1.0 Hombre CPU Chip (Nathaniel)

The CPU chip acts as the primary control for Hombre based systems. Figure XXX shows a block diagram of the chip. The chip contains an embedded PA-RISC processor, DMA engine, blitter, memory controls, and bus interfaces. In small systems, the internal processor executes out of display VRAM. Larger systems may add system DRAM to increase available bandwidth for certain DMA operations. An external PA-RISC processor may be added to increase performance. It is intended that when this processor is present that it will execute primarily out of system DRAM while the internal processor executes primarily out of display VRAM. Appendix D contains a detailed description of the behavioral schematics for the CPU chip.

1.1 CPU Pins

Clock signals

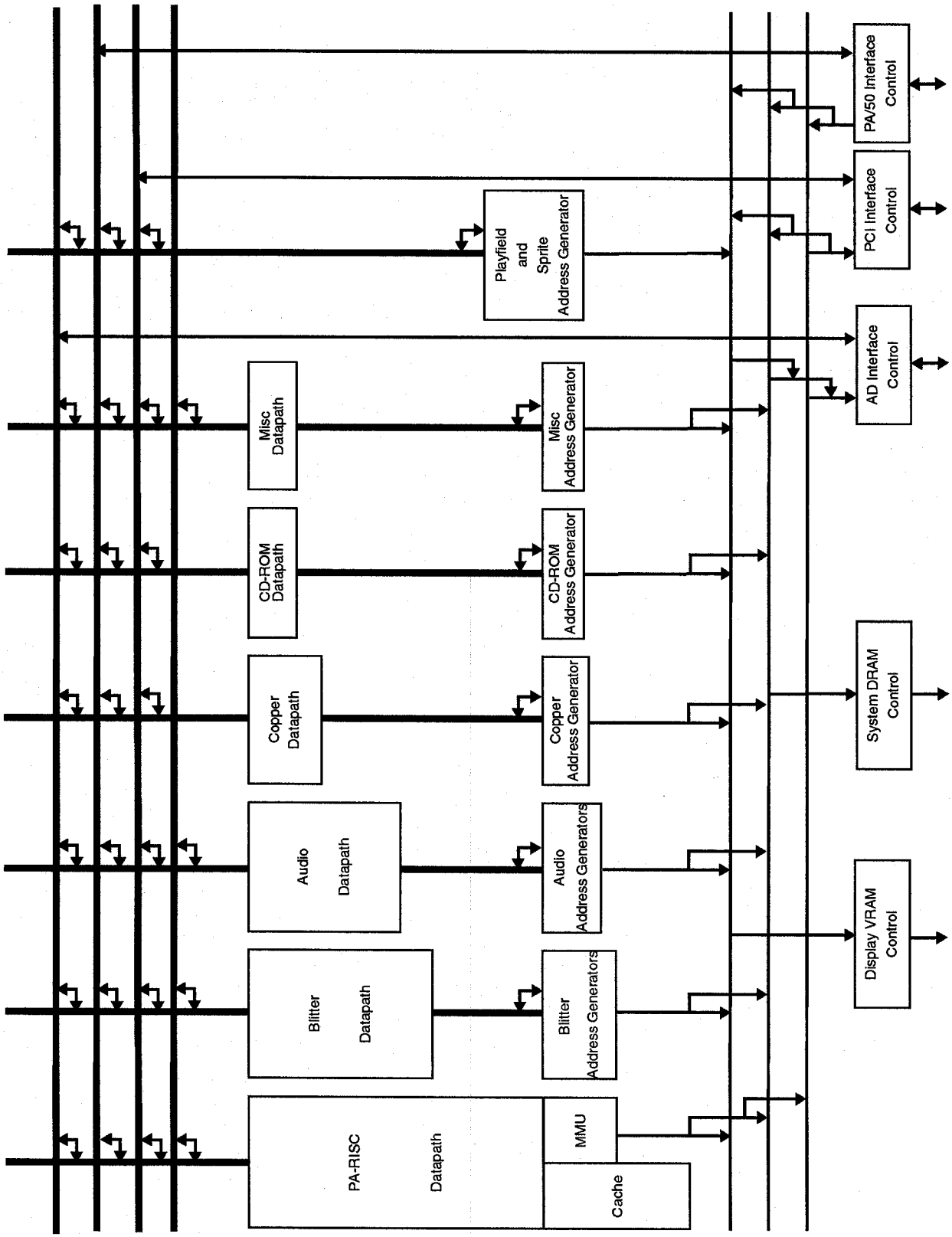
Name	*#*	*Type*	*Description*
sys_clk	1	Input	50 Mhz system clock
sys_reset0	1	Input	System reset

System DRAM Interface

Name	*#*	*Type*	*Description*
s_ma1[10:0]	11	Output	Multiplexed ROW/COL address for System DRAM. The number of signals actually used is a function of a configuration option. This option declares the organization of the DRAMs being used in the system.
s_ras0[7:0]	8	Output	Per Bank Row Address Strobe. Each RAS signal is connected to its own bank of DRAM. The chip therefore supports eight banks of DRAM. The bits of the address used to decode these signals is a function of a configuration option.
s_cas0[3:0]	4	Output	Per byte Column Address Strobe. Each byte lane is controlled by its own CAS signal. s_cas0[3] is to be connected to the most significant byte of the 32 bit wide memory.
s_r1w0	1	Output	Write Enable. This signal, when low indicates that a write operation is to occur.
s_buf_in1out0	1	Output	Buffer direction. This signal controls the direction of the bi-directional buffer which isolates the AD bus from the PA/50 bus. It is high when information is to flow from the AD bus to the PA/50 bus and is low when information is to flow from the PA/50 bus to the AD bus.
s_buf_en0[1:0]	2	Output	Buffer enables. These signals enable the bi-directional buffers which isolate the AD bus fromthe PA/50 bus. For systems which have 32 bit graphics memory, only the s_buf_en0[0] signal is used. When there is a 64 bit graphics memory, these signals not only isolate the two busses, but also provide a 64-bit to 32-bit multiplexor function.

Display VRAM Interface

Name	*#*	*Type*	*Description*



d_ma1[10:0]	11	Output	Multiplexed ROW/COL address for Display VRAM. The number of signals actually used is a function of a configuration option. This option declares the organization of the VRAMs being used in the system.
d_ras0[3:0]	4	Output	Per Bank Row Address Strobe. Each RAS signal is connected to its own bank of VRAM. The chip therefore supports four banks of VRAM. The bits of the address used to decode these signals is a function of a configuration option.
d_cas0	1	Output	Column Address Strobe. All byte lanes are controlled by a single CAS signal. Writes to individual bytes is performed by Read-Modify-Write cycles.
d_r1w0	1	Output	Write Enable. This signal, when low indicates that a write operation is to occur.
d_dtoe0	1	Output	Mode control for VRAM.
d_sc1	1	Output	VRAM serial shift clock.
d_se0	1	Output	VRAM shift enable.

Bus Interface

Name	*#*	*Type*	*Description*
ad[63:0]	64	In/Out	Multiplexed address/data bus. In low cost configurations, only ad[31:0] are bonded out.
d_le	1	Output	Data Latch Enable. Indicates when active that the ad bus contains data which should be latched on the next sys_clk edge.
d_rga	1	Output	Register Address Enable. Indicates when active that the ad bus contains an address which should be latched on the next sys_clk edge.

PA/50 Interface (See Hitachi PA/50 Spec for more information.)

Note that in low cost configuration, none of these signals are bonded out.)

Name	*#*	*Type*	*Description*
pa_ad[31:0]	32	In/Out	PA/50L Address/Data Bus
pa_adp[3:0]	4	In/Out	PA/50L Data Parity
pa_perr0	1	Output	PA/50L Parity Error
pa_byte_cntl[3:0]	4	Input	PA/50L Byte Control
pa_cycle_start	1	Input	PA/50L Cycle Start
pa_r1w0	1	Input	PA/50L Read/Write
pa_rdy0	1	Output	PA/50L Ready
pa_sar0	1	Input	PA/50L Same Address Region
pa_cycle_type[1:0]	2	Input	PA/50L Cycle Type
pa_lock0	1	Input	PA/50L Lock
pa_int[4:0]	5	Output	PA/50L Interrupts

Peripheral Component Interface (PCI)

Name	*#*	*Type*	*Description*
pci_ad[31:0]	32	In/Out	PCI Mulitplexed Address/Data
pci_c1_be0[3:0]	4	In/Out	PCI Bus Command - Byte Enables
pci_par	1	In/Out	PCI Parity
pci_frame0	1	In/Out	PCI Cycle Frame
pci_trdy0	1	In/Out	PCI Target Ready
pci_irdy0	1	In/Out	PCI initiator Ready
pci_stop0	1	In/Out	PCI Stop
pci_devsel0	1	In/Out	PCI Device Select

pci_id_sel	1	In/Out	PCI Initialization Device Select
pci_req0[5:0]	6	Input	PCI Requests to arbiter
pci_gnt[[5:0]	6	Output	PCI Grants from arbiter
pci_clk	1	Output	PCI clock
pci_rst	1	In/Out	PCI reset

CD-ROM Interface

Name	*#*	*Type*	*Description*
cd_data	1	Input	CD-ROM drive data output
cd_bclk	1	Input	CD-ROM data clock
cd_lrck	1	Input	CD-ROM left-right clock
cd_c2po	1	Input	CD-ROM C2 error pointer
cd_intf	6	TBD	CD-ROM drive control interface. The current CD32 CD-ROM drive control interface requires 11 signals, almost all of which are low bandwidth signals... Many of these signals could be piggybacked onto the cd_data stream or encoded into their own serial stream. This would save a number of pins here at the expense of slightly more complex processing in the CD-DRIVE. Could the on-drive micro handle this??

