	DMAEN(DSKLEN) & DMAEN(DMACON) & DSKEN(DMACON)
13	DISKWRITE	Mirror of bit 14 (WRITE) in DSKLEN
12	WORDEQUAL	This bit true only while DSKSYNC register equals the data from disk
11-08	0	Not used
07-00	DATA	Disk byte data

DSKBYTRX x 214 R P Disk Data byte and status read
This register is the Disk-Microprocessor data buffer. Data from the disk (in read mode) is loaded into this register one word at a time, and bit 31 (DSKBYT) is set true. DMAON, DISKWRITE, WORDEQUAL and bits 7-0 of DATA are the same bits as the equivelent bits in DSKBYTR register.

BIT#		
31	DSKBYT	Disk byte ready (reset on read)
30	DMAON	DMAEN(DSKLEN) & DMAENX(DMACONX) & DSKENX(DMACONX)
29	DISKWRITE	Mirror of bit 14 (WRITE) in DSKLEN
28	WORDEQUAL	This bit true only while DSKSYNC register equals the data from disk
27	DSKBUSY	Controller not yet in state 0
26	CRCERR	CRC error on read
25	NOTFOUND	Gap timeout or match failure on read
24	x	unused, 0 returned
23-16	DATA	The actual byte data
15-00	x	unused, 0 returned

DSKCRCI 590 W P Initial CRC value register

31-18	x unused, set to 0 for upward compatibility.
17-16	RLLI. Initial state of RLL encode/decode. On write, encode state machine is set to this value during raw or sync data. On read set to this value at match with sync register.
15-00	Initial value of CRC register. The CRC is reloaded with this at the raw->encoded data boundary (or sync->encoded boundary in TRACKPLUS mode (==3)). If some of the raw data is included in the CRC on the disk, start with a value other than FFFF. eg. on MFM, the A1A1A1FB at the start of data will be raw data header matching. The CRCI register should be started at E295 to make things come out right. RLLI should start at either FFFF or E295. CRC.C is a program to help find the initial value for other circumstances.

DSKDAT 026 W P Disk DMA Data write
DSKDATR 008 R P Disk DMA Data read
This register is the 16 bit Disk-DMA data buffer. It contains 2 bytes of data that are either sent to (write) or received from (read) the disk. The write mode is enabled by the WRITE bit of the DSKLEN or DSKLENX register. The DMA controller automatically transfers data to or from this register and RAM

when DSKEN of DMACON is set. When the DMA transfer is finished (Length=0) a Disk Block Interrupt is generated. See interrupts below. (This was a dummy early read address in past revisions of the chip. New bus timing precludes the need for special early read cycles.)

DSKDATX x 228 W P Disk DMA Data write
DSKDATRX x 208 R P Disk DMA Data read

This register is the 32 bit Disk-DMA data buffer. It contains 4 bytes of data that are either sent to (write) or received from (read) the disk. The write mode is enabled by the WRITE bit of the DSKLEN or DSKLENX register. The DMA controller automatically transfers data to or from this register and RAM when DSKENX OF DMACONX is set. When the DMA transfer is finished (Length=0) a Disk Block Interrupt is generated. See interrupts below.

DSKGAP 594 W P Disk Gap Timer register. Register used to time out (in encoded bytes) gaps in IBM disk format. Also includes sector count.

31-18	x	Not used. Set to 0 for upwards compatibility
17-12	SECTCNT	Sector count. (no backup register- can be rewritten any time)
11-08	x	Not used. Set to 0 for upwards compatibility
07-00	GAPLEN	Gap Length. (has backup register) read gap must be greater than the byte cnt starting after the addr mark match data and the last byte counted is the sync mark for the data header. write gap can be calculated as follows: (gap*16) raw bits (mode 1 only) + 12 1/2 raw bits + (6 C14 ticks + 4 CPLL ticks) rounded up to next integral # of bits + 12 CPLL ticks + 3 C14 ticks

ex: (4M disk @ standard speed, fasta=fastb=1, gap==0)

```
12 1/2 bits (bit times)
 1     bit  (rounded up ticks)
 .96   bits (CPLL & C14 ticks)
------
14.46  bits
   (In normal situations and normal
    accuracy, use two bytes.)
```

DSKHDPT 59C W A New disk DMA header pointer.

DSKLEN 024 W P Disk length

Write twice to start, 0 when done.
This register contains the length (number of words) of Disk DMA data. It also contains 2 control bits. These are a DMA enable bit, and a DMA direction(read/write) bit.

BIT#		
15	DMAEN	Disk DMA Enable
14	WRITE	Disk Write(RAM to Disk) if 1
13-0	LENGTH	Length (# of words) of DMA data.

DSKLENX x 598 W P Disk length

Write twice to start, 0 when done.

This register contains the length (number of words) of Disk DMA data. It also contains 2 control bits. These are a DMA enable bit, and a DMA direction(read/write) bit.

BIT#		
31	DMAEN	Disk DMA Enable
30	WRITE	Disk Write(RAM to Disk) if 1
29-24	LEN1	2 byte increments for LEN1 stuff (see mode desc-mostly hdr)
23-22	x	Not used- write 0 for upward compatibility
21-16	LEN2	2 byte increments for LEN2 stuff (see mode desc-mostly hdr)
15-00	LEN3	2 byte increments for LEN3 stuff (see mode desc-mostly data)

DSKPLLF x 224 W P Disk Phase Lock Loop Frequency constants adjust.

This register allows adjustments to the PLL frequency constants so that disk performance can be optimized in non-standard applications. This register has been added primarily to support CD ROMs and secondarily to permit tweaks should problems be experienced in the 4Meg environment. This register will not be advertised in order to curtail the generation of new disk protection schemes.

31-28,15-12 WRITEPER period for raw bit time on write in 1/2 CPLL tick steps. CPLL is 28M/4 if fastb==0, 28M if fastb==1;

FASTB	FASTA	raw bit period
0	0	WRITEPER= (ticks of 28M/2) desired
0	1	WRITEPER= (ticks of (int(28M/2)))*2
1	0	WRITEPER= (ticks of 28M*2)
1	1	WRITEPER= (ticks of 28M*2)*2

another way to look at it:

FASTB	FASTA	raw bit period
0	0	WRITEPER= (ticks of 28M/2)
0	1	WRITEPER= same number is 2* faster than above(00)
1	0	WRITEPER= same number is 2* faster than above(01)
1	1	WRITEPER= same number is 2* faster than above(10)

if not VARYPLL, 56 (0x38) is the number used (56 ticks at slowest speed)

27-16 FREQMAX max freq
11-0 FREQMIN min freq

@fasta==0, freq#=16384/(CPLL ticks desired)
@fasta==1, freq#= 8192/(CPLL ticks desired)

On chip reset, freq is set initially to 584 (0x248)

note: DSKPLLF should be written before DSKPLLP
if not VARYPLL max- 644 (0x284)
min- 528 (0x210)

If new values of max and min are set, and the current freq is outside this range, and TSTRESET is clear, then the PLL will move into the range at a rate of 4 notchs/CPLL tick (FASTB=0/1->140/35ns) at freqgentle=0; at freqgentle=1, it is 2 notches/tick, at freqgentle=2, 1 notch/tick. DSKPLLR can also be used to check when it's there.
If DSKPLLP is written with TSTRESET set, then the frequency is set to FMIN immediately.

DSKPLLP x 220 W P Disk Phase Lock Loop Phase constants adjust.

This register will not be advertised in order to curtail the

generation of new disk protection schemes.

31- TSTRESET zeros the sum of the PLL (for chip test purposes)

30- FREQGENTLE1 amount of freq feedback

29- FREQGENTLE0

FREQGENTLE1	FREQGENTLE0	RESULT
0	0	same as power up standard (~8 bits of 1 to settle)
0	1	2* less (greater lock accuracy, ~16 bits of 1 to settle)
1	0	4* less (much greater lock accuracy, ~32 bits of 1 to settle)
1	1	same as 2

The settling time depends on many things- These conditions are a +-9% lock range worst case settling at otherwise standard powerup conditions.

