Section 3. XGA Function

XGA Function Introduction

The XGA function is generated by the type 2 video subsystem.
There are two levels of XGA Function available:

- XGA
- XGA-NI

This chapter describes the capabilities and operation of the XGA function.

The XGA function has three modes:

- VGA
- 132-column text
- Extended Graphics.

VGA Mode

In VGA mode, the XGA video subsystem is VGA register compatible, as defined in the VGA function description.

132-Column Text Mode

In this mode, text is displayed in 132 vertical columns using 200, 350, or 400 scan lines.

Extended Graphics Mode

Extended Graphics mode provides the following software and hardware support.

IBM PS/2 8514/A Adapter Interface Compatibility

Compatibility is provided through the XGA Adapter Interface, a device driver supplied with the subsystem as programming support for applications operating in the disk operating system (DOS) environment.

High Resolution Support

Depending on the display attached and the size of video memory installed, the image on a screen can be defined using 1024 PELs and 768 scan lines with 256 colors.

Direct Color Mode

In this mode, each 16-bit PEL in video memory specifies the color of the PEL directly. This allows 65,536 colors to be displayed using 640 PELs and 480 scan lines.

Packed PEL Format

In the packed PEL format, reads and writes to the video memory access all the data that defines a PEL (or PELs) in a single operation.

Hardware Sprite

The sprite is a 64 x 64 PEL image. When enabled, it overlays the picture that is being displayed. It can be positioned anywhere on the display without affecting the contents of video memory.

Display Identification

Signals from the attached display identify its characteristics. Applications use this information to determine the maximum resolution and whether the display is color or monochrome.

Coprocessor

The coprocessor provides hardware drawing-assist functions throughout real or virtual memory. The following functions can be used with the XGA Adapter Interface:

- PEL-block and bit-block transfers (PxBlt)
- Line drawing
- Area filling
- Logical and arithmetic mixing
- Map masking
- Scissoring
- X and Y axis addressing.

The following table details the functional enhancements to the XGA-NI subsystem:

Function	XGA	XGA-NI
Text Mode Character Attributes	VGA only	VGA or Main Frame Interactive (MFI)
132 Column Text Mode	8 PEL Characters	8 or 9 PEL Characters
Coprocessor Support	1,2,4 and 8 Bits Per PEL	1,2,4,8 and 16 Bits Per PEL
Palette Colors	256 Colors From 256K	256 Colors From 16M
PEL Frequencies	Fixed (Max=45MHz)	Programmable (Max=90MHz)
Example Resolutions: PELS x Lines, Simultaneous Colors, Vertical Refresh Rate, I/NI, (VRAM Size)	640x480, 256, 60Hz, NI, (512KB) 640x480, 64K, 60Hz, NI, (1MB) 1024x768, 16, 43.5Hz, I, (512KB) 1024x768, 256, 43.5Hz, I, (1MB)	640x480, 64K, up to 75Hz, NI 800x600, 256, up to 75Hz, NI 800x600, 64K, up to 60Hz, NI 1024x768, 256, up to 75Hz, NI

Note:

I/NI	Interlaced/Non-Interlaced
KB	1024 Bytes
MB	1048576 Bytes
K	1024

Figure 3-1. XGA-NI Functional Enhancements

XGA Components

The XGA video subsystem components include:

- System bus interface
- Memory and CRT controller
- Coprocessor
- Video memory
- Attribute controller
- Sprite controller
- Alphanumeric (A/N) font and sprite buffer
- Serializer
- Palette
- Video digital-to-analog convertor (DAC).