Audio

Name	*#*	*Type*	*Description*
aud_data_in[1:0]	2	Input	Audio Serial Input Data. Each of these 2 lines contain CD serial formatted stereo audio.
aud_lrck_in[1:0]	2	Input	Audio Serial Input Left/Right Clock.
aud_bclk_in[1:0]	2	Input	Audio Serial Input Bclk
aud_data_out	1	Output	Audio Serial Output Data
aud_lrck_out	1	Output	Audio Serial Output Left/Right Clock
aud_bclk_out	1	Output	Audio Serial Output Bclk

Chip Selects

Name	*#*	*Type*	*Description*
ad_rom_cs0	1	Output	Chip select for AD bus resident ROM.
ad_video_cs0	1	Output	Chip select for AD bus resident VIDEO chip.
ad_other_cs0	1	Output	Chip select for AD bus resident auxlliary chip.
pa_rom_cs0	1	Output	Chip select for PA/50 bus resident ROM.
ad_config_cs0	1	Output	Chip select for AD bus resident config register.

Test Signals

Name	*#*	*Type*	*Description*
jtag_tclk	1	Input	JTAG Clock
jtag_tms	1	Input	JTAG Test Mode Select
jtag_di	1	Input	JTAG Data Input
jtag_do	1	Input	JTAG Data Output

Supplies

Name	*#*	*Type*	*Description*
Power	10	VDD	Power
Ground	20	VSS	Ground

1.2 PA-RISC Core

The computational core implements a level 1, Edition 3 of the PA-RISC 1.1 Architecture Reference Manual. SFU instructions have been added to enhance the performance of graphics functions.

1.2.1 Processor Datapath

Figure XXX shows a block diagram of the datapath which implements the PA-RISC instruction set. It is designed to allow 64-bit SFU instructions to be added to enhance graphics performance. Hence, the entire datapath is 64 bits in width. The datapath implements a traditional 5 stage integer pipeline. The thirty two PA-RISC integer registers are implemented as a register file of 16 64-bit registers. Bits are interleaved throughout the datapath to permit upper and lower 32 bit quantities to be accessed.

Additional logic has also been included to permit some floating point operations to be performed in hardware. A floating point register file has been included, again interleaving upper and lower 32 bit words to allow either 32 or 64 bit accesses to be performed. Other preliminary thoughts on floating point support hardware are shown in the diagram with dotted lines.

The datapath will be implemented using custom design techniques.

1.2.2 Processor Control

Processor control will be implemented using standard cell techniques.

1.2.3 MMU

At least 8 TLB entries.

1.2.4 Cache

At least 2K Instruction and 1K Data Cache.

1.2.5 SFU Instructions

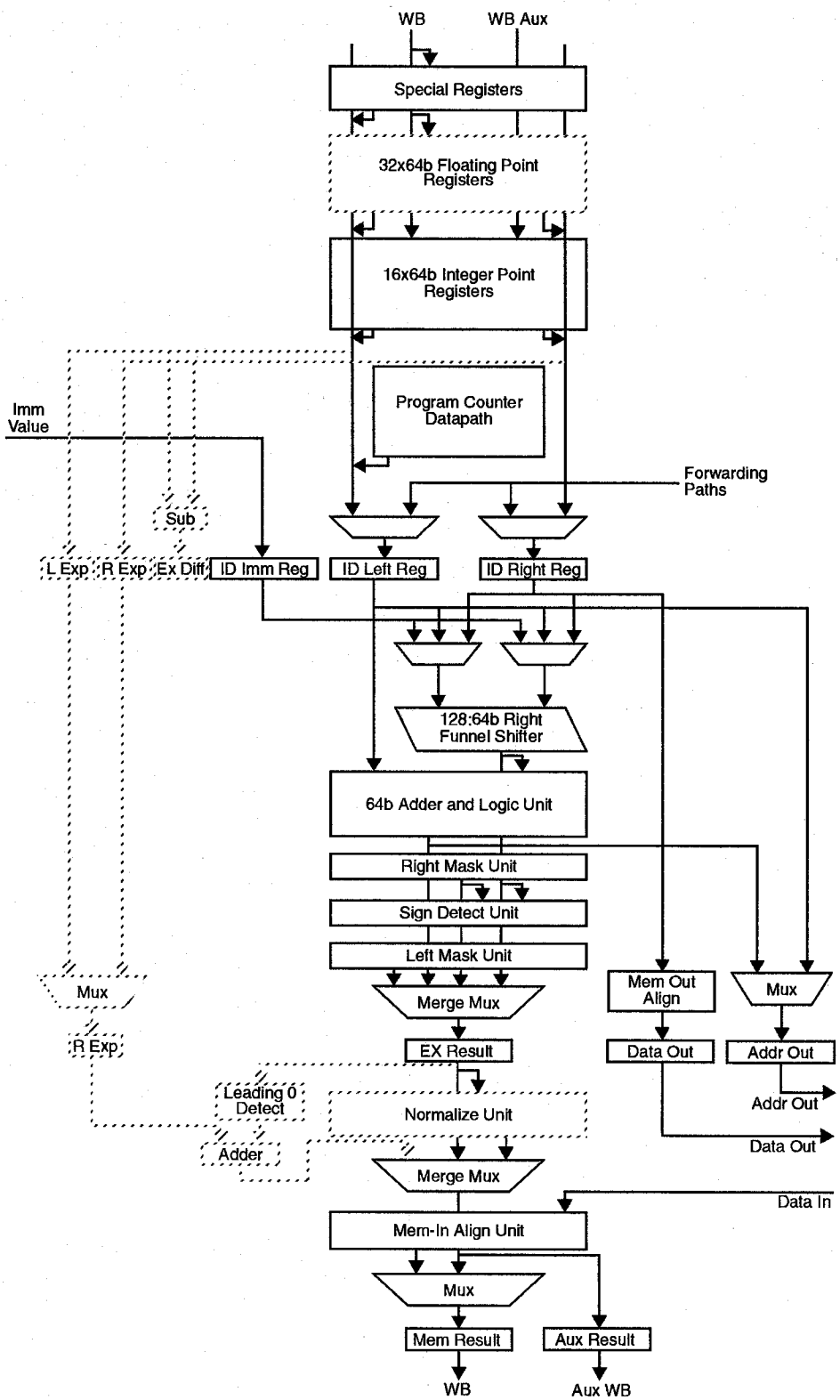
Instructions have been added to the PA-RISC instruction set using the SFU mechanism to enhance graphics performance. Further detail on the instructions is given in Appendix XX.

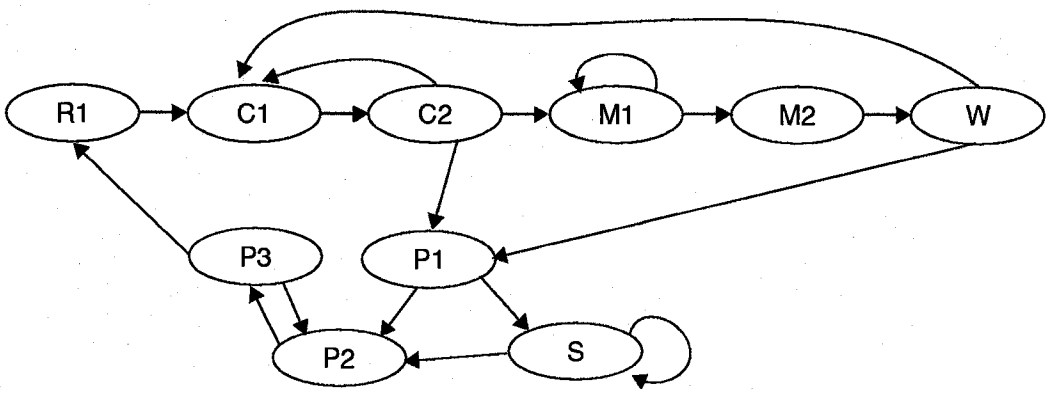
1.3 Display VRAM Control

Figure XXX shows the state diagram of the Display VRAM controller.

1.4 System DRAM Control

Figure XXX shows the state diagram of the System DRAM controller.





1.5 Blitter

Figure XXX shows a block diagram of the Blitter. It is capable of performing basic functions on either 8 or 16-bit pixel values. Blitting of 24-bit true-color data is accomplished by performing 3 8-bit blits. The blitter operates as a DMA device. Once setup, it autonomously manipulates memory to achieve the operation it was programmed for.

1.5.1 Blitter registers

The blitter registers are memory mapped locations. Each blitter control register consists of a writable register and a work register. The value written into the writable register is copied into the work register upon blitter startup. This permits the blitter to be setup while it is actively working on a previously set up job.

1.5.2 Blitter data types

The blitter operates on the following data types:

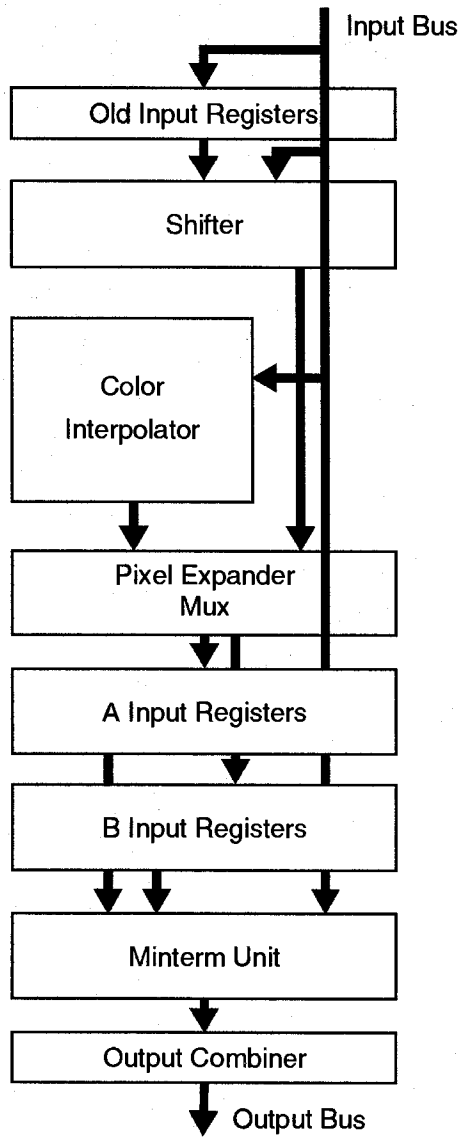
- 8-bit pixels: Each pixel is contained within a single byte. The byte represents one of 256 values which will either be used as an index into a color lookup table or as a direct value to the video D/As.
- 16-bit pixels: Each pixel is contained within a pair of bytes. The byte pair must exist on a half-word boundary. The sixteen bits represent 5 bits each of red, green, and blue. 16-bit pixels must exist on natural boundaries.
- 1-bit pixels: Each pixel is a single bit. This data type is only available as an expand source.