28- VARYPLL Allows variable PLL parameters to take effect (as long as DSKEN (old mode) is zero)

27-16 CORRPOS Positive phase correction

12- READSUM On DSKPLLR, read sum instead of frequency (test purposes)

11-0 CORRNEG Negative phase correction

CORRPOS & CORRNEG should be approx.(from expected center freq)

+-450 at freqgentle==0
+-300 at freqgentle==1
+-200 at freqgentle==2

more or less- check response under conditions of interest with the PLL simulator (DSKPLLG.C) to check for stability if not VARYPLL: CORRPOS- 1020 (0x3FC), CORRNEG- 124 (0x7C)
(also chip reset-> these values)
DSKPLLP should be written after DSKPLLF.
Standard PLL settings using variable write:

```
wrw   DSKPLLF  0x32848210
wrw   DSKPLLP  0x13FC007C
```

(28 ticks of CPLL/window at fasta= 0)
(14 ticks of CPLL/window at fasta= 1)

See 'Setting the Disk Phase Locked Loop Parameters' section of AAA specification for more information on these bits.

DSKPLLR x 230 R P Disk Phase Lock Loop frequency read register. (or sum in READSUM mode (for chip test))

	!READSUM	READSUM
31-24	0	0 (not used)
23-20	0	sum5- upper 4 bits of half sum
19-16	0	sum4- lower 4 bits of half sum
15-12	0	sum3- bits 15-12 of sum
11-08	freq2 (11-8)	sum2- bits 11-8
07-04	freq1 (7-4)	sum1- bits 7-4
03-00	freq0 (3-0)	sum0- bits 3-0

Note that there is a pipeline every 4 bits in the PLL adder, so the sum/freq read at a given timepoint is actually results from several time points mixed: e.g.freq count down

512= #200
511 1FF

```
                        510   1FE  results without pipeline
                              1FE
                              1FE

                             #200
                             #20F
                             #2FE
                             #1FE  results with pipeline
                              1FE
```

This register can be used to test lock under normal circumstances, and to test the sum for faster chip testing in that case.

DSKSYNC 07E W P Disk sync register, holds the match code for disk read synchronization. See ADKCON bit 10

DSKSYNCX 27C W P Disk sync register, holds the longword match code for disk read synchronization. See ADKCONX bit 11 Use lower bits to match if SYNCEN is not set for full 32 bits.

DSKTST 5A4 W P Disk test register.

```
15  x
14  x
13  x
12  dsktstrd.  When set, the following information can be
    read from the DSKBYTR register:
          15   dkstate 3
          14   dkstate 2
          13   dkstate 1
          12   dkstate 0   (dkstate 0 thru 3 indicate the
                            state of the disk state machine)
          11   lenfin
          10   gapend
          09   sectend
11  secttstcnt -- count sector register once when writing
                  register with this bit high
10  x
 9  x
 8  x
 7  gaptstld   -- load gap counter from gap register
 6  gaptstcnt  -- count gap register once when this bit is
    written
 5  gapfc2     -- force a carry into bit 8 of the gap cntr
                  so that the higher bits count faster than
                  normal
 4  gapfc1     -- force a carry into bit 4 of the gap cntr
 3  lentstcnt  -- count the length counter once
 2  lenfc3     -- force carry into bit 12 of length counter
 1  lenfc2     -- force carry into bit  8 of length counter
 0  lenfc1     -- force carry into bit  4 of length counter
```

All bits are reset on chip reset or startup of disk controller.

```
DSKPTH  020  W  A   Disk pointer (High 5 bits)
DSKPTL  022  W  A   Disk pointer (Low 16 bits, bit 0 forced low)
```

This pair of registers contains the 20 bit word address of Disk DMA data. These address registers must be initalized by the processor or coprocessor before disk DMA is enabled. If DSKEN of DMACON is enabled, 16 bit word transfers from the

address pointed to by this register will be done with each disk DMA request. If DSKENX of DMACONX is enabled, 32 bit longword transfers from the address pointed to by DSKPTX (A1 and A0 forced low) will be done with each disk DMA request.

DSKPTX 234 W A Extended disk pointer register.
This register is the same as DSKPTH&L above except it extends disk addressing capabilities to 16Mbytes. Whenever DSKPTL or DSKPTH are written, bits 23-21 of DSKPTX are cleared.

EQU1STOP 570 W A First equalization pulse stop position.
This register contains the horizontal beam position required to deassert the first equalization pulse (EQU1). The EQU1 pulse is always set at the HSSTRT beam position.
See HTOTALX for bit positions.
Required value for both NTSC and PAL: EQU1STOP = 070 (hex)

EQU2STOP 574 W A Second equalization pulse stop position.
This register contains the horizontal beam position required to deassert the second equalization pulse (EQU2). The EQU2 pulse is always set at the HCENTER beam position.
See HTOTALX for bit positions.
Required value for both NTSC and PAL: EQU2STOP = 236 (hex)

GENERALW 560 W A General purpose register, write.
GENERALR 424 R A General purpose register, read.
This is a general purpose 32 bit register. Normally, system software (or the COPPER) will store a blitter operation code in this register to allow tracking of the last completed blit operation when the COPPER interrupt circuits are used to restart the blitter.

GRAPHICS 500 W L Graphics fetch cycle. Indicates current cycle is a graphics fetch cycle to transfer data to LINDA.

HBSTOP 1C6 W A Horiz line position for HBLANK stop
HBSTRT 1C4 W A Horiz line position for HBLANK strt
These are the start,stop positions for the HBLANK that comes out on the CBLK* pin.
(See HTOTAL for bits.)
Note- These registers are copied to working registers at each horizontal count 10 decimal. Therefore, data written into these registers (as well as HCENTER, HSSTOP, HSSTRT, and HTOTAL below) will not effect the display until the next horizontal count 10 event.

HBSTOPX 3CC W A Extended Horiz line position for HBLANK stop
HBSTRTX 3D0 W A Extended Horiz line position for HBLANK strt
These are the same as HBSTRT and HBSTOP above, except they have been extended for more resolution and range.
(See HTOTALX for bits.)

HCENTER 1E2 W A Horizontal position of VSYNC on long field
This is necessary for interlace mode.
(See HTOTAL for bits.)

HCENTERX 3D4 W A Extended Horizontal position of VSYNC on long field
(See HTOTALX for bits.)

HHPOSR 1DA R A Horizontal beam counter read
HHPOSW 1D8 W A Horizontal beam counter write

This is the horizontal beam counter. The counter is writable for test purposes only and should never be written. Comparisons for HBSTRT,STOP,HTOTAL,HSSTRT,HSSTOP are made against this counter. This register latches with the lightpen when lightpen is enabled.
(See HTOTAL for bits)

HSSTOP 1C2 W A Horiz line position for HSYNC stop
Sets # of lowres pixels / 2 for sync stop (HTOTAL for bits)

HSSTRT 1DE W A Horiz line position for HSYNC start
Sets # of lowres pixels / 2 for sync start (HTOTAL for bits)

HSSTOPX 3D8 W A Extended Horiz line position for HSYNC stop
HSSTRTX 3DC W A Extended Horiz line position for HSYNC start
These are the same as HSSTRT and HSSTOP above, except they have been extended for more resolution and range.
(See HTOTALX for bits.)

HTOTAL 1C0 W A Highest color clock count in horiz line

```
Bit# 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
       x  x  x  x  x  x  x  x h8 h7 h6 h5 h4 h3 h2 h1
```

(x's should be driven to 0 for upward compatibility)
Horiz line has this many + 1 280ns increments(pairs of lowres pixels) This register format has been retained for backwards compatibility. For chip test purposes, both this register and HTOTALX are preset to all ones when reset is asserted, but only when BEAMLCK of BEAMCON is a 0.

