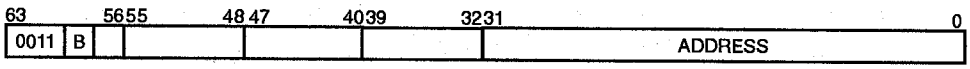
# Appendix A Copper Registers and Instructions

**Name:** **JUMP Instruction**

**Format:**




**Purpose:** The JUMP instruction causes the copper to fetch instructions from the location specified in the instruction. The B field specifies whether the next sequential instruction should be saved in the return register.

**Fields:**

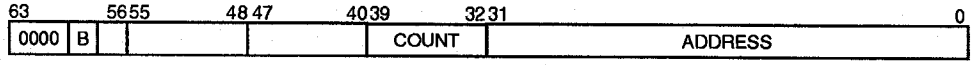
B: Specifies whether the return address should be saved in the return register. When B is zero, the return address is not saved. When B is one, the next sequential address is stored in the return register. This option permits simple single level subroutines to be performed.

LINE: Specifies the video line where instruction execution should resume.

ADDRESS: Specifies the location where execution should continue. This location must be on a 64-bit boundary.

**Name:** **MOVE Instruction**

**Format:**




**Purpose:** The MOVE instruction causes 1 to 256 of the doublewords following the instruction to be loaded to the contiguous address space specified in the instruction. If the MOVE is loading a single address, the B field specifies whether one or both of the fullwords are loaded.

**Fields:**

ADDRESS: The destination address for the data following the instruction. This address must be aligned to a 64-bit boundary.

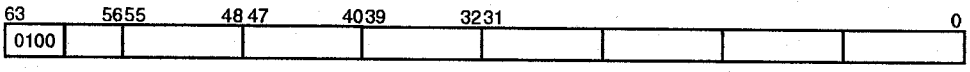
COUNT: The number of 64-bit words to transter.

B: When COUNT is 1, this field specifies which fullwords should be written according to the following:

| Code | Description |
|---|---|
| 01 | Low Fullword (bits 31:0) are written |
| 10 | High Fullword (bits 63:32) are written |
| 11 | Entire Doubleword written |

**Name:** **RETURN Instruction**

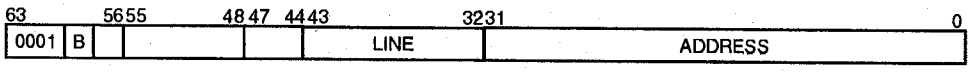
**Format:**




**Purpose:** The RETURN instruction causes the copper to fetch instructions from the location specified in the return register.

**Name:** **WAIT Instruction**

**Format:**




**Purpose:** The WAIT instruction causes the copper to suspend execution of instructions until the video line specified in the instruction has been reached. The B field specifies where the copper will fetch instructions from when the line is reached.

**Fields:**

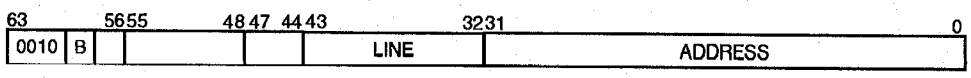
B: Specifies where the copper should fetch instructions from when the video line condition has been met. When B is zero, the copper simply stalls until the LINE specified is reached. Instructions execution is then resumed. When B is one, the copper stalls until the LINE specified is reached, but then continues execution at the ADDRESS specified in the instruction. The next sequential address is also stored in the return register when B is one. This option permits simple single level subroutines to be performed.

LINE: Specifies the video line where instruction execution should resume.

ADDRESS: Specifies the location where execution should resume when the B bit is set. This location must be on a 64-bit boundary.

**Name:** **WAIT_RELATIVE Instruction**

**Format:**




**Purpose:** The WAIT instruction causes the copper to clear an internal line counter and suspend execution of instructions until the number of video lines specified in the instruction have been passed. The B field specifies where the copper will fetch instructions from when the line is reached.

**Fields:**

B: Specifies where the copper should fetch instructions from when the video line condition has been met. When B is zero, the copper simply stalls until the LINE specified is reached. Instructions execution is then resumed. When B is one, the copper stalls until the LINE specified is reached, but then continues execution at the ADDRESS specified in the instruction. The next sequential address is also stored in the return register when B is one. This option permits simple single level subroutines to be performed.

LINE: Specifies the number of video lines that execution should be stalled.

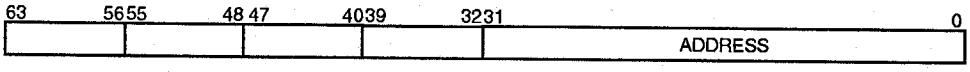
ADDRESS: Specifies the location where execution should resume when the B bit is set. This location must be on a 64-bit boundary.