1.5.3 Blitter operations

The blitter performs the following operations:

1.5.3.1 Rectangular Blits

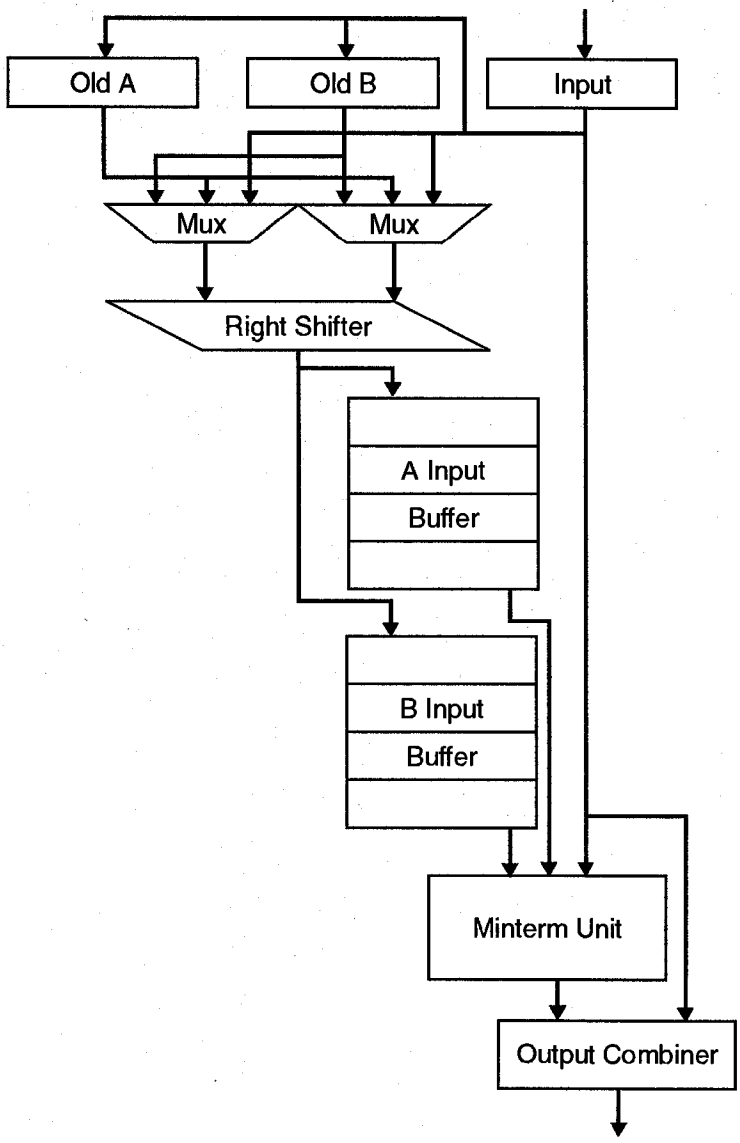
The blitter is capable of performing blits on three sources. Figure XXX shows the portion of the blitter datapath used for normal rectangular blits. One of the sources must also be the destination location. The data from the three sources may be combined using any of the possible 256 binary operations as specified by the minterm field. The destination data is modified via a memory read-modify-write operation.



1.5.3.2 Expands

The blitter is capable of performing 1:8 and 1:16 expands. A single bit source is expanded into either an 8-bit or 16-bit pixel by mapping all zeros of the source to the value held in BLT_COLOR_0 and all ones of the source to the value held in BLT_COLOR_1. Figure XXX shows that portion of the blitter datapath used for expands and line draws.

Before the expand is initiated, the BLT_COLOR_0 and BLT_COLOR_1 registers are loaded with color values which represent the colors to be expanded when 0's and 1's respectively are found in the source. Source data is read and either two or four bits at a time, depending on the expand mode (1:8 or 1:16) are used to select the values to be merged into the output register. Control circuitry

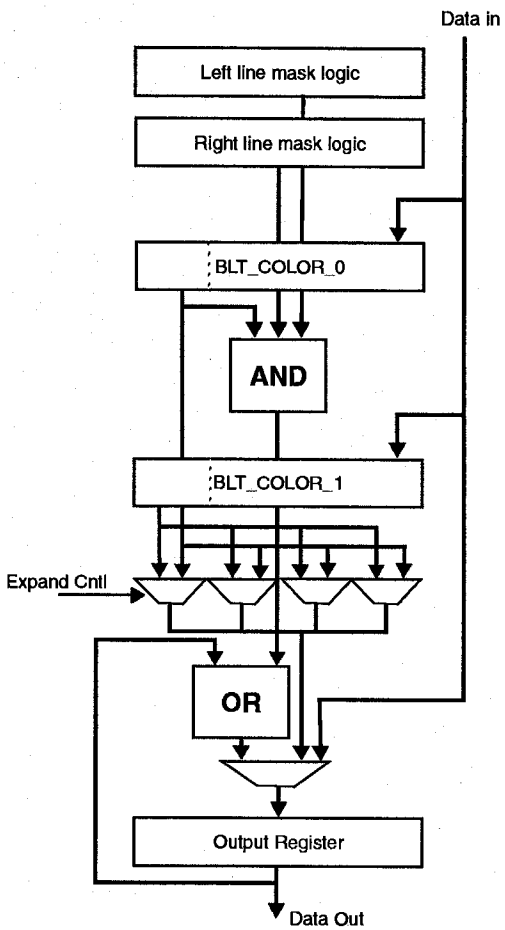


will keep track of whether all pixel positions or only some of the pixel positions of the destination are to be modified. Either a write or a read-modify write of the destination is performed.

1.5.3.3 Line Draws

The blitter is capable of performing 8-bit and 16-bit line draws using a Breshenham algorithm.

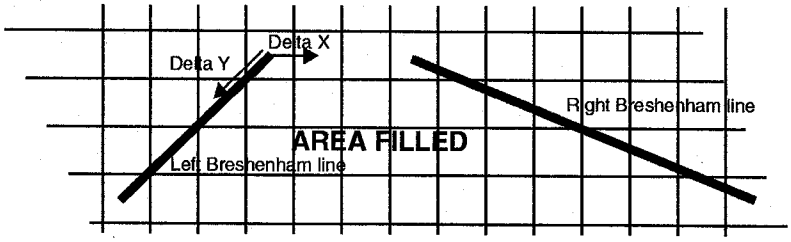
Before the line drawing function is selected, BLT_COLOR_0 is loaded with the color to be placed at the pixel locations representing the line.The Breshenham loop registers are loaded with the initial Breshenham error and error increment values. The Breshenham address generator is also loaded.



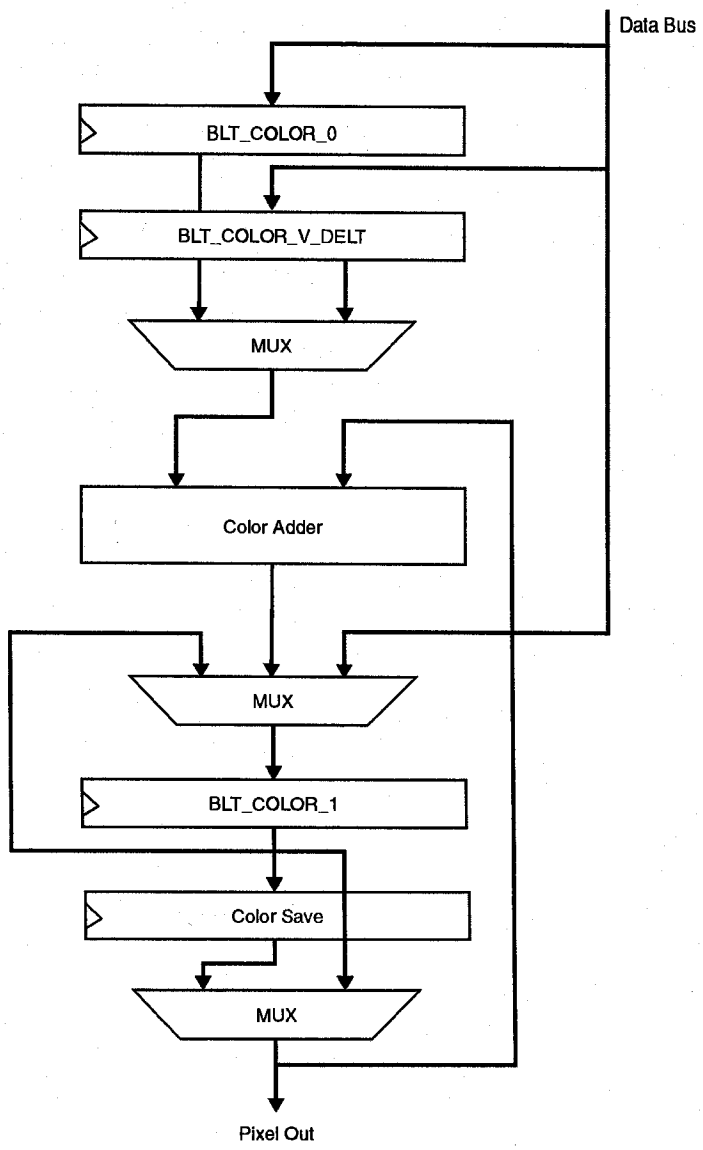
When the hardware is enabled, address bits are used to determine which pixel or half-word of the currently pointed to location should be written to. These bits are used to create left and right masks which block the unwanted bits of the BLT_COLOR_0 register. The resulting value is merged into the output register. Control circuitry will keep track of whether all pixel positions or only some of the pixel positions of the destination are to be modified. Either a write or a read-modify write of the destination is performed.