HTOTALX x 3C0 W A Extended horiz line count register
This register is the same as HTOTAL above, except it has been give more horizontal resolution and range. Bits h01 and h00 extend the horizontal counter and give it the ability to count high resolution pixels, while h9 extends the horizontal counter range to support high resolution monitors. Note that in the old systems(A500,A2000) h01 was referred to as h0. A horizontal line contains HTOTALX + 1 highres pixels. Timing constraints in dual chip systems demand h00 always be set to one in all system configurations such that horizontal lines always contain an even number of high resolution pixels.

```
HTOTALX Bit# 31-11  10  09 08 07 06 05 04 03 02 01  00
              x     h9  h8 h7 h6 h5 h4 h3 h2 h1 h01 h00
                        |  |  |  |  |  |  |  |
```

The following diagram shows how values written into HTOTAL map into HTOTALX.

```
                        |  |  |  |  |  |  |  |
HTOTAL Bit# 15-8        07 06 05 04 03 02 01 00
              x         h8 h7 h6 h5 h4 h3 h2 h1  -  -
```

INTREQ 09C W P Interrupt Request bits (clear or set)
INTREQR 01E R P Interrupt request bits (read)
This register contains interrupt request bits(or flags). These bits may be polled by the processor, and if enabled by the bits listed in the next register, they may cause processor interrupts. Both a set and clear operation are required to load arbitary data into this register. The bit assignments are identical to the Enable register below.

INTENA 09A W P Interrupt Enable bits (clear or set bits)
INTENAR 01C R P Interrupt Enable bits Read
This register contains interrupt enable bits. The bit assignment for both the request, and enable registers

is given below.

BIT#	FUNCT	LEVEL	DESCRIPTION
15	SET/CLR		Set/Clear control bit. Determines if bits written with a 1 get set or cleared. Bits written with a zero are always unchanged.
14	INTEN		Master interrupt (enable only , no request)
13	EXTER	6	External interrupt
12	DSKSYN	5	Disk Sync register(DSKSYNC or DSKSYNCX) matches Disk data
11	RBF	5	Serial port Receive Buffer Full
10	AUD3	4	Audio channel 3 block finished
09	AUD2	4	Audio channel 2 block finished
08	AUD1	4	Audio channel 1 block finished
07	AUD0	4	Audio channel 0 block finished
06	BLIT	3	Blitter finished
05	VERTB	3	Start of Vertical blank
04	COPER	3	Coprocessor
03	PORTS	2	I/O Ports and timers
02	SOFT	1	Reserved for software initiated interrupt.
01	DSKBLK	1	Disk Block finished
00	TBE	1	Serial port Transmit Buffer Empty

The extended interrupt request and enable registers have been added because of the hardware departure from fixed DMA time slots. Since DMA's are now processed on a priority basis rather than using the old guaranteed time slot scheme, it is possible that data overruns may occur in time based DMA channels of heavily loaded systems. This can be a particular problem when the user is attempting to invoke all DMA channels and at the same time enabling all bitplanes of a 1024 pixel or higher resolution display. Because the system has been designed to provide more bus bandwidth than the A500 to A3000 systems, old programs will run without problems. The problem will only become apparent in new high res systems, and especially those running with Fast Page mode RAMs rather than Video DRAMs. The extended interrupt request and enable registers are provided to inform the CPU when data overruns have occurred so that the CPU can take appropriate action to maintain data integrity.

INTREQX x 29C W P Interrupt Request bits (clear or set)
INTREQRX x 218 R P Interrupt request bits (read)

This register contains interrupt request bits(or flags). These bits may be polled by the processor, and if enabled by the bits listed in the next register, they may cause processor interrupts. Both a set and clear operation are required to load arbitary data into this register. The bit assignments are identical to the Enable register below.

INTENAX x 294 W P Interrupt Enable bits (clear or set bits)
INTENARX x 21C R P Interrupt Enable bits Read

This register contains interrupt enable bits. The bit assignment for both the request, and enable registers is given below. Note that bits 14-0 provide the same data as bits 14-0 of INTENA.

BIT#	FUNCT	LEVEL	DESCRIPTION
31	SET/CLR		Set/Clear control bit. Determines if bits written with a 1 get set or cleared. Bits written with a zero are always unchanged.
30	TBE2	1	Serial port 2 Transmit Buffer Empty.
29	AUD7OVR	4	Audio Channel 7 Data Overrun.
28	AUD6OVR	4	Audio Channel 6 Data Overrun.
27	AUD5OVR	4	Audio Channel 5 Data Overrun.

26	AUD4OVR	4	Audio Channel 4 Data Overrun.
25	AUD7	4	Audio channel 7 block finished
24	AUD6	4	Audio channel 6 block finished
23	AUD5	4	Audio channel 5 block finished
22	AUD4	4	Audio channel 4 block finished
21	VIDOVR	5	Graphics or Sprite Data Overrun.
20	DSKOVR	5	Disk Data Overrun. (Disk read or write mode can be determined by looking at DISKWRITE of DSKBYTR.) If a dmareq goes out and the previous response was not serviced in time, this happens. If the software wishes to stop the controller, it should set this bit. When all requests have cleared out the controller will give a DSKBLK intrpt and stop. The DSKBUSY bit can be checked for stoppage. If the dma bit is turned off it will also stop, but maybe not cleanly, with requests still in the pipeline, and it may not start up correctly immediately afterwards.
19	RBF2	5	Serial Port 2 Receive Buffer Full.
18	AUD3OVR	4	Audio Channel 3 Data Overrun.
17	AUD2OVR	4	Audio Channel 2 Data Overrun.
16	AUD1OVR	4	Audio Channel 1 Data Overrun.
15	AUD0OVR	4	Audio Channel 0 Data Overrun.

Overrun intreqs occur if data doesn't get there by the next request, in both dma and interrupt driven modes. Overrun intreqs also occur as a channel is transitioning to the idle state if no new data is available. These intreq's can be used in combination with the audstop bit (AUDxCTLX) to stop things cleanly.

14	INTEN		Master interrupt (enable only , no request)
13	EXTER	6	External interrupt
12	DSKSYN	5	Disk Sync register(DSKSYNC or DSKSYNCX) matches Disk data
11	RBF	5	Serial port Receive Buffer Full
10	AUD3	4	Audio channel 3 block finished
09	AUD2	4	Audio channel 2 block finished
08	AUD1	4	Audio channel 1 block finished
07	AUD0	4	Audio channel 0 block finished

intreqs come as the last 2 byte chunk is starting to be output as follows:
8 bit samples- when 2nd to last sample is processed;
16 bit samples- when last sample is processed.
The audio address pointer registers and the audio length registers can be changed at this time to affect the next dma fetch. If the audio hardware detects that the audstop bit is set at this time (when these intreq bits are set), audio dma will end at the end of this new buffer (giving an audovr intreq). If the audio period or volume registers are changed at this time (having received an audio intreq), the new values will take effect on the start of the next sample (8 bit samples- last sample before the next dma; 16 bit samples- first sample received on next dma)
if the dma enable bit for the channel in question is off, intreqs come whenever a new data chunk is needed:
in 32 bit mode, <= 1 sample time latency is required with 16 bit data,
<= 2 sample time latency is required with 8 bit data.