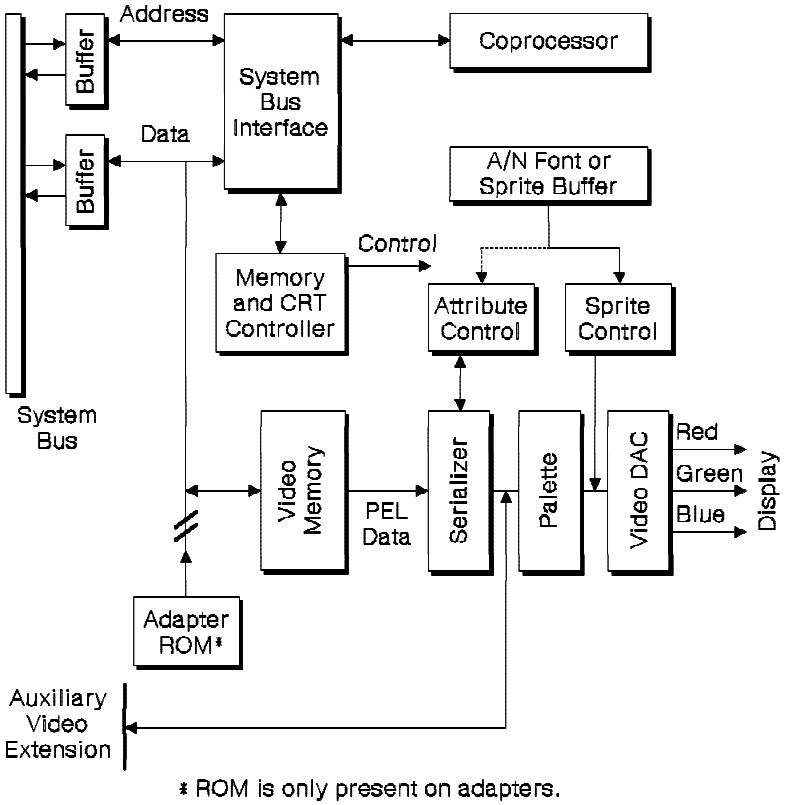


Figure 3-2. XGA Video Subsystem

System Bus Interface

The system bus interface controls the interface between the video subsystem and the system microprocessor. It decodes the addresses for VGA and XGA I/O registers, the memory addresses for the coprocessor memory-mapped registers, and video memory.

It also provides the busmaster function, and determines whether the system data bus is 16 or 32 bits wide.

Memory and CRT Controller

The memory and CRT controller controls access to video memory by the system microprocessor, displays the contents of video memory on the display, and provides support for the VGA and 132-column text modes.

Coprocessor

The coprocessor provides hardware drawing-assist functions. These functions can be performed on graphics data in video memory and system memory.

The coprocessor updates the video memory independently of the system microprocessor. Instructions are written to a set of memory-mapped registers; the coprocessor then executes the drawing function.

The coprocessor functions are:

PEL-Block or Bit-Block Transfers
: Transfers a bit map, or part of a bit map, from one location to another:

- Within video memory
- Within system memory
- Between system and video memory.

Line Drawing
: Draws lines, with a programmable style, into a bit map in video memory or system memory.

Area Fill
: Fills an outlined area in video memory or system memory with a programmable pattern.

Logical and Arithmetic Mixing
Provides logical and arithmetic operations for use with data in video memory or system memory.

Map Masking
Controls updates to each PEL for all drawing functions.

Scissoring
Provides a rectangular-mask function for use instead of the mask map.

X and Y Axis Addressing
Allows a PEL to be specified by its X and Y coordinates within a PEL map, instead of by its linear address in memory.

Video Memory

The video subsystem uses a dual-port video memory to store on-screen data, so that video memory can be read serially to display its contents as the data is being updated.

Attribute Controller

The attribute controller works with the memory and CRT controller to control the color selection and character generation in the 132-column text mode and VGA text modes.

Sprite Controller

The sprite controller is used to display and control the position and image of the sprite (cursor). The sprite is not available in 132-column text mode or VGA modes.

The Serializer, Palette, and Video DAC

The serializer takes data from the serial port of video memory in 16- or 32-bit widths (depending on the size of video memory) and converts it to a serial stream of PEL data. The PEL data addresses a palette location, which contains the color value. The color value is passed to the DAC, which converts the digital information into red, green, and blue analog signals for the display.

Alphanumeric (A/N) Font and Sprite Buffer

This buffer holds the character fonts in 132-column text mode and VGA modes. It also stores the sprite image in Extended Graphics mode.