1.5.3.4 Shaded Fills

The blitter is capable of filling an area between two lines defined by a set of Breshenham address generators. Figure XXX shows an example fill operation.



The area between the Left Breshenham line the the Right Breshenham line is fill ed with a color value. The fill value is represented by an 8.24 fixed point number or a triplet of 5.5 fixed point numbers. It may be incremented by another fixed point value (shown as Delta X and Delta Y) thereby providing the capability of solid color or Gourand shaded fills. Figure XXX shows a block diagram of the color interpolator.



Before the fill begins, BLT_COLOR_1 is loaded with the initial color value (the pixel value at the upper left corner of the fill area. This value is also saved in the Color_Save register. BLT_COLOR_0 is loaded with the horizontal color delta value (the change in the color value as the scan proceeds right). BLT_COLOR_-

V_DELT is loaded with the vertical color delta value (the change in the color value when the scan line is incremented).

When the fill is operating, the value held in BLT_COLOR_1 is passed out as the current pixel. It is also fed back and incremented by the value held in BLT_COLOR_0 for each pixel in a scan line. This continues until the line terminates due to reaching the opposite side of the fill area as defined by the right line Breshenham address generator.

To move to the next line, the value stored in the Color_Save register is added to the value held in BLT_COLOR_V_DELT to determine the value of the first pixel of the next scan line (which is then loaded into BLT_COLOR_1 and saved in Color_Save).

Solid fills are accomplished by loading zero into the BLT_COLOR_0 and BLT_COLOR_V_DELT registers.

The Color Adder is a normal two's complement adder. Circuitry is added which breaks the carries out of bit positions 9, 19, and 29 when 16-bit pixels are being processed. Additional circuitry is added to detect carries out of those positions and either force 0 or the maximum value depending on the sign of the increment value being added.

1.5.4 Texture Mapping

The blitter performs a rudimentary form of texture mapping by using the fill hardware with an additional Breshenham DMA channel. This third Breshenham channel is used to source the pixels from the texture map to be used by the fill. This form of texture mapping does not filter pixels from the source, but rather skips to the next best source (texture map) pixel as the source for the fill. More sophisticated texture mapping will be accomplished via software running on the PA-RISC core.

1.5.5 Blitter memory operations

The blitter may burst up to 4 64-bit values per access to and from memory. Writes to memory may modify individual bytes using read-modify-write cycles. Destination accesses which do not require data to be fetched will perform write cycles only.

1.6 Audio Unit

1.7 CD-ROM Interface

The CD-ROM interface in Hombre is similar to the one included in the Amiga CD^{32} game console. It has been designed to permit read-ahead caching of CD-ROM sectors and software error correction with a minimum of overhead to the CPU. There are 3 separate interfaces betwen the system and the CD drive. They are all controlled by DMA accesses to and from memory.

1.7.1 Drive Control Interface

Control and status of the drive mechanics is accomplished through a DMA driven serial link to/from the CD drive. An area of memory is specified as a source circular buffer for commands and a receiving circular buffer for response using CD_CMD_ADDR. The addresses placed in these fields must be aligned to a 256 byte boundary, thereby creating a 256 byte circular buffer for each.

Each buffer also has associated with it an 8 bit index register and 8 bit compare register. Both are cleared at reset. Data is placed into the control buffer by the CPU and the relative buffer address of the last command is placed into the compare register. DMA is enabled, causing bytes to be extracted and sent to the CD drive, incrementing the index register as they go, until the index register becomes equal to the compare register. Further commands may be added to the buffer by the CPU while the DMA is active. The DMA activity will continue as long as the two registers are not equal. It is up to software to insure that the CPU does not get ahead of the DMA and overrun the circular buffer.

The receive buffer works in a similar way. The CPU expects a certain number of response bytes from the drive and modifies the receive compare register accordingly. When receive DMA is enabled, status data from the drive is placed in memory, starting at the index address and continuing until the index equals the compare register.

1.7.2 SubCode Interface

SubCode data from the drive is DMA'd into memory to the location specified by CD_SUBCODE_ADDR. This address must be aligned to a 256 byte address and specifies the memory area for two 128 byte subcode buffers. An 8 bit index register manages the data into the buffer.

Each subcode block consists of 98 bytes of data (including sync bytes). When a complete block has been transfered to memory, an interrupt is generated (when enabled) and the upper bit of the index register is inverted while the lower bits are cleared, thereby reversing the buffer halves.

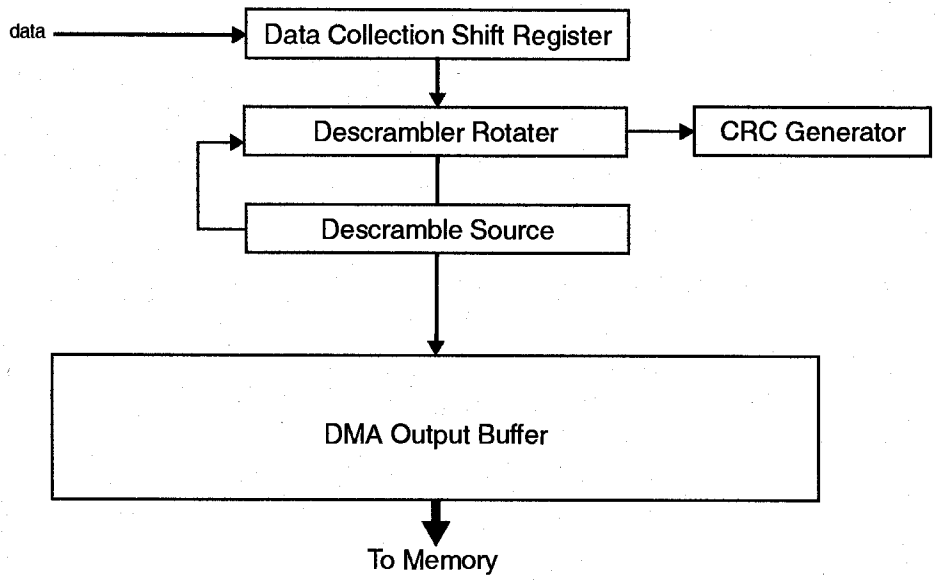
1.7.3 Data Interface

Data from a CD-ROM sector consists of header information, error detection information, error correction information, and the data itself. Additionally, each sector has status information from the decoder.

Since positioning of the CD drive head mechanism is not precise and many times multiple sectors are to be accessed. a block of sectors is requested. The drive is asked to position the pickup head at a particular location on the disk and the data DMA is enabled. Circuitry receives the bit stream from the drive and once a sync mark has been detected, information is placed into memory.

The memory area defined for CD data buffers is specified by the CD_DATA_ADDR register. This address must be aligned to a 64k byte boundary. A 4 bit index register and 4 bit compare register maintain individual 4k byte buffers. Data within each 4k byte buffer is stored at specific offsets.

Figure XXX shows a block diagram of the CD-ROM interface. It receives a 4



wire data interface from the CD drive and assembles data for delivery to the memory interface. Sync detection and descrambling are performed in hardware. The CRC portion of the CD error detection algorithm is also performed in hardware.

Data is collected and DMA'd to memory in 4 doubleword bursts.

Serial data is received from the CD drive into the data collection shift registers. Comparitors detect sync marks which specify sector boundaries. Once a sync mark has been detected, data is descrambled and error detection (CRC) is performed across the data.

Data words are counted, with header information and error detection and correction words being stored through one DMA pointer and data block words being stored through another DMA pointer.

1.8 Copper

The copper is a video line sychronized, programmable sequencer. It is capable of executing lists of instructions which load registers, waiting for specific video lines, or waiting for vertical blank. A simple one level subroutine mechanism is provided.

1.8.1 Instructions

Copper instructions must be aligned on 64-bit boundaries. The following instructions are executed by the copper.

1.8.1.1 Move Instruction

The move instruction is capable of loading up to 256 contiguous 64-bit registers from the locations following the move instruction. When the number of loads in-

dicates that one register is loaded, a field in the instruction determines which (or both) of the fullwords are to be written.

1.8.1.2 Wait Instruction

The wait instruction suspends copper instruction execution until the given line is reached. If the B bit in the instruction is set, the instruction after the wait is satisfied is taken from the address given in the instruction and the address of the next instruction in the stream is placed in the save register, otherwise it is taken from the next address in the current stream.

1.8.1.3 Wait Relative

The wait relative instruction clears a secondary line counter and suspends copper instruction execution until the given number of video lines have passed. If the B bit in the instruction is set, the instruction after the wait is satisfied is taken from the address given in the instruction and the address of the next instruction in the stream is placed in the save register, otherwise it is taken from the next address in the current stream.

1.8.1.4 Jump

The jump instruction causes the next copper instruction to be taken from the address given in the instruction.

1.8.1.5 Return

The return instruction causes the next instruction to be taken from the address in the save register.