06	BLIT	3	Blitter finished
05	VERTB	3	Start of Vertical blank
04	COPER	3	Coprocessor
03	PORTS	2	I/O Ports and timers
02	SOFT	1	Reserved for software initiated interrupt.
01	DSKBLK	1	Disk block interrupt. Occurs at the end of a disk transfer or when an error has caused disk controller stop. Also occurs in CD mode (==2) every sector when it is time to reload the pointer register (for doing double buffered output)
00	TBE	1	Serial port Transmit Buffer Empty

JOY0DAT 00A R P Joystick-mouse 0 data (left vert,horiz)
JOY1DAT 00C R P Joystick-mouse 1 data (right vert,horiz)

These addresses each read a pair of 8 bit mouse counters. 0=left controller pair, 1=right controller pair. (4 counters total). The bit usage for both left and right addresses is shown below. Each counter is clocked by signals from 2 controller pins. Bits 1 and 0 of each counter may be read to determine the state of these 2 clock pins.This allows these pins to double as joystick switch inputs.

Mouse counter usage

BIT#	15,14,13,12,11,10,09,08	07,06,05,04,03,02,01,00
0DAT	Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0	X7 X6 X5 X4 X3 X2 X1 X0
1DAT	Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0	X7 X6 X5 X4 X3 X2 X1 X0

Mouse Port

Pin	usage
1	vertical
3	vertical quadrature
2	horizontal
4	horizontal quadrature

Joystick port switches

To detect these Directions	Pin#	read these counter bits from joystick and XOR them
Forward	1	Y1 xor Y0 (BIT#09 xor BIT#08)
Left	3	Y1
Back	2	X1 xor X0 (BIT#01 xor BIT#00)
Right	4	X1

CONN PIN	JOYSTICK FUNCTION	MOUSE FUNCTION	MOUSE USAGE NAME	PINNAME
L1	FORW*	Y	M0V	LFORWARD*
L3	LEFT*	YQ	M0VQ	LLEFT*
L2	BACK*	X	M0H	LBACK*
L4	RIGH*	XQ	M0HQ	LRIGHT*
R1	FORW*	Y	M1V	RFORWARD*
R3	LEFT*	YQ	M1VQ	RLEFT*
R2	BACK*	X	M1H	RBACK*
R4	RIGH*	XQ	M1HQ	RRIGHT*

JOYTEST 036 W P Write to all 4 Joystick-mouse counters at once.

Mouse counter write test data

BIT#	15,14,13,12,11,10,09,08	07,06,05,04,03,02,01,00

```
                0DAT  Y7 Y6 Y5 Y4 Y3 Y2 xx xx  X7 X6 X5 X4 X3 X2 xx xx
                1DAT  Y7 Y6 Y5 Y4 Y3 Y2 xx xx  X7 X6 X5 X4 X3 X2 xx xx
```

MLSYNC 5A0 W A — Mid line sync. Determines the horizontal position of the VSYNC transition for alternate fields when in composite NTSC/PAL compatibility modes. VSYNC start will alternate between this position and HRESET(count 0xa). (See HTOTALX for bits.)

MONICAID x 07C R M — Video out chip revision level (was DeniseID) The original Denise (8362) does not have this register, so whatever value is left over on the bus from the last cycle will be there. DeniseHR (8373) will return FC in the lower 8 bits. MONICA will return F0 in the lower 8 bits if she's in a single graphics chip system, and F1 if she's in a dual system. The upper 8 bits of this register are reserved.

MONITORID 488 R External — Monitor ID. When this RGA address is present on the bus, external hardware will drive a longword onto the bus indicating the type of monitor currently plugged into the system. To avoid the need for additional external hardware, MARY has a decoder for this address and provides an enable signal, MONIDEN*.

MONTEST 49C W M — MONICA Test Register.
bit 2- DIGON Digital bus on. Enables the RGB digital output bus in the single system. This bit is ignored in dual systems.
All other bits in this register are undefined, and should be driven low.

POT0DAT h 012 R P — Pot counter data left pair (vert,horiz)
POT1DAT h 014 R P — Pot counter data right pair (vert,horiz)
These addresses each read a pair of 8 bit pot counters. (4 counters total). The bit assignment for both addresses is shown below. The counters are stopped by signals from 2 controller connectors with 2 pins each(left=0, right=1).

```
BIT#  15,14,13,12,11,10,09,08  07,06,05,04,03,02,01,00
----- ----------------------- ----------------------
RIGHT Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0  X7 X6 X5 X4 X3 X2 X1 X0
LEFT  Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0  X7 X6 X5 X4 X3 X2 X1 X0
```

CONNECTORS

loc.	dir.	sym	pin
RIGHT	Y	RY	9
RIGHT	X	RX	5
LEFT	Y	LY	9
LEFT	X	LX	5

The pots will give a full scale (FF) reading with about 500kohms in one 60hz frame time.

POTGO 034 W A P — Pot Port direction and data, pot counter start, and test bits.

Write only; ANDREA , MARY
This register controls a 4 bit bi-directional I/O port that shares the same 4 pins as the 4 pot counters above.
Audio sampling control bits.

BIT#	FUNCT	DESCRIPTION

----	-----	----------
15	OUTRY	Output enable for pin POTRY
14	DATRY	I/O data for pin POTRY
13	OUTRX	Output enable for pin POTRX
12	DATRX	I/O data for pin POTRX
11	OUTLY	Output enable for pin POTLY
10	DATLY	I/O data for pin POTLY
9	OUTLX	Output enable for pin POTLX
8	DATLX	I/O data for pin POTLX
7	TSTCNT	tick the pot counters + divider 1 notch
6	TSTCNTEN	use the tstcnt above. Otherwise it counts all the time
5	FC2	force a carry into bit 6 of POT divider (for chip test)
4	FC1	force a carry into the beginning of the 8 bit POT cntr.
3	INT1_EXT0	select POTMOUSE or external sampling.
2	AUD_SAMP	set audio sampling mode.
1	x	must be set to 0
0	START	start pot counters (normal and audio sampling)

Always write x bit to 0.
Only bit 3 is decoded in ANDREA.
To enable POTMOUSE sampling, the following bits must be set:

FC1	1	Force pot counter to count on BUSCLKs.
AUD_SAMP	1	enable sampling
INT1_EXT0	1	select internal (POTMOUSE) sampling
START	1	start the pot counters

To enable external sampling, the following bits must be set:

AUD_SAMP	1	enable sampling
INT1_EXT0	0	select external sampling

POTLY and POTLX may be used for I/O during internal and external audio sampling mode
POTRX and POTRY may be used for I/O during external sampling.
The right mouse port may not be used during internal or external audio sampling modes

POTINP 016 R P Pot pin data read

This register controls a 4 bit bi-directional I/O port that shares the same 4 pins as the 4 pot counters above. It also permits software to distinguish MARY from PAULA by polling the MARYID bits (PAULA=0, MARY=1), and determine if audio sampling mode is enabled.

BIT#	FUNCT	DESCRIPTION
15	OUTRY	Output enable for Paula pin POT1Y
14	DATRY	I/O data Paula pin POT1Y
13	OUTRX	Output enable for Paula pin POT1X
12	DATRX	I/O data Paula pin POT1X
11	OUTLY	Output enable for Paula pin POT0Y
10	DATLY	I/O data Paula pin POT0Y
09	OUTLX	Output enable for Paula pin POT0X
08	DATLX	I/O data Paula pin POT0X
07-04	MARYID	MARY Identification
03	INT1_EXT0	select POTMOUSE or external sampling.
02	AUD_SAMP	set audio sampling mode.
01-00	x	

PRISET n 430 W A Write Processor and Blitter priority control bits.

The PROPRI and BLTPRI bits of DMACONX when PRISET is written. The value on D10 is written into BLTPRI and the value on D12 is written into PROPRI. Access to this register has priority over the blitter regardless of current BLTPRI settings. Note that this register is intended for sparing use by interrupt routines. Abuse of this register will void the purpose of BLTPRI.

PRITEST n 408 R A Read Processor and Blitter priority control bits. See PRISET for data format. Access to this register has priority over the blitter regardless of current BLTPRI settings.

RAMATRN n 400 R A RAM Attribute registers, read.