**Name:** **COPR_VB_ADDR**

**Address:** **0x**

**Access:**

**Format:**




**Purpose:** COPR_VB_ADDR specifies the location where copper instructions are to be fetched from when a vertical blank interrupt is received.

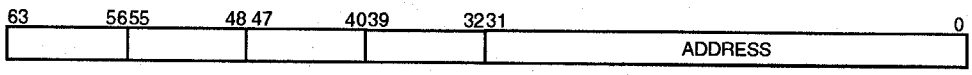
ADDRESS:Defines the address where the copper fetches instructions when a vertical blank interrupt is received. This address must be on a doubleword boundary.

**Name:** **COPR_WAIT_ADDR**

**Address:** **0x**

**Access:**

**Format:**




**Purpose:** COPR_WAIT_ADDR specifies the location where copper instructions are to be fetched from when a wait interrupt is received.

**Fields:**

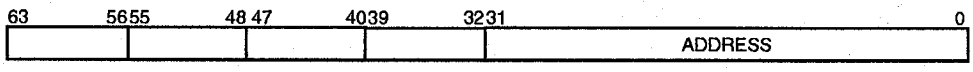
ADDRESS: Defines the address where the copper fetches instructions when a wait interrupt interrupt is received. This address must be on a doubleword boundary. This register is normally loaded from the WAIT instruction but may be overwritten.

**Name:** **COPR_WAIT_REL_ADDR**

**Address:** **0x**

**Access:**

**Format:**




**Purpose:** COPR_WAIT_REL_ADDR specifies the location where copper instructions are to be fetched from when a wait relative interrupt is received.

**Fields:**

| | |
|---|---|
| ADDRESS: | Defines the address where the copper fetches instructions when a wait interrupt relative interrupt is received. This address must be on a doubleword boundary. This register is normally loaded from the WAIT_RELATIVE instruction but may be overwritten. |

# Appendix A Sprite Registers

**Name:** **SPR_ADDR_x**

**Address:** **0x**

**Access:**

**Format:**

**Purpose:** SPR_ADDR_x, 0 <= x <= 15, controls the location of the data to be displayed with the associated sprite and the address of the next sprite control block.

**Fields:** DATA_ADDR: Defines the address of the data associated with this sprite. The address must be naturally aligned as a function of the sprite width (i.e. If the sprite is 32 pixels in width, the address must be on a 4-doubleword boundary. This insures that a fetch of a line of the sprite will not cross a page boundary.).

NEXT_ADDR: Defines the address of the sprite control block for the next sprite. The new SPR_CNTL_x and SPR_ADDR_x register will be loaded from this address when the current sprite is finished. The address of a sprite control block must be on a 2-doubleword boundary.

UPDATE: Address update amount in pixels: The increment amount selected is added to the original address of the sprite fetch to produce the address of the next line of sprite data.

| Incr Amt | Code | Pixels Added |
|---|---|---|
| 0001 | 8 | 0x000000008 |
| 0010 | 16 | 0x000000010 |
| 0011 | 32 | 0x000000020 |
| 0100 | 64 | 0x000000040 |
| 0101 | 128 | 0x000000080 |
| 0110 | 256 | 0x000000100 |
| 0111 | 512 | 0x000000200 |
| 1000 | 1024 | 0x000000400 |
| 1001 | 2048 | 0x000000800 |

E End of list:

| | |
|---|---|
| 0 | List continues at NEXT_ADDRESS |
| 1 | List is terminated after this sprite |

**Name:** **SPR_CNTL_x**

**Address:** **0x**

**Access:**

**Format:**

**Purpose:** SPR_CNTL_x, 0 <= x <= 15, controls the position of the associated sprite on the screen and the attributes and options associated with it.

**Fields:**

VSTART: Defines the display line where the first line of the sprite will become visible.

HSTART: Defines the pixel number where the left-most pixel of the sprite will become visible.

HEIGHT: Defines the number of lines contained in this sprite.

WIDTH: Width of sprite in pixels:

| Code | Pixels |
|---|---|
| 001 | 8 |
| 010 | 16 |
| 011 | 32 |
| 100 | 64 |
| 101 | 128 |

REPEAT: Repeat mode:

| | |
|---|---|
| == 0 | Use sprite data once |
| != 0 | Sprite line data is repeated horizontally. The number indicated is the count minus 1 of the times the data will be used. |

H_EXPAND: Indicates the number of pixels that are displayed for each sprite pixel.

| Code | Display pixels per sprite pixel |
|---|---|
| 00 | 1 |
| 01 | 2 |
| 10 | 4 |
| 11 | 8 |

V_EXPAND: Indicates the number of lines that are displayed for each sprite pixel.

| Code | Display lines per sprite pixel |
|---|---|
| 00 | 1 |
| 01 | 2 |
| 10 | 4 |
| 11 | 8 |

PRIORITY: The priority level of this sprite. The encoded value in this field is compared to the priority values of other sprites and playfields which are active at any given pixel time. The highest priority, non-transparent object is made visible. Lower priority numbers have the highest priority. When more than one object has the same priority level, the object with the lower physical number (Sprite 0 is lower than Sprite 3) is displayed.