1.8.2 External Interrupt Sources

The copper may be interrupted by one of the following:

1.8.2.1 Vertical Blank

The start of vertical blank causes the copper, when enabled to execute instructions from the location specified in the COPR_VB_ADDR register. The address of the previous instruction stream is lost.

1.8.2.2 Video Line Match

After a WAIT instruction has been executed, a match of the vertical video line counter with the requested line will cause the copper to resume excution from the address specified in the COPR_WAIT_ADDR register.

1.8.2.3 Relative Line Match

After a WAIT_RELATIVE instruction has been executed, an internal counter is cleared, then incremented at every line beginning. When the number of lines specified in the WAIT_RELATIVE instruction have passed, the copper resumes execution from the address specified in the COPR_WAIT_REL_ADDR register.

1.9 DMA Address Pointers

Figure XXX shows a block diagram of the DMA address unit.

1.9.1 Playfield Address Generators

1.9.2 Sprite Address Generators

1.9.3 CD-ROM Address Generators

1.9.4 Copper Address Generators

1.9.5 Audio Address Generators

1.9.6 Blitter Breshenham Address Generators

1.9.7 Blitter Blit Address Generators

Block Copy Address Generators

1.10 PCI Control

Provides a PCI bus interface for both the internal PA-RISC processor and for the external Hitachi PA-RISC processor.

In another mode, the PCI interface will act as a slave to a PCI based processor. Registers which have PCI addresses for the purpose of communications and control from PCI based processors.

Little-endian addressing.

1.11 External Processor Interface

1.0 Hombre Video Chip (Natalie)

The VIDEO chip provides the main interface to the video subsystem. Figure XXX shows a block diagram of the chip. The chip contains an interface to the graphics data bus, buffers for both scan line and sprite video, monitor control circuitry, and user interface (joystick, etc.) circuitry. Blocks which shift data from the buffers to the video DACs contain circuitry for effects. Appendix B contains a detailed description of the behavioral schematic.

1.1 Video Pins

Clock signals

Name	*#*	*Type*	*Description*
sys_clk	1	Input	50 Mhz system clock
sys_reset0	1	Input	System reset
pix_clk	1	Input	PIxel Clock
uart_clk	1	Input	UART Clock

Display VRAM Interface

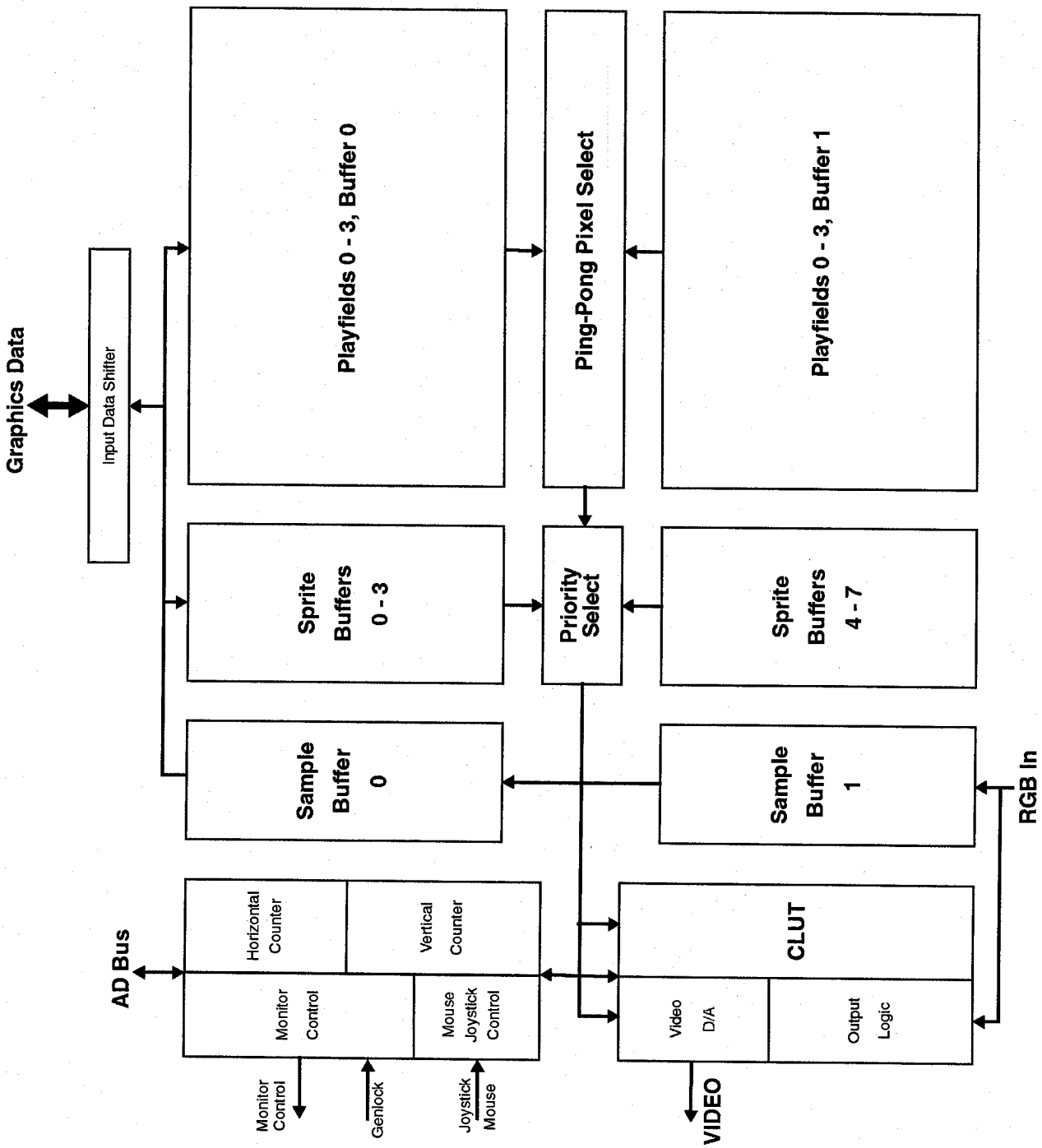
Name	*#*	*Type*	*Description*
d_sc1	1	Output	VRAM serial shift clock.
d_se0	1	Output	VRAM shift enable.

Bus Interface

Name	*#*	*Type*	*Description*
ad[63:0]	64	In/Out	Multiplexed address/data bus. In low cost configurations, only ad[31:0] are bonded out.
d_le	1	Output	Data Latch Enable. Indicates when active that the ad bus contains data which should be latched on the next sys_clk edge.
d_rga	1	Output	Register Address Enable. Indicates when active that the ad bus contains an address which should be latched on the next sys_clk edge.
gd[63:0]	64	In/Out	Graphics data bus. In low cost configurations, only gd[31:0] are bonded out.

Video and Monitor Control

Name	*#*	*Type*	*Description*
vd_csync0	1	Output	Composite Sync
vd_vsync0	1	Output	Vertical Sync
vd_hsync0	1	Output	Horizontal Sync
vd_vblank0	1	Output	Vertical Blank
vd_color_burst0	1	Output	Color Burst
vd_lpen0	1	Input	Light Pen
vd_clk_ext	1	Input	External Clock input (genlock)
vd_clk_ext_en	1	Input	External Clock enable (genlock)
vd_a_red	1	Output	Analog Red Video
vd_a_green	1	Output	Analog Green Video
vd_a_blue	1	Output	Analog Blue Video
vd_d_red	8	In/Out	Digital Red
vd_d_green	8	In/Out	Digital Green
vd_d_blue	8	In/Out	Digital Blue



Mouse Port

Name	*#*	*Type*	*Description*
mouse_rcv	1	Output	Serial Mouse receive data (To mouse)
mouse_xmt	1	Input	Serial Mouse transmist data (From mouse)
mouse_rts	1	Input	Serial Mouse request to send
mouse_cts	1	Output	Serial Mouse clear to send
mouse_dsr	1	Input	Serial Mouse data set ready
mouse_dtr	1	Output	Serial Mouse data terminal ready

Game Controller / Joystick Ports

Name	*#*	*Type*	*Description*
joy0_fwd	1	Input	Joystick 0 Forward or Mouse V
joy0_back	1	Input	Joystick 0 Back or Mouse H
joy0_left	1	Input	Joystick 0 Left or Mouse VQ
joy0_right	1	Input	Joystick 0 Right or Mouse HQ
joy0_but_0	1	Input	Joystick 0 Pot X or Button 0
joy0_but_1	1	Input	Joystick 0 Pot Y or Button 1
joy0_but_2	1	Input	Joystick 0 Fire or Button 2
joy1_fwd	1	Input	Joystick 1 Forward or Mouse V
joy1_back	1	Input	Joystick 1 Back or Mouse H
joy1_left	1	Input	Joystick 1 Left or Mouse VQ
joy1_right	1	Input	Joystick 1 Right or Mouse HQ
joy1_but_0	1	Input	Joystick 1 Pot X or Button 0
joy1_but_1	1	Input	Joystick 1 Pot Y or Button 1
joy1_but_2	1	Input	Joystick 1 Fire or Button 2

Test Signals

Name	*#*	*Type*	*Description*
jtag_tclk	1	Input	JTAG Clock
jtag_tms	1	Input	JTAG Test Mode Select
jtag_di	1	Input	JTAG Data Input
jtag_do	1	Input	JTAG Data Output

Supplies

Name	*#*	*Type*	*Description*
Power	10	VDD	Power
Ground	16	VSS	Ground

1.2 Display Technology

The video chip contains the circuitry which buffers and displays video data. Up to 4 video streams and up to 16 sprites may be fetched from memory to be displayed. The number of sprites is implementation dependent. The way in which the video streams and sprites interact is a function of the mode in which the chip is placed and the priorities associated with each stream.