RAMATWN n 47C W A RAM Attribute registers, write.
These registers are used to store information about the physical RAM in the RGA address space. ANDREA uses the first memory cycle following a system reset to read configuration data about chip DRAM. RAMATN is written with the config data obtained from auto config logic present on each RAM card. ANDREA uses this information to determine how to access RAM when performing bitplane DMA fetches. Other RAMATN data indicates to software whether an external blitter is present on the RAM card, and whether the card expects 9 or 10 address (RAMADDR) lines.
When ANDREA asserts the CONFIG_RD output, the auto config logic built into each RAM card drives it's configuration data onto the AD bus. The mother PC board is fabricated such that each card will drive onto a different portion of the AD bus. (The PC board has been designed such that the 0000 state is driven when no RAM board has been plugged into a slot.)
Note that the address space of each high end system RAM board is twice as big as that in a low end system. Software should take this into consideration when programming multiple BLIT chips.
It is not possible to mix 9 address line/10 address line boards in a system. Board 0 must be populated, and nybble bit 2 of this board is used by ANDREA to determine which RAMADDR lines to drive.

Bus Position	Config data received
31-28	RAM board 7
27-24	RAM board 6
23-20	RAM board 5
19-16	RAM board 4
15-12	RAM board 3
11-8	RAM board 2
7-4	RAM board 1
3-0	RAM board 0

Bit Position in nybble	Meaning	
3	Slot filled.	0- no board present, 1- slot occupied.
2	RAM size.	0- 9 address line RAM chips, 1- 10 address line RAM chips.
1	BLIT chip.	0- no external BLIT chip 1- BLIT chip present.
0	RAM type.	0- Fast Page DRAM,

1- Fast Page VRAM.

REFPTR 028 W A Refresh pointer
This register is used as a Dynamic RAM refresh address generator. It is writeable for test purposes only, and should never be written by the microprocesor.

SER1STOP 578 W A First Serration pulse stop position.
This register contains the horizontal beam position required to deassert the first serration pulse (SER1). The SER1 pulse is always set at the HSSTRT beam position.
See HTOTALX for bit positions.
Required value for both NTSC and PAL: SER1STOP = 1D2 (hex)

SER2STOP 57C W A Second Serration pulse stop position.
This register contains the horizontal beam position required to deassert the second serration pulse (SER2). The SER2 pulse is always set at the HCENTER beam position.
See HTOTALX for bit positions.
Required value for both NTSC and PAL: SER2STOP = 00C (hex)

SERDAT 030 W P Serial Port Data and stop bits write
This address writes data to a Transmit data buffer. Data from this buffer is moved into a serial shift register for output transmission, whenever it is empty. This sets the Interrupt Request TBE (transmit buffer empty). A stop bit must be provided as part of the data word. The length of the data word is set by the position of the stop bit.

BIT#	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
USE	0	0	0	0	0	0	S	D8	D7	D6	D5	D4	D3	D2	D1	D0

note: S= stop bit =1 , D= data bits

SERDAT2N 484 W P Serial Port 2 Data and stop bits write
This register functions identically to SERDAT above, except it controls UART 2.

BIT#	31	30	-10	09	08	07	06	05	04	03	02	01	00
USE		0		S	D8	D7	D6	D5	D4	D3	D2	D1	D0

note: S= stop bit =1 , D= data bits

3)- UART baud counter test bit. Looks counter carry chain after the 6th counter bit and forces a carry into the 7th counter bit

SERDATR 018 R P Serial Port Data and Status read
This address reads data from a Receive data buffer. Data in this buffer is loaded from a receiving shift register whenever it is full. Several interrupt request bits are also read at this address, along with the data, as shown below.

BIT#		
15	OVRUN	Serial port receiver overun
14	RBF	Serial port Receive Buffer Full (mirror)
13	TBE	Serial port Transmit Buffer Empty (mirror)
12	TSRE	Serial port Transmit shift reg. empty
11	RXD	RXD pin receives UART serial data for direct bit test by the micro
10	RMORE	Read More. Indicates there is more data to be read from the receive fifo.
09	STP	Stop bit
08	STP-DB8	Data bit if LONG, Stop bit if not.
07	DB7	Data bit
06	DB6	Data bit
05	DB5	Data bit
04	DB4	Data bit
03	DB3	Data bit
02	DB2	Data bit
01	DB1	Data bit

00	DB0	Data bit

SERDATR2N 41C R P Serial Port 2 Data and Status read

This register functions identically to SERDATR above, except it is used to read UART 2.
This address reads data from a Receive data buffer. Data in this buffer is loaded from a receiving shift register whenever it is full. Several interrupt request bits are also read at this address, along with the data,as shown below.

BIT#		
31	OVRUN	Serial port 2 receiver overun
30	RBF2	Serial port 2 Receive Buffer Full(mirror)
29	TBE2	Serial port 2 Transmit Buffer Empty(mirror)
28	TSRE2	Serial port 2 Transmit shift reg. empty
27	RXD2	RXD2 pin receives UART serial data for direct bit test by the micro
26	RMORE2	Read More from port 2. Indicates there is more data to be read from the receive fifo.
25	STP	Stop bit
24	STP-DB8	Data bit if LONG, Stop bit if not.
23	DB7	Data bit
22	DB6	Data bit
21	DB5	Data bit
20	DB4	Data bit
19	DB3	Data bit
18	DB2	Data bit
17	DB1	Data bit
16	DB0	Data bit

SERPER 032 W P Serial Port Period and control

This register contains the control bit LONG refered to above, and a 15 bit number defining the serial port Baud Rate. If this number is N, then the Baud Rate is 1 bit every (N+1)*.2794 Microseconds.

BIT#		
15	LONG	Defines Serial Receive as 9 bit word.
14-00	RATE	Defines Baud Rate=1/((N+1)*.2794 microsec)

SERPER2 48C W P Serial Port 2 Period and control

This register contains the control bit LONG referred to above, and a 15 bit number defining the serial port Baud Rate. If this number is N, then the Baud Rate is 1 bit every (N+1)*.2794 Microseconds.

BIT#		
15	LONG	Defines Serial port 2 Receive as 9 bit word.
14-00	RATE	Defines Baud Rate=1/((N+1)*.2794 microsec)

SPRxPOS 140 W A M Sprite x Vert-Horiz start position data
SPRxCTL h 142 W A M Sprite x position and control data

These 2 registers work together as position, size and feature Sprite control registers. They are usually loaded by the Sprite DMA channel, during horizontal blank, however they may be loaded by either processor any time. When these registers are written, bits SV10, SH10, and EV10 of the SPRxPOSX and SPRxCTLX registers are cleared.
Note- These registers are copied to working registers at each horizontal count 10 decimal. Therefore, data written into these registers will not effect the display until the next horizontal count 10 event. This means it is not possible to reuse a Sprite on the same display line. The effect of this is to delay Sprite output one display line so as to match the line delay associated with the line data buffer chip.

SPRxPOS register:

BIT#	SYM	FUNCTION
15-08	SV7-SV0	Start vertical value. High bit(SV8) is in SPRxCTL reg below.
07-00	SH8-SH1	Start horizontal value. Low bit(SH0) is in SPRxCTL reg. below.

SPRxCTL register: (writing this address disables sprite horizontal comparator circuit)

BIT#	SYM	FUNCTION
15-08	EV7-EV0	End (stop) vert. value. low 8 bits
07	ATT	Sprite attach control bit(odd sprites)
06	SV9	start vert. value 10th bit
05	EV9	end (stop) vert. value 10th bit
04	SHSH	start horiz. Allows sprite start on highres pixel boundaries. Compares to H00 of new horizontal counter.
03	FAST	When set causes Sprite to be output at the highres pixel rate.
02	SV8	Start vert. value 9th bit
01	EV8	End (stop) vert. value 9th bit
00	SH0	Start horiz. value Low bit -- (1 lowres pixel increment) Compares to H01 of new horizontal counter.