**Name:** **SPR_DATA_x**

**Address:** **0x**

**Access:**

**Format:**

**Purpose:** SPR_DATA_x, 0 <= x <= 15, is the destination address used by DMA to fill the sprite data buffer. The sprite data buffers operate in a ping-pong fashion: one is being filled by the DMA, while the other, filled during the previous line time is shifted into the video stream when the horizontal beam counter matches the HSTART field. See "video ping-pong buffering" for an explanation of timing.

**Fields:** Each byte of the data register contains an 8 bit pixel. (8 pixels per fetch)

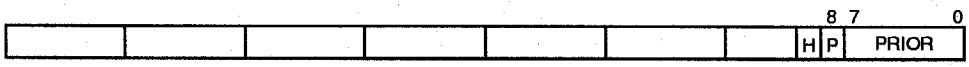
# Appendix A System Registers

**Name:** **SYS_CONFIG**

**Address:**

**Access:**

**Format:**




**Purpose:** SYS_CONFIG is automatically loaded from the AD bus at system reset time. It contains fields which define system configuration and may only be changed at reset time.

**Fields:**

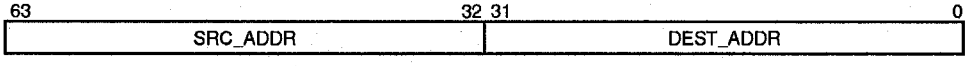
| | |
|---|---|
| AD_WS: | AD ROM wait states. Contains the number of clock cycles introduced when the AD ROM is addressed. |
| PA_WS: | PA ROM wait states. Contains the number of clock cycles introduced when the PA ROM is addressed. |
| D_MA: | Number of Display VRAM multiplexed address. |
| S_MA: | Number of System DRAM multiplexed address. |
| PCI_EN: | Enable PCI access to AD registers |

**Name:** **SYS_COPY_ADDRESS**

**Address:** **0x**

**Access:** **Fullword/Doubleword, RW**

**Format:**




**Purpose:** SYS_COPY_ADDRESS defines the source and destination addresses for the DMA copier. The addresses must be set before initiating the copy. The addresses must be on 64-bit boundaries.

**Fields:**

SRC_ADDR: 64-bit aligned address specifying the starting source address for the hardware block copier.

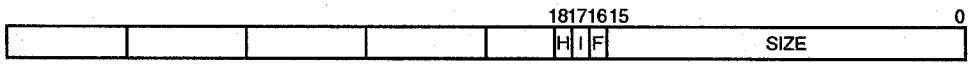
DEST_ADDR: 64-bit address specifying the starting destination address fo the hardware block copier.

**Name:** **SYS_COPY_SIZE**

**Address:** **0x**

**Access:** **Fullword, RW**

**Format:**




**Purpose:** SYS_COPY_SIZE defines the size and mode of the hardware block copy. The copy is initiated when this register is written. Copies are performed from low addresses to high addresses using burst mode accesses. It is the responsibility of software to insure that overlap of source and destination does not occur.

**Fields:**

SIZE: Specifies the number of 64-bit transfers to be made to the destination.

F: If set, the copy is made from the internal transfer registers to the destination. No source accesses are performed. If not set, the copy is made from the source to the destination through the internal transfer registers. Source fetches fill the 64 bytes of internal copy space then write them to the destinaltion.

I: Interrupt upon completion.

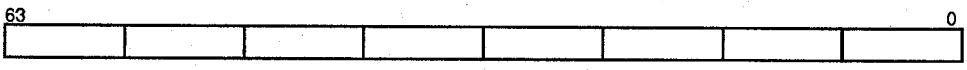
H: High priority copy. If set, the copy memory access priority is set above that of the CPU and BLITTER. If not set, the copy memory access priority is set below that of the CPU.

**Name:** **SYS_COPY_TRANSFER**

**Address:** **0x**

**Access:** **Fullword/Doubleword, W**

**Format:**




**Purpose:** The SYS_COPY_TRANSFER registers are used as holding registers for the system copy function. They may also be loaded from the CPU for fill operations.

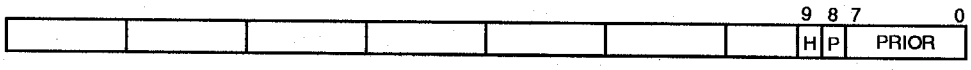
# Appendix A Video Registers

**Name:** **VID_AREA_x**

**Address:** **0x**

**Access:**

**Format:**




**Purpose:** VID_AREA_x, 0 <= x <= 7, contains display control information used when Mapped Video mode is active.

**Fields:**

PRIOR: The priority level of this pixel.