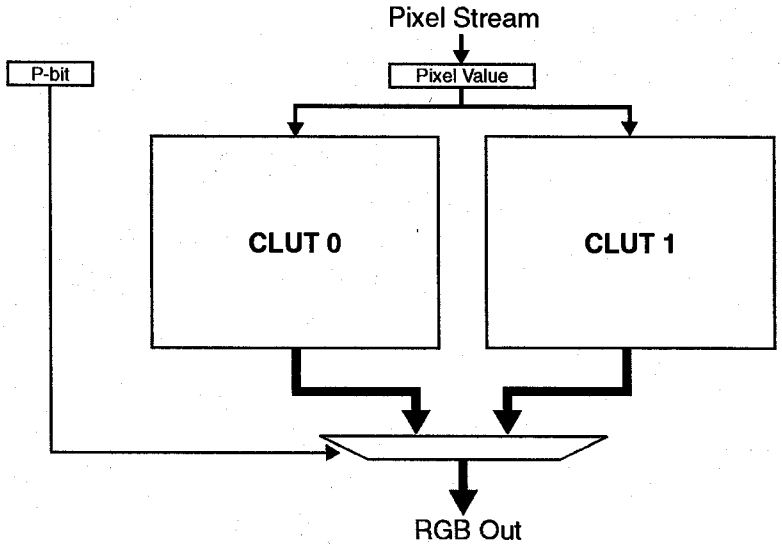
1.3 Video modes

All video is displayed from canvases held in display memory. Up to four streams of data may be fetched for each line of video data. The use of these streams is

determined by the mode field of the VID_CNTL register. Each stream presents a chunky pixel for display each pixel time. The four streams of data are used according to the specified mode for each playfield. Note that not all combinations of playfield modes necessarily make sense.

1.3.1 Palette Index Mode

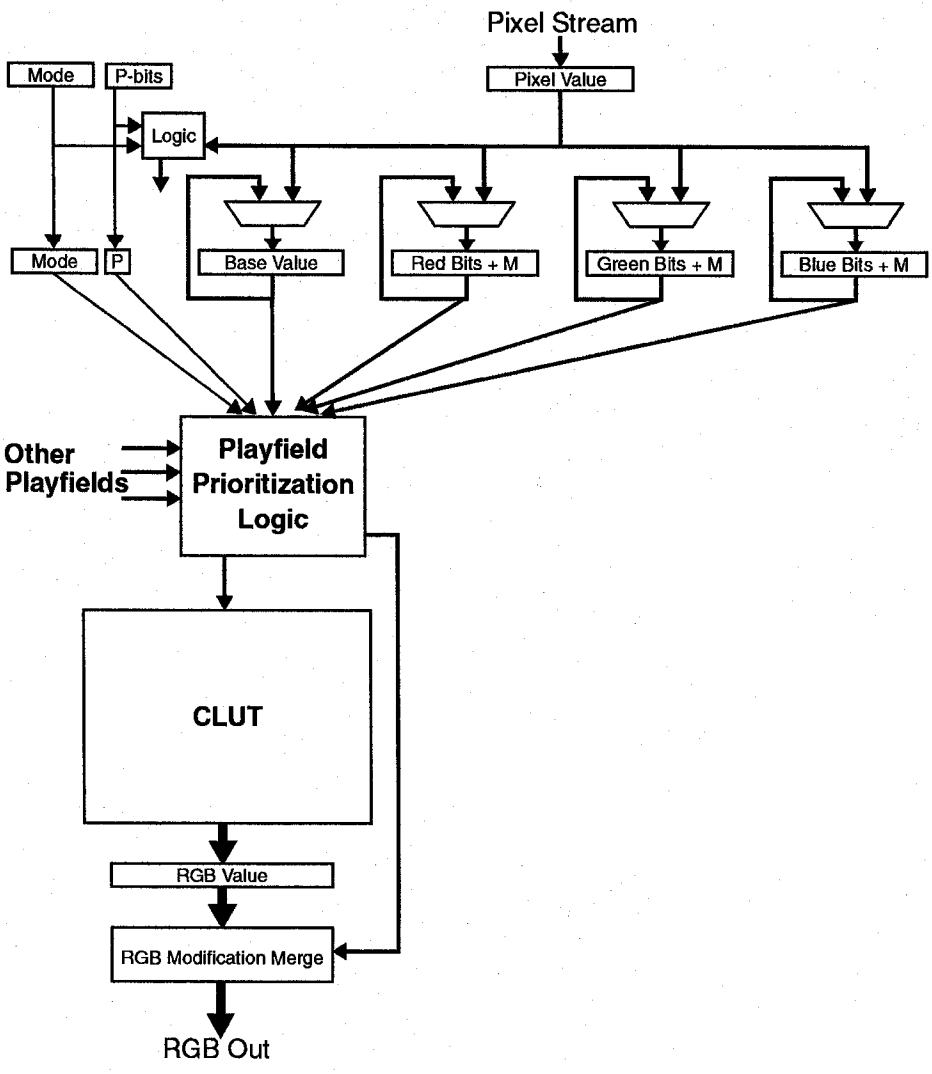
The 8 bits of data presented for each pixel is interpretted as an index into a color lookup table. The resulting 24 bits (8 bits each of RGB) of data is sent to the video DACs. The eight bit index is presented to one of the two 256x24 CLUTs. The CLUT used is determined by the least significant bit in the P field of the playfield control register. This process is shown in Figure XXX.



1.3.2 HAM8 Mode

The 8 bits of data presented for each pixel is interpretted as a HAM8 value. Figure XXX shows the HAM8 interpretation process. Each 8 bits of HAM8 data consists of a 2-bit control field and a 6-bit value field. The control field specifies one of four ways in which the value is to be used. A control field of 0b00 indicates that the value field is a base RGB value index. This index is concatenated onto the P bits in the playfield control register to obtain an 8-bit color index value. This value is used to obtain a 24-bit RGB value as perfromed in the Palette Index mode. Control fields of 0b01, 0b10, and 0b11 contain modifiers for the red, green, and blue RGB values respectively. This value replaces the least significant 6 bits of the RGB value before being sent to the D/A.

Since each of the playfields is capable of being placed in HAM8 mode and since it is unknown at any given pixel time which playfield will be active (as a function of playfield priority, sprite priority, etc.), in order to maintain the current values for all playfields, either the CLUT must be multiported or additional logic must be added to each playfield datapath. The figure shows additional logic at the source of each pixel stream which determines the HAM8 base value and color modification values for that channel. HAM8 data is calculated for each pixel time



regardless of whether the pixel from that playfield will be displayed or not. This allows the value to be maintained even when other playfields or sprites have higher priority. When a new base value emerges from the pixel stream, it is captured into the base value register and the M bits in the color modification registers are cleared. Any modification value which is received is captured into the appropriate color register and the M bit for that color is set.

The base value for each playfield is prioritized among other playfields and sprites. The P-bits are carried along with the value through the prioritization logic. When a HAM8 value emerges as the highest priority pixel, the base value concatenated with the P-bits for that playfield is used to lookup a 24-bit RGB value. The color modification values for that playfield are then applied (as a function of the M bit being set) before passing the value to the D/A's.

A base value of zero with no modification is interpretted as transparency by the prioritization logic.

1.3.3 Direct Mode

The 8 bits of data presented for each pixel is sent directly to the D/A. Playfield 0 is connected directly to the red D/A. Playfield 1 is connected direclty to the green D/A. Playfield 2 is connected directly to the blue D/A. This mode is only useful when playfields 0, 1, and 2 all are configured in this mode, thus providing 24-bit true color images. A direct mode enabled on playfield 3 is ignored.

When playfield 3 is enabled (in some other mode), any non-transparent value on playfield 3 is given priority over the direct channels. This allows a 24-bit true value image to have a playfield overlay.

1.3.4 Combined Mode

Playfields 0 and 1, and playfields 2 and 3 may be attached to form 16-bit high color pixels. To enable this mode, the mode field on playfield 0 or 2 (primary) is set to combined mode and the corresponding playfield's (secondary) mode is disabled. All other specifications for the playfield are taken from the primary playfield control registers.

The 16-bit values come from contiguous halfwords and are stored in the line buffers for the primary and secondary playfields. The 16 bits are interpretted (from left to right) as 1 bit of reserved data, 5 bits of red data, 5 bits of green data, and 5 bits of blue data. A value of zero is interpretted as transparency.

1.3.5 Priority Mode

Priority mode is only available on playfield 0. When enabled, playfield 0 pixel values are interpretted as the priority value for the associated playfield 1 pixel. The playfield 1 control register priority value is ignored. This mode permits each pixel in playfield 1 to have its own priority value.

1.3.6 Mapped Mode

1.3.7 Mapped Video mode

Four streams of video data are fetched for each video line. For each pixel position, the data from play-field 0 is used to interpret the way in which the other three video data streams are used. Each 8 bit data item from play-field 0 is interpretted as:

7 6 5 4 3 2 1 0

T <--->
ID

where:

T: True color. When 1, play-fields 1 through 3 are sent directly out as 24 bit true color. When 0, the pixel is controled by the area ID.

ID: Area ID. Use the specified VID_AREA register to control the pixelbeing displayed.

1.4 Display Scaling

Each playfield control register has fields which permit pixels and lines to displayed up to 8 times. The three bit field for each expand direction holds the number - 1 times that the pixel/line will be displayed. For example, a value of 0 in both fields indicates that the pixel and line should be displayed once.

1.5 Screen Priority

At any given pixel location of the active display, up to 20 sources of pixel informationcan be valid and contending for the D/A converters: these are the 4 playfields and the 16(8) sprites. The data which is ultimately displayed is a function of the display mode, the value of the pixel, and its priority.

1.5.1 Priorities

Priorities are interpetted as 8 bit unsigned values. Value 0 has highest priority. Value 255 has lowest priority.