SPRxPOSX x 380 W A Sprite x Vert start and stop position data
SPRxCTLX x 384 W M Sprite x horizontal position and control data

These 2 registers work together as position, size and feature Sprite control registers. Most of the bits in these registers are identical to the bits in SPRxPOS and SPRxCTL. These registers are provided to permit SPRITE start and stop at arbitrary locations on high resolution monitors. They are usually loaded by the Sprite DMA channel during the preceeding display line, however they may be loaded by either processor any time.
(Note:
1) There is a one line pipeline delay in the SPRITE data registers. Data written directly to the SPRITE register will actually appear one line later, and if the SPRITE DMA channel is enabled, this data will be overwritten by the DMA channel write. Because of this, SPRITEs cannot be reused on the same display line.)
2) The last entry of the SPRITE list must be entered twice when burst64 is enabled. The last entry of the SPRITE list must be entered four times when burst128 is enabled. This assures the SPRITE list will not become misaligned. (SPRITE lists must be longword aligned in 32 bit mode, double longword aligned in 64 bit mode, and quadruple longword aligned in 128 bit mode.)

SPRxPOSX register:

BIT#	SYM	FUNCTION
28	BURST128	Use a Burst mode fetch to satisfy 128 bit SPRITE access.
27	BURST64	Use a Burst mode fetch to satisfy 64 bit SPRITE access.
26-16	SV10-SV0	Start vertical value.
10-00	EV10-EV0	Stop vertical value.

SPRxCTLX register: (writing this address disables sprite

horizontal comparator circuit)

BIT#	SYM	FUNCTION
25	CONSPRT (also known as BDRSPRT)	Constant sprite. When set, the sprite is displayed everywhere including blanking intervals except line 0(Note, this demands that Sprite DMA channel reads of position and control registers occur during line 0 as opposed to the normal VBSTOP-1 line.) This allows the display of sprite data in the screen margins. margins.
24	SMIRROR	When set, this bit indicates that the sprite is to be mirrored each time it is repeated. This allows easy generation of double-sized sprites when they are symetrical about a vertical axis. This field only has effect when HSRPT > 0. The mode is realized in hardware by reversing sprite register shift direction each time the sprite is repeated.
23	ATT	SPRITE attach control bit(odd SPRITEs)
22-17	HSRPT	Horizontal sprite repeat. The number put in this field defines how many (continuous) times plus one a sprite is repeated each horizontal line. HSRPT=0 means display the sprite once each line.
16	SRES	SPRITE resolution. Reset on power up.
10-00	HS10-HS0	Start horizontal value. (Note: HS0 is SHSH of SPRxCTL, HS1 is SH0 of SPRxCTL HS0 compares to H00 of hori. counter, HS1 compares to H01 of hori. counter, HS2 compares to H1 of hori. counter, HS3 compares to H2 of hori. counter, etc)

SPRxDATA	144	W	M	Sprite x image data register A
SPRxDATB	146	W	M	Sprite x image data register B

These registers buffer the Sprite image data. They are usually loaded by the Sprite DMA channel but may be loaded by either processor at any time. When a horizontal comparison occurs the buffers are dumped into shift registers and serially output to the display, MSB first on the left.
Note: Writing to the A buffer enables (arms) the sprite. Writing to the SPRxCTL register disables the sprite.
If enabled, data in the A and B buffers will be output whenever the beam counter equals the sprite horizontal position value in the SPRxPOS register. When SRES of SPRxCTLX is 0 and lowres mode is enabled,
1 sprite pixel is 1 bitplane pixel wide, and in highres mode, one sprite pixel is two bitplane pixels. When SRES of SPRxCTLX is 1, in lowres mode 1 sprite pixel is one bitplane pixel, and in highres mode 1 sprite pixel is also one bitplane pixel.
DMA transfers to this register are done when SPREN of DMACON is set. When these registers are written, the least significant word of the corresponding extended SPRITE data register is cleared.
Note- These registers are copied to working registers at each horizontal count 10 decimal. Therefore, data written into these registers will not effect the display until the next horizontal count 10 event. This means it is not possible to reuse a Sprite on the same display line.

SPRxDATAX x 340 W M Extended Sprite x image data register A
SPRxDATBX x 344 W M Extended Sprite x image data register B
These extended registers are the same as the SPRxDAT registers except provide the capacity for 32 bit SPRITEs. DMA transfers to this register are done when SPRENX of DMACONX is set.

SPRxPTH 120 W A Sprite x pointer (High 5 bits)
SPRxPTL 122 W A Sprite x pointer (Low 16 bits, bit 0 forced low)
This pair of registers contains the 20 bit word address of Sprite x (x=0,1,2,3,4,5,6,7) DMA data. These address registers must be initialized by the processor or coprocessor every vertical blank time. If SPREN of DMACON is enabled, 16 bit word transfers from the address pointed to by this register will be done with each sprite DMA transfer. If SPRENX of DMACONX is enabled, 32 bit longword transfers from the address pointed to by SPRnPTX(A1 and A0 forced low) will be done.
Note- These registers are copied to working registers at each horizontal count 10 decimal. Therefore, data written into these registers will not effect the display until the next horizontal count 10 event.

SPRnPTX 320, W A Sprite n pointer (24 bits, A1, A0 forced low)
324, (n=0-7) These registere are the same as SPRxPTH&L above except
328, they extend sprite addressing capabilities to 16Mbytes.
32C, Whenever SPRxPTL or SPRxPTH are written, bits 23-21 of SPRnPTX
330, are cleared.
334,
338,
33C

VBSTOP 1CE W A Vertical line for VBLANK stop
VBSTRT 1CC W A Vertical line for VBLANK strt
The logic level generated by these registers is logically ORed with the level generated by the HBSTRT and HBSTOP registers and output to the CBLK* pin.
See VTOTAL for bit positions.
Warning- system software calls should be made to make changes to these registers.

VEQUSTOP 580 W A Vertical Equalization Stop Position.
This register contains the vertical beam position, or raster line, at which vertical equalization is terminated. Vertical equalization is assumed to start at the beginning of vertical blanking, or VBSTRT.
Required value for NTSC: VEQUSTOP = 00090000 (line 9).
for PAL: VEQUSTOP = 80070000 (line 7 and 1/2).

VPOSR 004 R A Read Vert most sig. bits (and frame flop)
VPOSW 02A W A Write Vert most sig. bits (and frame flop)
BIT# 15,14,13,12,11,10,09,08,07, 06,05,04,03,02,01,00
USE LOF I6 I5 I4 I3 I2 I1 I0 LOL -- -- -- --v10 v9 V8
LOL=Long line bit.
LOF=Long frame (auto toggle control bit in BPLCON0, LACE)
I0-I6 chip identification:
8361(regular) or 8370(fat) (agnus-ntsc) = 10
8367(pal) or 8371(fat-pal) (agnus-pal) = 00
8368(hr) or 8372(fat-hr) (agnushr) = 20 PAL, 30 NTSC
ANDREA, single = 40
ANDREA, dual = 60
v9,v10-- chips with identifier 20 or 30 only

Note- Hardware generated LOL no longer exists the way it had in previous revisions of the chip set (LOF functions as before). This is because NTSC line lengths can be generated directly using the extended HTOTALX register. The LOL bit shown above is now a simple toggle flip flop which changes state every line. This toggle flip flop preserves a level of software compatability.
LOF is a writable bit while LOL is cleared when written. LOL is always clear in PAL mode.
In non-interlaced display modes, the LOF bit should be set to allow sprites a full horizontal line time to be serviced during the vertical blanking region (see section 5.5.3, Software Control of Graphics).
Warning- system software calls should be made to make changes to these registers.