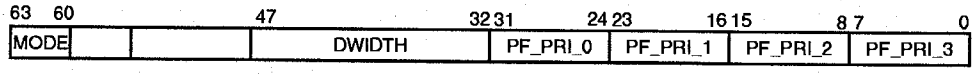
P: Palette bank specifier.

**Name:** **VID_CNTL_MONITOR**

**Address:**

**Access:**

**Format:**




**Purpose:** VID_CNTL_MONITOR contains control information for the monitor. The information written into this register is made active on the next line time.

**Fields:**

LP_EN: Enables lightpen

N_P: NTSC/PAL enable. When active, the signals specific to broadcast style video (equalization, serration, etc.) are enabled. When inactive, these signals are not merged into the video.

I: Interlace. When active, the monitor is considered to be interlaced.

B_EN: Beam enable. When active, the display is enabled.

HSYNC_POS: Horizontal Sync Positive. When active, the horizontal sync signal is high active.

VSYNC_POS: Vertical Sync Positive. When active, the vertical sync signal is high active.

CSYNC_POS: Composite Sync Positive. When active, the composite sync signal is high active.

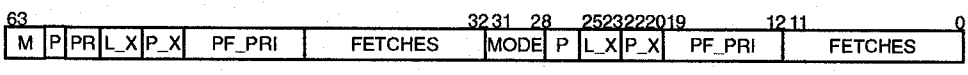
SOG: Sync on Green. When active, the sync signal is inserted into the Green analog output.

**Name:** **VID_CNTL_DISPLAY**

**Address:** **0x**

**Access:** **Fullword, RW**

**Format:**




**Purpose:** VID_CNTL_DISPLAY contains control information for the video display subsystem. Each fullword contains control information for one playfield. Four playfields have been defined. The fullword at address 0x.... contains the control information for playfield 0, address 0x... contains the control information for playfield 1, etc. The information written into this register is made active on the next line time.

**Fields:**

M: Playfield mode:

| Code | Mode |
|---|---|
| 000 | Playfield Disabled |
| 001 | Palette index mode. The data contained in each pixel is used as an index into one of the two color lookup tables. The least significant P bit specifies which lookup table will be used. |
| 010 | HAM8 mode. The data contained in each pixel is used as a HAM8 value. Each 8 bit HAM8 pixel contains a two bit control field and a six bit value field. See the description of HAM8 for further details. Palette 0 is always used for HAM8 base value lookups. The P field is concatenated onto the six bit base to specify one of 4 regions of the palette to be used to interpret the base value. |
| 011 | Direct mode. The pixel value is sent directly to the pixel output. Playfield 0's direct output is connected to the Red D/A; Playfield 1's direct output is connected to the Green D/A; Playfield 2's direct output is connected to the Blue D/A. Playfield 3 is ignored when placed in direct mode. |
| 100 | Combined mode. Playfields 0 and 1 and Playfields 1 and 2 may be combined to form 16-bit pixels. When using this mode, playfield 0 (or 2) should be set to the combined mode and their corresponding other half (1 (or 3)) should be disabled. |

| | |
|---|---|
| 101 | Priority mode. Only playfield 0 may be placed in this mode. When in this mode, playfield 0 contains the priority for playfield 1. Each of playfield 1's pixel priority values is received from playfield 0's data stream. Any other playfield placed in this mode will be disabled. |
| 110 | Mapped mode. |

P: Palette Specifier. Specifies which of the two palettes will be used for this playfield.

PR: Palette Region. This two bit field is used to specify a palette region. This field is only used in HAM8 mode.

L_X: Line Expansion. This field specifies the number of times (minus 1) each line of source data should be displayed. The value 0 indicates that each source line should be displayed for one line time.

P_X: Pixel Expansion. This field specifies the number of times (minus 1) each pixel should be displayed. The value 0 indicates that each source pixel should be displayed for one pixel time.

PF_PRI: The priority level of the play-field. The encoded value in this field is compared to the priority values of other play-fields and sprites which are active at any given pixel time. The highest priority, non-transparent object is made visible. Lower priority numbers have the highest priority. When more than one object has the same priority level, the object with the lower physical number (Play-field 0 is lower than play-field 3) is displayed.

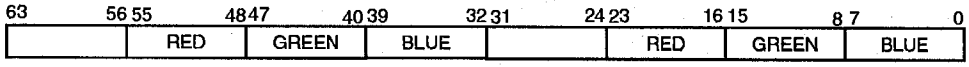
FETCHES: Defines the number of 64-bit fetches to be performed for this playfield for each line of the display.