1.5.2 Sprite Priority

Each sprite has associated with it a priority value. This priority is specified in the PRIORITY field and selects one of 256 priority values for this sprite. During each pixel time, the priorities of the eight possible sprites are compared along with their pixel values to determine which sprite's pixel should be displayed. Should two sprites contain identical priorities and be non-transparent, the sprite originating from the lower numbered sprite channel will be favored.

A pixel value of zero indicates transparency and forces its priority to a value lower than priority 255. Hence, a pixel value of 0 overrides the sprite's specified priority. A sprite that is not active during a particular pixel time has its pixel value forced to zero, thereby forcing the pixel not to be selected.

1.5.3 Play-field Priorities

For video modes where 2, 3, or 4 playfields are active, each play-field has associated with it a priority value. This priority is specified in the PF_PRI field of the VID_CNTL_DISPLAY register. During each pixel time, the priorities of the four playfields are compared along with their pixel values to determine which playfield's pixel should be displayed. Should two play-fields contain identical priorities and be non-transparent, the play-field originating from the lower numbered play-field will be favored.

A pixel value of zero indicates transparency and forces its priority to a value lower than priority 255. Hence, a pixel value of 0 overrides the playfield's specified priority. A play-field that is not active during a particular pixel time has its pixel value forced to zero, thereby forcing the pixel not to be selected.

1.5.3.1 True color with chunky overlay mode

For the video mode where three of the play-fields represent RGB values and the fourth represents a chunky overlay plane, the true color pixel will be displayed anywhere that the overlay plane has the transparent value.

1.5.3.2 Prioritized chunky pixels

For the prioritized chunky pixel modes, play-field 0's values are used as the priority levels for play-field 1. Other play-fields, when used, continue to use the priority specified in the VID_CNTL register. This mode permits each pixel in play-field 1 to have its own priority value.

1.5.3.3 Combined priorities

The highest priority sprite and highest priority chunky pixel are prioritized to determine the pixel value to be displayed. In true color modes, this pixel value is then evaluated for transparency to determine whether it or the true color value should be displayed.

1.6 Display specification

The video display is controlled in both the horizontal and vertical directions by sync and blanking signals. The sync signals control when a line or frame should begin while the blanking signals control which part of a horizontal line and which lines contain visible data. Shifting video data onto the screen during the time that blanking is not enabled causes video images to be displayed.

1.6.1 Horizontal Control

Horizontal scan control is implemented as a 12 bit counter running at the pixel clock rate. Twelve bits permit a horizontal line of up to 4096 counts. Various registers are provided which contain values which are compared to the horizontal counter. The signals produced by these matchers are used to generate the monitor horizontal controls.

1.6.1.1 Horizontal Line Size

The number of pixels in a video line is defined in the VID_HORIZ_TOTAL register. Interlaced displays require that the center of the horizontal line also be specified in this register. When the horizontal counter reaches this value, it is reset to 0.

1.6.1.2 Horizontal Sync

A horizontal line begins with the assertion of HSYNC. The position of HSYNC relative to the value in the horizontal counter is controlled by the VID_HORIZ_SYNC register. Traditionally, HSYNC should begin at count 0.

1.6.1.3 Horizontal Blank

The position of the HBLANK signal relative to the value in the horizontal counter is controlled by the VID_HORIZ_BLANK register. HBLANK normally starts at the end of a line, before the HSYNC signal for the next line and ends at the point where the display should start.

1.6.1.4 Horizontal Window

Horizontal window determines where on a line video data should be actively displayed. Data is displayed (shifted through the D/A's) when the window is active. The position of the window is controlled by the VID_HORIZ_WINDOW register.

1.6.1.5 Horizontal Equalization

1.6.1.6 Horizontal Serration

1.6.1.7 Horizontal Burst

1.6.2 Vertical Control

Vertical scan control is implemented as a 13 bit counter. This counter is incremented at a half-line rate. Hence, its value is changed at the point defined by the VID_HORIZ_TOTAL.HCENTER and VID_HORIZ_TOTAL.HTOTAL points. Thirteen bits permit a screen of up to 8192 counts. 8192 half-lines represents 4096 lines. Various registers are provided which contain values which are compared to the vertical counter. The signals produced by these matchers are used to generate the monitor vertical controls.

1.6.2.1 Vertical Frame Size

The number of lines in a video frame is defined in the VID_VERT_TOTAL register.

1.6.2.2 Vertical Sync

A video frame begins with the assertion of VSYNC. The position of VSYNC relative to the value in the vertical counter is controlled by the VID_VERT_SYNC register. Traditionally, VSYNC should begin at line 0.

1.6.2.3 Vertical Blank

The position of the VBLANK signal relative to the value in the vertical counter is controlled by the VID_VERT_BLANK register.

1.6.2.4 Vertical Window

Vertical window determines which lines of the frame should contain active horizontal lines. Data is only displayed while the window is enabled. The position of the vertical window relative to the value in the vertical counter is controlled by the VID_VERT_WINDOW register.

1.6.2.5 Vertical Equalization

1.7 Video Ping-Pong Buffering:

In order to take maximum advantage of the available memory bandwidth from Dynamic VRAMs, static line buffers are provided to receive the "next" line's video data and shift it out at a constant rate during the active line time. This allows the Display VRAMs to accomodate the RAS/CAS pauses they need for page crossings, etc. across a complete line access. It also permits the entire line time (including the blanking time) to be used for the fetches, and permits multiple playfields to be fetched sequencially for simultaneous display. Two buffers are provided: while one is being used to provide data to the display, the other is being filled from the display memory.

This buffering also affords the opportunity to horizontally scroll video data as it is being fetched. As data is fetched for buffering, the word fetched along with the previously fetched word are shifted by the number of pixels represented by the least 3 bits of the frame start address (VID_PTR_x). The shifted pixels are stored in the line buffer. When the complete line has been fetched, it represents the (already) horizontally scrolled pixels.

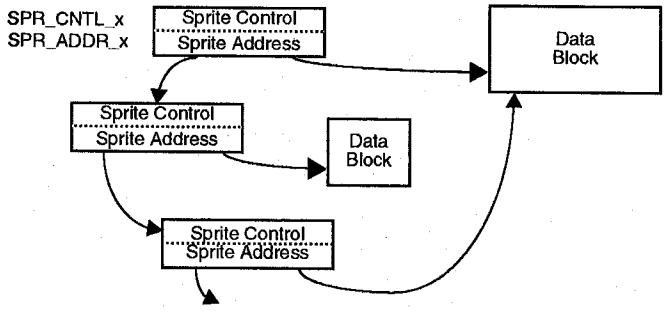
When the next line time occurs, the buffers are exchanged. The data in the active display buffer (the one just filled) is shifted out and used according to the video mode specified in the VID_CNTL register. The other buffer (the one used to shift out data during the previous line time) is now filled from display memory.

The active horizontal display occurs during the time in which horizontal blanking is not active. Therefore, the buffers are switched and fetching for a new line begins on the "un-assertion" of horizontal blank (HBSTOP).

Since data being fetched at any given time will not be seen until the next line time, the control associated with that data must also be delayed. Any video register written with new data will not be used until the next line time.

1.8 Sprites

Sprites are image objects which may be displayed independently of the video streams and each other. Hardware is provided for eight sprite channels. A sprite channel controls a sprite image through the servicing of a sprite control list. A sprite control list consists of a linked list of pairs of 64 bit doublewords, each of which makes up a sprite control block. This block is written into the SPR_ADDR_x and SPR_CNTL_x registers for the associated sprite, and remains in effect until the specified number of lines of image data for the sprite have been fetched. The SPR_CNTL_x contains fields which indicate when the sprite data should be displayed. When the VSTART line matches the early vertical counter, the pixels pointed to by the DATA_ADDRESS are fetched. The width of the sprite is also specified in the SPR_CNTL_x register. Sprite data is placed into ping-pong buffers similar to those used by the video streams. When the specified number of lines has been fetched, the next sprite control block is fetched from the NEXT_ADDRESS field. A sprite using the same sprite channel may begin on the next visible display line. Figure XXX shows a sprite channel.



Sprite pixels are chunky pixels. The value 0x00 is reserved as the transparency color. The width of sprites is fixed at 8, 16, 32, 64, or 128 pixels. Sprite data must be stored at a naturally aligned address. This insures that a fetch of a line of the sprite will not cross a page boundary. Sprite pixels are used as an index into a color lookup table.

Separating the control and data portions of a sprite specification lead to the following advantages over the existing mechanism:

- Sprite data may be shared by multiple sprites. Since the sprite control block simply points to the data rather than contain it, multiple sprites may easily point to the same data.
- Sprites may be easily animated. Since the sprite control block points to the data to be used by the sprite, multiple blocks may be present in off screen memory, each of which is pointed to at subsequent frames to achieve an animation effect. Alternatively, since the sprite data area is itself a canvas, graphics can be rendered into it.
- A sprite could point to an area of video memory where a frame-grabber is depositing its results. This could allow "a video window" to be moved around the display via one or more sprites.

1.8.1 Sprite data and sizing information

Sprite data is held in an appropriately aligned canvas. The number of pixels fetched from the DATA_ADDRESS and shifted onto the display is determined by the WIDTH field. The H_EXPAND field specifies the number of display pixels produced for each sprite pixel. This allows a sprite to be magnified by 1, 2, 4, or 8 as it is being displayed. A non-zero REPEAT field causes the data from a sprite line to be repeated horizontally, REPEAT+1 times. The sprite data is shifted repeatedly (as modified by the H_EXPAND field) until the repeat count is exhausted or the horizontal blank signal indicates the end of the display line.