VHPOSR 006 R A Read Vert and Horiz Position of beam, or lightpen
VHPOSW 02C W A Write Vert and Horiz Position of beam, or lightpen
BIT# 15,14,13,12,11,10,09,08,07,06,05,04,03,02,01,00
USE V7 V6 V5 V4 V3 V2 V1 V0,H8 H7 H6 H5 H4 H3 H2 H1
RESOLUTION = two low res pixels.
Warning- system software calls should be made to make changes to these registers.

VHPOSRX 204 R A Read Vert and Horiz Position of beam, or lightpen
BIT# 31-27 26-16 15-11 10-0
USE x v10-v0 x h9-h1,h01,h00
(x bits should be written with 0 to retain upwards compatibility)
Warning- system software calls should be made to make changes to these registers.

VHPOSWX 22C W A Write Vert and Horiz Position of beam, or lightpen
BIT# 31 30-27 26-16 15-11 10-0
USE vhalf x v10-v0 x h9-h1,h01,h00
(x bits should be written with 0 to retain upwards compatibility)
Warning- system software calls should be made to make changes to these registers.

VPPOSRN 428 R A Read Primary vertical position of beam.
This register permits reads of the primary vertical counter while the other registers(ie VHPOSRX) permit reads of the secondary vertical counter. Writes to the secondary counter via VHPOSWX or VHPOSW also updates the primary counter. When this is done, vhalf is cleared.
BIT# 31 30-27 26-16 15-11 10-0
USE vhalf x v10-v0 x h9-h1,h01,h00
(x bits should be written with 0 to retain upwards compatibility)

VSSTOP 1CA W A Vert. position for VSYNC stop.
VSSTRT 1E0 W A Vert. position for VSYNC start
See VTOTAL for bit positions.
Warning- system software calls should be made to make changes to these registers.

VTOTAL x 1C8 W A Highest numbered vertical line
It's the line number to reset the counter, so there's this many + 1 lines in a field. The exception is if the LACE bit is set (BPLCON0), in which case the vhalf counter bit is enabled and the vertical counter counts half

lines instead of full lines. When this is the case, both HTOTAL as well as HCENTER are used to increment the vertical counter, and there are this many + 1 half lines in a field. (Note: vhalf must be set to a 0 to produce a spec interlaced NTSC or PAL display, 1 to produce a non-interlaced display.)

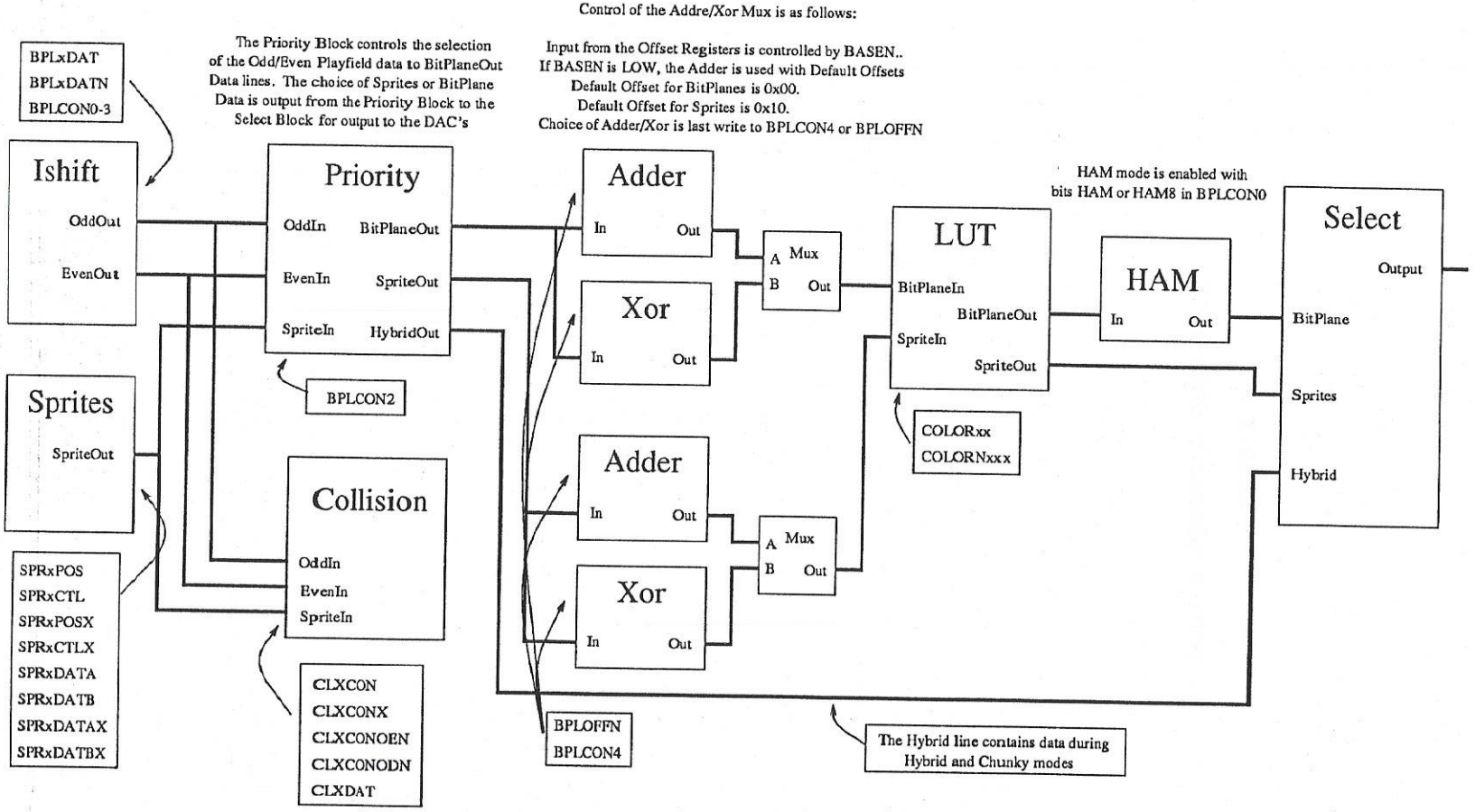
Bit#	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
	vhalf	x	x	x	x	v10	v9	v8	v7	v6	v5	v4	v3	v2	v1	v0

Warning- system software calls should be made to make changes to this register.

APPENDIX G

MONICA GRAPHICS DATA PATH.

Monica Graphics Data Paths



APPENDIX H

APPLICATION NOTE: A LOW COST AAA SYSTEM.

The accompanying schematic shows the AAA chips in a full system environment. (Disclaimer: all possible effort was made to check and double check the schematic for correct and accurate electrical hookup, but even as such it is quite likely the schematic contains hookup errors. The schematic has been provided as an educational aide only to familiarize the reader with the AAA chips, and show how they they can be used to build an Amiga system.) This system design is based on the proposed A600 system and peripherals. Every attempt was made to produce the cheapest AAA system possible. All system glue logic (including the functions provided by the 8520 port chips of existing systems) is shown absorbed into two 160 pin gate array chips. Other cost tradeoffs were also made to keep system costs down. As such, some of the features provided by the AAA chips cannot be used in this system. For example, to save the cost of four eight-bit wide unidirectional tristate buffers and an extra wide memory expansion connector, the system shown does not support the use of VRAM. Due to the lack of VRAM in the system, no provisions have been made to support the AAA framegrabber mode (VRAM is required to use framegrabber mode). Another major tradeoff made is the choice of pixel clock frequencies. This has been limited in the system to the same set of frequencies needed to drive the proposed AA+ display system. This implementation can only drive the same set of monitors which will be driven with the proposed AA+ chips.