**Name:** **VID_COLOR_x_y**

**Address:** **0x**

**Access:**

**Format:**




**Purpose:**

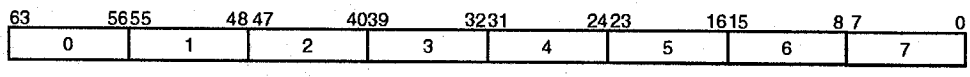
**Fields:**

**Name:** **VID_DATA_x**

**Address:** **0x**

**Access:**

**Format:**




**Purpose:** VID_DATA_x, 0 <= x <= 3, is the destination address used by DMA to fill the video data buffer. The video data buffers operate in a ping-pong fashion: one is being filled by the DMA, while the other, filled during the previous line time is shifted into the video stream.

See "video ping-pong buffering" for an explanation of timing.

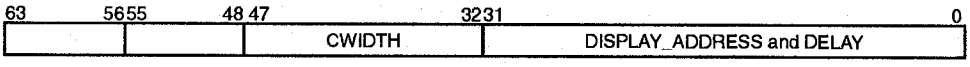
**Fields:** Each byte of the data register contains an 8 bit pixel. (8 pixels per fetch)

**Name:** **VID_PTR_x**

**Address:** **0x**

**Access:**

**Format:**




**Purpose:** VID_PTR_x, 0 <= x <= 3, controls the location of the data to be displayed with the associated video field. The information written into this register is made active on the next line time.

**Fields:** DISPLAY_ADDR: Defines the address of the data associated with the first doubleword for this video field.

This address will be transferred into the current line register at vertical blank time when AUTO mode is enabled. When AUTO mode is not enabled, this value is transferred into the current line register at the next line time.

See "video ping-pong buffering" for an explanation of timing.

The current line register is used to fetch the contents of the first line of video for this video field. Further lines for the video field will be fetched from the address held in the working register, incremented by the WIDTH field.

DELAY: Delay in pixels from the first pixel of this double word to the first pixel to be displayed.

Note that the concatenation of the DISPLAY_ADDRESS and the delay form the byte address of the pixel to be displayed at the left- most position of the first line.

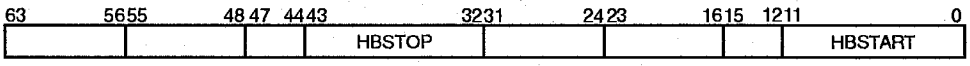
CWIDTH: Defines the width of the canvas in pixels. (must be a multiple of 8). This value will be added to the initial line DISPLAY_ADDRESS to determine the address for the next line access.

**Name:** **VID_HORIZ_BLANK**

**Address:** **0x**

**Access:**

**Format:**




**Purpose:** VID_HORIZ_BLANK controls the position of the HBLANK signal. Display data is shifted when the blanking signal is not active (and vertical blanking is not active). The information written into this register is made active on the next line time.

**Fields:**

HBSTART: Defines the pixel number where the HBLANK signal will become active.

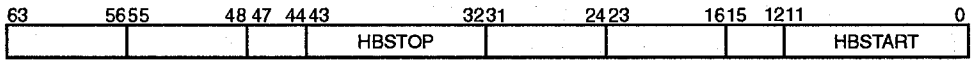
HBSTOP: Defines the pixel number where the HBLANK signal will become inactive.

**Name:** **VID_HORIZ_BURST**

**Address:** **0x**

**Access:**

**Format:**




**Purpose:** VID_HORIZ_BURST controls the position of the vd_color_burst0 signal. This signal is used to control the position of the color-burst in NTSC and PAL display modes. The information written into this register is made active on the next line time.

**Fields:**

HBSTART: Defines the pixel number where the vd_color_burst0 signal will become active.

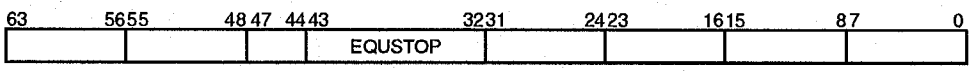
HBSTOP: Defines the pixel number where the vd_color_burst0 signal will become inactive.

**Name:** **VID_HORIZ_EQU1STOP**

**Address:** **0x**

**Access:**

**Format:**




**Purpose:** VID_HORIZ_EQU1STOP controls the position at which the first equalization pulses are stopped. It is assumed that the pulses start at the VID_HORIZ_SYNC.HSSTART position. The information written into this register is made active on the next line time.

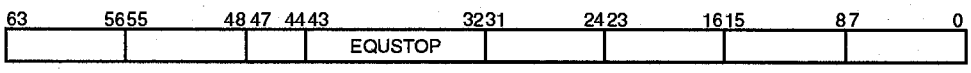
**Fields:** EQUSTOP: Defines the pixel number where EQU1 will become inactive.