The height of a sprite is determined by the HEIGHT and V_EXPAND fields. The HEIGHT field, shifted left by 0, 1, 2, or 3 as determined by the V_EXPAND field is added to the VSTART field to determine the line where the sprite should stop. The V_EXPAND field specifies the number of displayed lines (1, 2, 4, or 8) which should use a line of sprite data.

The UPDATE field specifies the value to be added to the fetch address of one sprite line to advance to the fetch address of the next sprite line.

1.8.2 Sprite positioning

A sprite may be positioned at any pixel location of the display. The VSTART and HSTART fields determine where the upper left pixel specified by DATA_ADDRESS will be positioned on the display.

1.8.2.1 Top edge condition

Sprites may be partially displayed at the top of the display (to cause the effect of the bottom of a sprite appearing at the top of a screen and subsequently moving down the screen) by adjusting the DATA_ADDRESS to the desired sprite line

and adjusting the height of the sprite. For example, if the DATA_ADDRESS points to the last line of the sprite, the height is set to one, and the VSTART value is set to the first visible line of the display, a single sprite line will be displayed at the top of the screen. At some later frame, the DATA_ADDRESS can be adjusted back by one line and the height increased by one to cause the appearance of the sprite moving down one line.

1.8.2.2 Left edge condition

Sprites may be partially displayed at the left of the display (to cause the effect of the right edge of a sprite appearing at the left of a screen and subsequently moving to the right) by adjusting the HSTART value. When the width of the sprite is greater than the HBSTOP value, the sprite can not be entirely hidden in the blanking area of the horizontal line. This is especially true for wide (i.e 128 pixel) sprites. In these cases, the DATA_ADDRESS may be adjusted to a point further into the sprite and the WIDTH adjusted to a more narrow sprite. The start point can now be adjusted to obtain the desired effect. (For example, if HBSTOP is 100 and a 128 pixel sprite is to have only its right edge shown, the DATA_ADDRESS must be adjusted to point to the 64th pixel, the WIDTH is set to 64 pixels, and the HSTART is set to 37.)

1.8.2.3 Bottom and right edge conditions

Blanking discontinues display of sprites.

1.8.3 Sprite priority

Each sprite is assigned a priority code in the PRIORITY field. The use of this field is discussed in the Screen Priority section.

1.9 Genlock

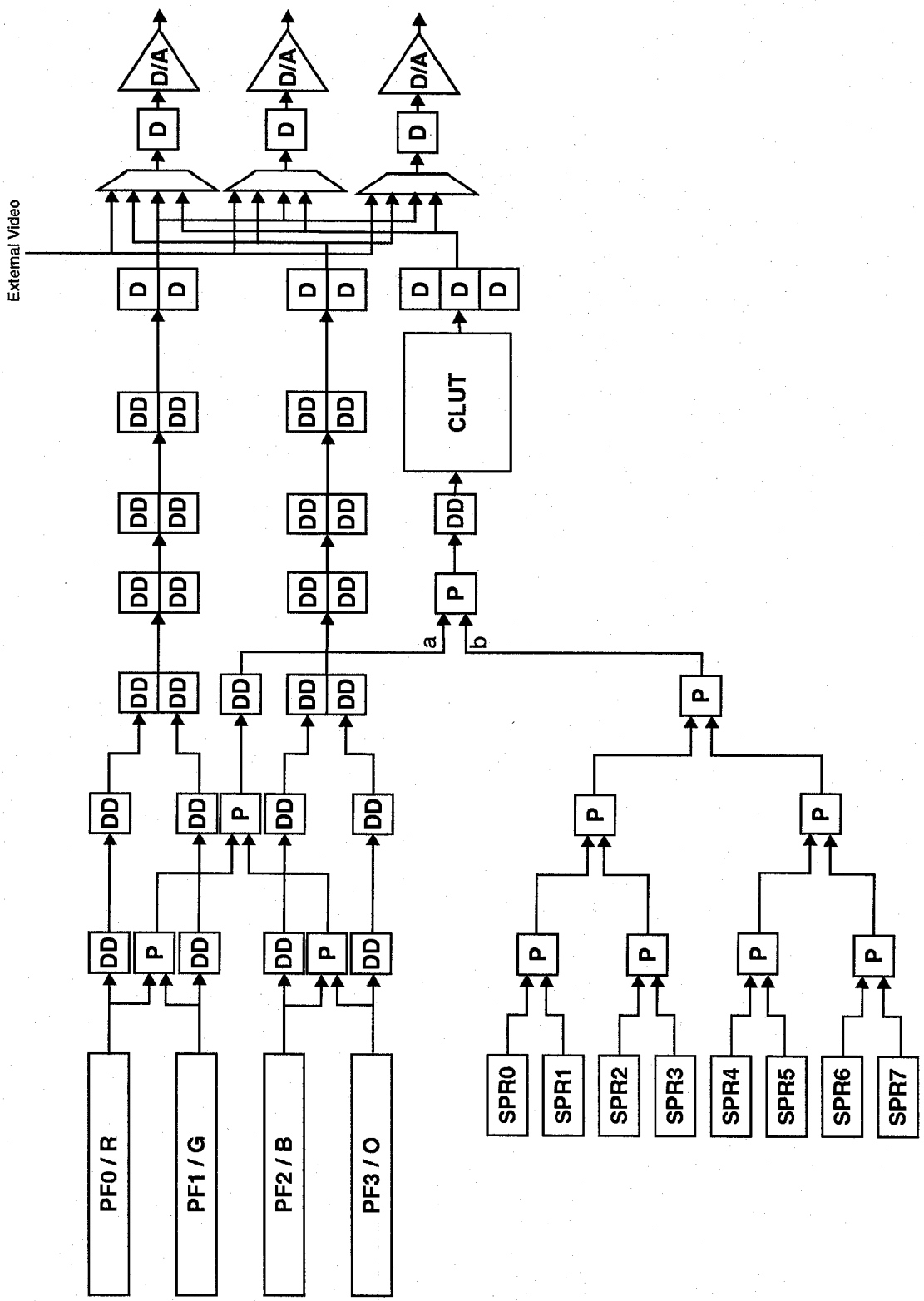
1.10 Video Buffer

1.11 Output Processing

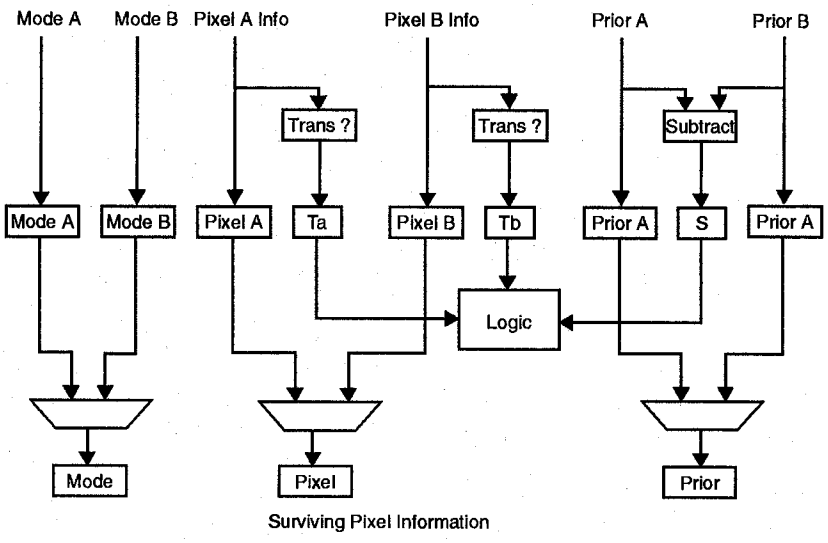
Figure XXX shows a block diagram of the video output processing. This section accepts pixels from the appropriate playfield and sprite buffers and determines, based on mode, priority, etc. which pixel should be sent to the video D/A converters for display.

In this diagram, blocks labeled PFx represent the current playfield buffers. Note that 16-bit video modes use playfield pairs (0/1 or 2/3). Playfields 0 through 3 may be used individually or in parallel depending upon the video modes selected. Note that when the playfields contain true color pixels, playfield 0 always contains red data, playfield 1 always contains green data, and playfield 2 always contains blue data.

Blocks labeled P are pipelined priority blocks. Each priority block requires two clocks and passes the surviving data and priority of the two entering streams as



a function of each stream's priority and whether the data value is transparent. Figure xxx shows the functionality of such a prioritizer.



Blocks labeled DD are two clock pipeline delays.

Blocks labeled D are single clock pipeline delays.

An extra stage of pipelining has been introduced before the CLUT to permit an additional 8 sprites to be added in some implementations and retain the same pipeline length.

The four playfields each present a byte of video data during each pixel time. For those video modes where each playfield is an independent video stream, the data is passed through the priority blocks, resulting in a pixel at the point labeled "a". For those modes where 16-bit pixels are used, or when 24-bit true color is selected, data passes through the delay flip-flops while the field's priority (in the case of 16-bit mode) is passed through the priority blocks.

Sprite buffers each present a byte of video data during each pixel time that they are active. When they are not active, the pixel is made transparent. The sprite data is passed through the priority blocks, resulting in a pixel at the point labeld-ed "b".

A prioritized pixel is presented to the CLUT which produces a 24-bit RGB value. This value is then MUX'd when appropriate with a 24-bit true color value which has been delayed. The resulting value is passed on to video D/As.

The pipeline lengh is 14 pixel clocks. Video control logic contains an equivalent number of delays such that the syncs, etc. match the video data.

1.11.1 Palette (CLUT)

The color look-up table (CLUT) consists of a 512 entry by 24 bit RAM. (This may be implemented as two 256 by 24 bit RAMs.) Since each pixel to be converted is only 8 bits, only 256 of the 512 values may be accessed. A control bit from the sprite or playfield control register determines which of the two 256x24 bit CLUTs should be used.

1.11.2 DACs