The AAA system presented provides all of the enhancements currently being specified for the AA+ system, plus:

800x600, 72Hz, 256 colors out of 16 Meg colors
640x400, 60Hz, 32k colors out of 32k colors
, 256 colors out of 16 Meg colors
640x200, 60Hz, 'True Color', 16 Meg colors out of 16 Meg
, 32k colors out of 32k colors
, 256 colors out of 16 Meg colors

provides special AAA video pixel modes
24-bit true color
packed pixel modes
others

provides 4x floppy density support (need high density drive)
hardware IBM sector mode support

CD quality audio
16 bit samples
sample rates to 110Khz

Advanced blitter
3 6x typical performance increase (as much as 11x during high resolution screen displays)
simplified blitter programming
clip-rect line draw
chunky pixel arithmetic

copper
can be used to restart blitter
has multiple move instruction

uart has 4 byte FIFO

Wide Sprites
16, 32, 64, and 128 bit Sprites (8 of them).
Sprite mirror and repeat modes.

system comes with 1 megabyte of DRAM

The table on the following page compares projected system costs for the proposed AA+ system and the AAA system shown. Several notes are in order with respect to these projected system costs:

1) No attempt was made to include assembly costs or yield.
2) Both of the new systems will witness a substantial reduction in component insertion count. The AA+ system will have the smallest insertion count.
3) The AA+ system will contain two 160 pin PQFP's. The 'minimum' AAA system will contain two 160 pin PQFP's and four 144 pin PQFP's.

The AA+ system provides the cheapest solution. However, the analysis shows that the AAA system is not outside of Commodore's historic low end system cost range. With a three times markup, the system would retail for around 650 dollars. A four times markup would yield a 850 dollar retail price.

Projected Low-End Systems Cost Comparisons

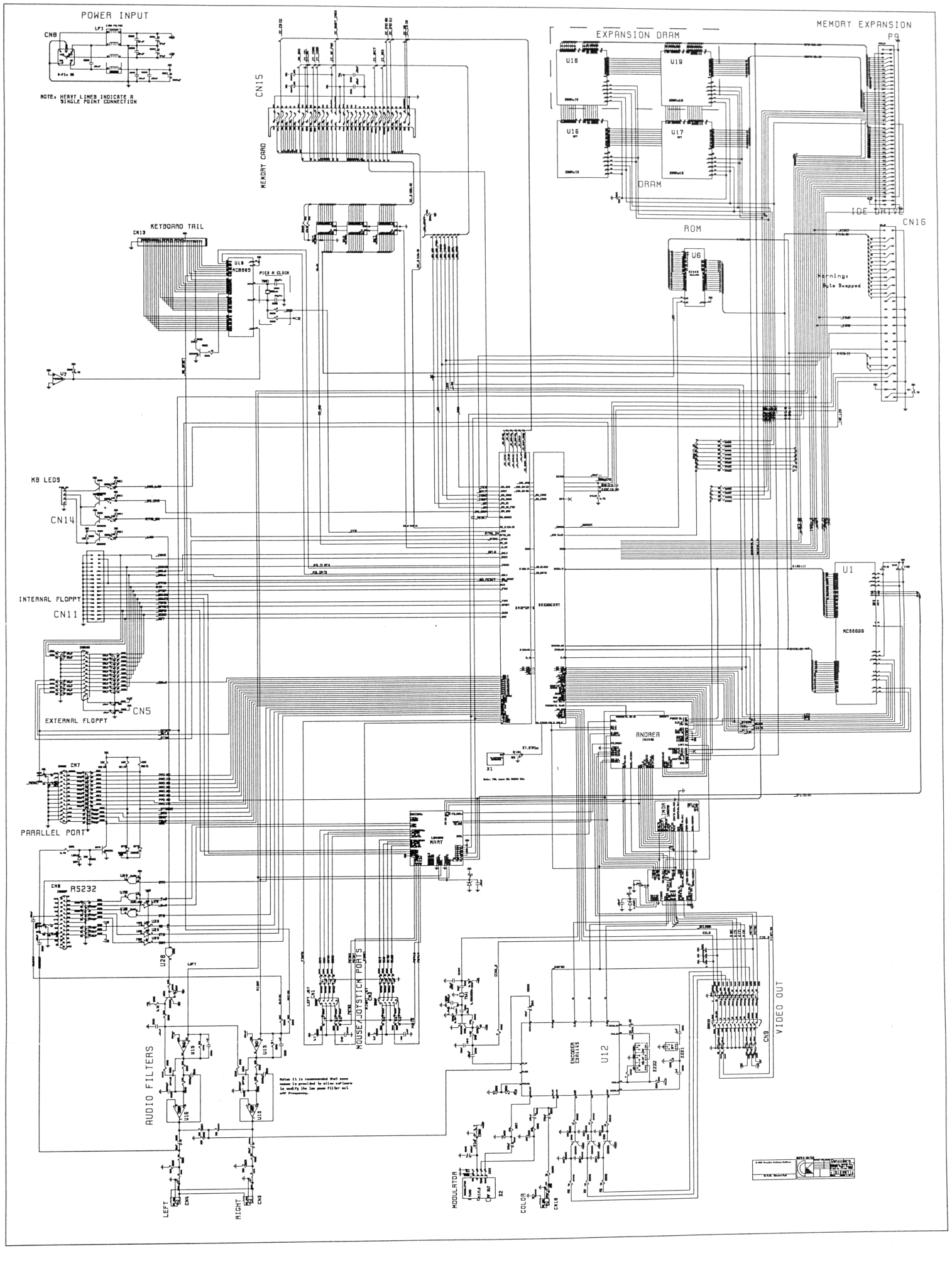
AA+		AAA	
1 MC1488D	$ 0.14	1 MC1488D	$ 0.14
1 MC1489D	$ 0.14	1 MC1489D	$ 0.14
1 LF347	$ 0.42	1 LF347	$ 0.42
1 PST518B	$ 0.10?	1 PST518B	$ 0.10?
5 74LS245	$ 1.05	3 74LS245	$ 0.63
1 74HCT168	$ 0.25?		
1 DRAM, 256Kx16	$14.165	2 DRAM, 256Kx16	$28.33
1 ROM, 256Kx16	$ 3.785	1 ROM, 256Kx16	$ 3.785
1 CXA1145	$ 1.19	1 CXA1145	$ 1.19
1 MC68000FN8	$ 1.983	1 MC68000FN8	$ 1.983
1 MC6805KYBD	$ 2.00?	1 MC6805KYBD	$ 2.00?
1 DEBI(160)	$ 5.60	1 AAAPORT(160)	$ 5.60
1 ARIEL(160)	$ 10.48	1 ANDREA(144)	$ 18.70
1 BELLE(84)	$ 8.97	1 MARY(144)	$ 10.36
		1 MONICA(144)	$ 16.38
		1 AAAGARY(160)	$ 5.60
		1 LINDA(144)	$11.40
Subtotal	-------		-------
18	$50.273	19	$106.758
Other	$107.567	Other	$107.567
	-------		-------
Total	$157.84	Total	$214.325
Retail	$480	Retail	$650

If both systems came with 1Megabytes of RAM:

Total	$172.01	Total	$214.325
Retail	$520	Retail	$650

Compare to A500 based on similar markups...

A500 costs...	~$178
A500 retail...	$540


POWER INPUT
CN8
NOTE: HEAVY LINES INDICATE A SINGLE POINT CONNECTION
MEMORY EXPANSION
EXPANSION DRAM
P9
U18
U19
U16
U17
DRAM
CN15
MEMORY CARD
IDE DRIVE
CN16
KEYBOARD TAIL
CN13
ROM
U6
Warning: Byte Swapped
PICO A CLOCK
KB LEDS
CN14
INTERNAL FLOPPY
CN11
U1
MC68000
EXTERNAL FLOPPY
CN5
ANDREA
X1
CN7
PARALLEL PORT
CN8
RS232
U28
MOUSE/JOYSTICK PORTS
AUDIO FILTERS
U15
U13
LEFT
CN4
RIGHT
CN3
ENCODER
U12
CN9
VIDEO OUT
MODULATOR
X2
COLOR
CN10

INDEX