**Name:** **VID_HORIZ_EQU2STOP**

**Address:** **0x**

**Access:**

**Format:**




**Purpose:** VID_HORIZ_EQU2STOP controls the position at which the second equalization pulses are stopped. It is assumed that the pulses start at the VID_HORIZ_TOTAL.HCENTER position. The information written into this register is made active on the next line time.

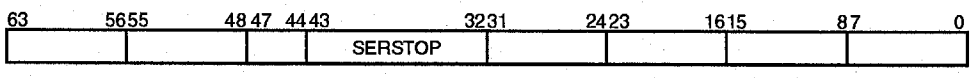
**Fields:** EQUSTOP: Defines the pixel number where EQU2 will become inactive.

**Name:** **VID_HORIZ_SER1STOP**

**Address:** **0x**

**Access:**

**Format:**




**Purpose:** VID_HORIZ_SER1STOP controls the position at which the first serration pulses are stopped. It is assumed that the pulses start at the VID_HORIZ_SYNC.H-SSTART position. The information written into this register is made active on the next line time.

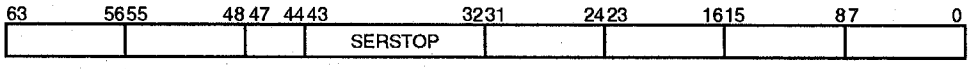
**Fields:** SERSTOP: Defines the pixel number where SER1 will become inactive.

**Name:** **VID_HORIZ_SER2STOP**

**Address:** **0x**

**Access:**

**Format:**




**Purpose:** VID_HORIZ_SER2STOP controls the position at which the second serration pulses are stopped. It is assumed that the pulses start at the VID_HORIZ_TOTAL.HCENTER position. The information written into this register is made active on the next line time.

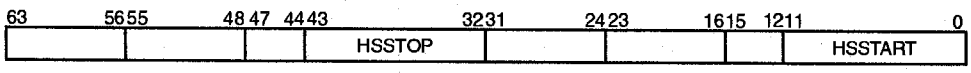
**Fields:** SERSTOP: Defines the pixel number where SER2 will become inactive.

**Name:** **VID_HORIZ_SYNC**

**Address:** **0x**

**Access:**

**Format:**




**Purpose:** VID_HORIZ_SYNC controls the position of the vd_hsync0 signal. The information written into this register is made active on the next line time.

**Fields:**

HSSTART: Defines the pixel number where the vd_hsync0 signal will become active.

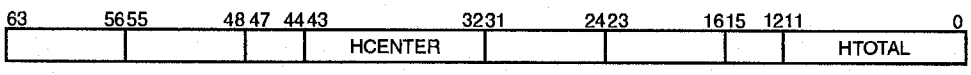
HSSTOP: Defines the pixel number where the vd_hsync0 signal will become inactive.

**Name:** VID_HORIZ_TOTAL

**Address:**

**Access:**

**Format:**




**Purpose:** VID_HORIZ_TOTAL controls the number of pixels in the video line. The information written into this register is made active on the next line time.

**Fields:**

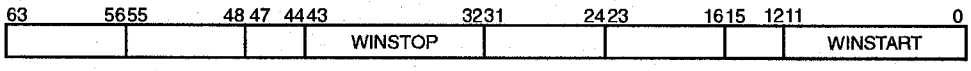
HTOTAL: Defines the number of pixels in the video line. The number of pixels minus 2 that are in a video line is placed in this field.H-CENTER:Defines the center of the horizontal line. The number of pixels minus 2 of the center of a video line is placed in this field.

**Name:** **VID_HORIZ_WINDOW**

**Address:** **0x**

**Access:**

**Format:**




**Purpose:** VID_HORIZ_WINDOW controls the position of the displayed window within the horizontal line. Display data is shifted when the window signal is active (and vertical window is active). The information written into this register is made active on the next line time.

**Fields:**

WINSTART: Defines the pixel number where the HWINDOW signal will become active.

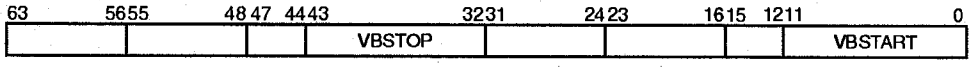
WINSTOP: Defines the pixel number where the HWINDOW signal will become inactive.

**Name:** **VID_VERT_BLANK**

**Address:** **0x**

**Access:**

**Format:**




**Purpose:** VID_VERT_BLANK controls the position of the vd_vblank0 signal. The information written into this register is made active on the next line time.

**Fields:**

VBSTART: Defines the line number where the vd_vblank0 signal will become active.

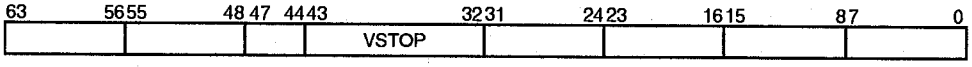
VBSTOP: Defines the line number where the vd_vblank0 signal will become inactive.

**Name:** **VID_VERT_EQU**

**Address:** **0x**

**Access:**

**Format:**




**Purpose:** VID_VERT_EQU controls the position where vertical equalization is terminated. It is assumed that vertical equalization begins at VID_VERT_SYNC.VSSTART. The information written into this register is made active on the next line time.

**Fields:**

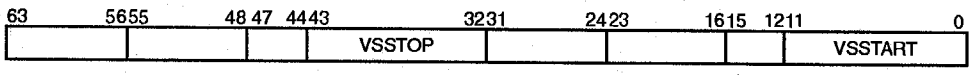
| | |
|---|---|
| VSTOP: | Defines the line number where the verticat equalization will become inactive. |

**Name:** **VID_VERT_SYNC**

**Address:** **0x**

**Access:**

**Format:**




**Purpose:** VID_VERT_SYNC controls the position of the vd_vsync0 signal. The information written into this register is made active on the next line time.

**Fields:**

VSSTART: Defines the line number where the vd_vsync0 signal will become active.

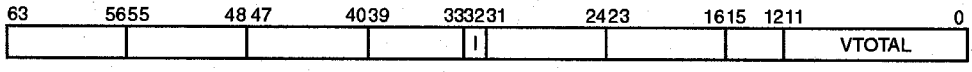
VSSTOP: Defines the line number where the vd_vsync0 signal will become inactive.

**Name:** **VID_VERT_TOTAL**

**Address:** **0x**

**Access:**

**Format:**




**Purpose:** VID_VERT_TOTAL controls the number of lines in the video field. The information written into this register is made active on the next line time.

**Fields:**

VTOTAL: Defines the number of full lines in video field. The number of full lines minus 1 is placed in this register.

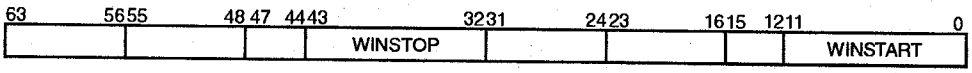
I: 0 for non-interlaced displays. 1 for interlaced displays.

**Name:** **VID_VERT_WINDOW**

**Address:** **0x**

**Access:**

**Format:**




**Purpose:** VID_VERT_WINDOW controls the position of the displayed window within the frame. Display data is shifted when the window signal is active (andl horizontal window is active). The information written into this register is made active on the next line time.

**Fields:**

WINSTART: Defines the line number where the VWINDOW signal will become active.

WINSTOP: Defines the line number where the VWINDOW signal will become inactive.

# Appendix A Video Chip Circuit Description

Figures XXX-YYY contain behavioral level schematics of the Video Chip.

## A.1 Video Horizontal Counter:

In order to achieve the high speeds demanded by high resolution displays, the horizontal counter has been designed as a combination of a traditional count up counter and a 4 bit shift register. This was done to allow the incrementer portion of the counter to be more easily designed using a traditional adder. The least significant two bits of the counter are implemented as a ring counter (shift register). When the cnt0 bit of the ring counter has a 1 in it, the mux for the upper bits selects the incremented value. This gives the incrementer 4 clock times to produce the next value. While the other bits of the ring counter are active, the current value in the upper bits is simply recirculated.

Note that by interleaving the bits (43, 11, 42, 10, ... 34, 2, 33, 1, 32, 0) of the data bus, routing to the match registers is eased.

The reset signal synchronously jams a 0 into the upper counter bits and cnt3, cnt2, and cnt1 of the ring counter, while forcing a 1 into cnt0.

The load signal is used to force particular counts into the register. This is used for testing purposes only.

The horiz_match_reg's are loaded from the data bus and consists of a 14 bit latch and 14 bit comparitor. The upper 10 bits compare the counter portion of the horizontal counter and therefore will only see a change every fourth pixel. The lower 4 bits see the output of the ring counter.

The vid_sync_ff is a synchronous set/clr DFF.

## A.2 Video Vertical Counter

The vertical counter is designed as a traditional count-up counter. It is driven by the pixel clock, so in most cycles will simply be recycled. This counter actually counts half-lines, as defined by the VID_HORIZ_TOTAL.HCENTER point of the horizontal scan.

## A.3 Sprite Buffer:

The sprite buffer consists of two 64 bit by 16 word ram (register) blocks. One is being written while the other is being used to shift pixel data out. Sprites may be expanded in either the horizontal or vertical directions. Both may be selected.

### A.3.1 vid_spr_buffer (0/1):

These are the SRAM blocks. I see these as being custom blocks of static RAM or latches, whichever make most sense. A row is written when the row select is active and the wr line is pulsed. All 64 bits are written at one time. The memory is read when the row_sel is active. All 64 bits are read at the same time. It is guaranteed that we will not be writing and reading a particular block at the same time.

### A.3.2 vid_spr_sel_reg (0/1):

These are 16 bit shift registers. Each bit in the register controls a row_select for the sprite memory block. The register is purely synchronous, although depending on whether the ram is being read or written, the clock will either be the system clock or the pixel clock.

The reset line forces the first bit of the shift register to be set on the next clock. The shift register advances when the en line is active. In the fastest case, the en line will only be active once every eight pixel clocks or once every sys_clock. Since this register is essentially counting the rows of memory read or written, it is also used to determine when the last row has been read during pixel display time. This is accomplished by decoding the width bus and asserting the done signal when the appropriate row select has gone active.

I see this block as being custom logic bolted directly to the side of the memory.

### A.3.3 vid_spr_reg:

This is a 64 bit DFF which holds the pixel data currently being shifted out. It is used only during display operations and either accepts information from one of the two buffers or the data is recirculated. It is clocked using the pixel clock, but new data will only appear at its inputs once every 8 pixel clocks. I see this as being custom logic associated with the mux below.

### A.3.4 vid_spr_mux:

This is a 3:1 mux which determines the inputs for the vid_spr_reg. It selects either one of the sprite memory outputs or to recycle the data in the register. I see this as being custom logic placed between the two memories.

### A.3.5 vid_spr_output_mux:

This is an 8:1 mux. Each of the eight 8-bit pixels is individually selected out of the vid_spr_reg by this mux. The output will be registered on the next pixel clock, so this mux is given a full pipeline delay. The control for the mux is mod-

eled as 8 individual select lines. It is guaranteed by the control logic that only one will be active at any given time so that a simple n transistor mux may be used.

I see this as being custom logic as well.

### A.3.6 vid_spr_pixel_sel_cntl:

This is also a shift register (ring counter). It is eight bits, with each bit providing a bit of the control for the vid_spr_output_mux. On reset, the bit representing the left-most pixel of the vid_spr_reg is set. This is modeled as bit 7. Each time the en line is active, the active bit is moved one bit down the register until it reaches bit 0. The next en time will cause the one to move back to bit 7 to restart. This condition (bit 0 active) also enables the last_pix output.

I see this as being custom logic as well.

### A.3.7 vid_spr_h_expand_cntl:

This is a counter which determines how many times to re-display a specific pixel of a sprite before moving to the next one. It implements the horizontal expand feature of the sprites. Since it must count at pixel clock frequencies and the maximum repeat is only 8, I have modeled this as a shift register (ring counter).

Here, on reset and when sprites are enabled (h_en is high and v_en is HIGH) and the low order bit of the shift register is active, the repeat count is loaded into the register. This is really the decoded repeat count, where only the values of 8, 4, 2, and 1 are permitted. Each pixel clock thereafter (while enabled) shifts the value to the right until bit 0 becomes active at which time the initial value is re-loaded.

When bit 0 is active, this indicates that this is the last pixel time that a particular pixel should be displayed. If there are more pixels in the vid_spr_reg to be displayed (last_pix not active), then the new_pix signal is made active. This is wired to the en of the vid_spr_pixel_sel_cntl and causes the shift register there to select the next pixel from the vid_spr_reg. If the last pixel is currently being displayed (last_pix is active), then a new word from the appropriate memory is loaded into vid_spr_reg. Otherwise, the contents of vid_spr_reg are re-circulated.

This should probably also be custom due to the high speed involved.

### A.3.8 vid_spr_clk_sel:

This is a mux which switches a number of signals as a function of which buffer is currently active.

### A.3.9 vid_spr_v_expand_cntl:

This is a counter which determines how many times to re-display a line before fetching and displaying another. It implements the vertical expand feature of the sprites. It also has DFFs which determine when to request memory fetches for sprite lines and when to switch sprite line buffers.

The ring counter is initialized as a function of the vertical repeat bus for this sprite. It takes on the initial value whenever the vertical sprite enable is inactive or during an active sprite whenever the count has reduced to the last line.

A DFF is used to delay the v_spr_en signal by one line count. This permits the detection of the edge of the v_spr_en. This is used to determine when to request memory for a new sprite line (remember, sprite data is fetched one line before it is needed).

A DFF is used as a toggle to select between the two buffers. The Q and QB outputs are used as the buf[0/1]_act signals.

### A.3.10 vid_spr_en_ff:

This is a DFF which is used to "turn-off" the horizontal sprite enable (h_spr_en) when the appropriate number of pixels has been displayed. This output is gated with h_spr_en. The DFF is set at the beginning of every line and reset when the done[0/1] signal for the appropriate buffer is active along with the last pixel signal.

An alternative approach to all of this may be to build one 64 bit by 32 block of memory. The vid_spr_sel_regs would still exist, but one would be along the top of the memory and the other would be along the bottom... This would eliminate the vid_spr_mux and possibly make the distribution of the incoming data easier.