Compatibility

8514/A Adapter Interface

The XGA function is *not* hardware register compatible with the 8514/A Adapter Interface. Applications written directly to the register-level interface of the 8514/A Adapter Interface do not run.

The XGA function is 8514/A Adapter Interface compatible in the DOS environment through a DOS Adapter Interface driver supplied with the XGA video subsystem.

Applications written to the 8514/A DOS Adapter Interface should run unchanged with the XGA Adapter Interface. The following differences, however, should be noted:

OS/2 protect mode adapter interface
: An XGA Adapter Interface driver is not available for the OS/2 protect mode.

640 x 480, 4 + 4 mode with 512KB display buffer
: This is not an Extended Graphics mode, but applications using this mode and written to the rules for the 8514/A Adapter Interface will run.

Dual-display buffer applications
: 8514/A applications using VGA or other advanced function modes that rely on two separate video display buffers do not run on a single-display configuration. These applications run correctly with two video subsystems (when one is an XGA), and each has a display attached.

Nondisplay memory
: The XGA and 8514/A nondisplay (off-screen) memory are mapped differently. Applications using areas of the off-screen memory for storage may not run.

Adapter interface code size
: The XGA Adapter Interface code size is larger than that for the 8514/A. This reduces the amount of system memory available to applications.

Adapter interface enhancements
: The XGA Adapter Interface is a superset of that provided with the 8514/A. Any 8514/A applications using invalid specifications of parameter blocks may

trigger some of the additional functions provided by the XGA Adapter Interface.

Use of LIM EMS drivers

Applications written to the 8514/A Adapter Interface that locate resources, such as bit maps or font definitions, in LIM EMS memory, and pass addresses of these resources to the adapter interface, require a LIM driver that has implemented the Physical Address Services Interface for busmasters.

Time-dependent applications

Some XGA and 8514/A functions run at different speeds. Applications that rely on a fixed performance may be affected by these differences.

XGA Adapter Interface directory and module name

The directory and module name of the XGA Adapter Interface \XGAPCDOS\XGAAIDOS.SYS is different from that of the 8514/A \HDIPCDOS\HDILOAD.EXE. Applications written to rely on the existence of either the specific 8514/A module name or directory do not run on the XGA Adapter Interface.

8514/A and XGA Adapter Interface code type

The XGA Adapter Interface is implemented as a .SYS device driver. The 8514/A Adapter Interface is implemented as a terminate and stay resident program. Applications written to rely on the adapter interface as a terminate and reside program do not run on the XGA Adapter Interface.

LIM EMS Drivers

The XGA coprocessor memory-mapped registers are located in system memory address space. They reside in the top 1KB of an 8KB block of memory assigned to the XGA subsystem. The lower 7KB of this block is used to address the ROM of an XGA subsystem on an adapter card.

Although an XGA subsystem integrated on the system board does not have a subsystem ROM, an 8KB block of memory is allocated to it to support the coprocessor memory-mapped registers. While the lower 7KB of this 8KB block does not contain any memory, the memory-mapped registers are accessed in the top 1KB of the block.

Applications or drivers, such as LIM EMS drivers that scan memory addresses looking for RAM or ROM signatures, may assume incorrectly that all 8KB of memory is available for use.

The location of the 8KB block of memory assigned to the XGA subsystem can be determined using the System Unit Reference diskette. See the LIM driver installation instructions for details on how to avoid address conflicts.

XGA Applications (Written To The Hardware Interface)

Some XGA applications which have a dependency upon specific monitor IDs or Characteristics may not function on XGA-NI See “XGA Display Mode Query and Set (DMQS)” on page 3-192.

VGA compatibility

The XGA subsystem is register compatible with the VGA, as defined in the VGA function description (see “Effects of VGA & XGA Mode Setting on Video Memory” on page 3-229 for switching between the different XGA subsystem modes).

| In addition to normal VGA text mode character attributes, the
| XGA-NI subsystem has hardware support for Main Frame
| Interactive (MFI) character attributes. See “Main Frame Interactive (MFI) Support” on page 3-17.

132-Column Text Mode

| In this mode the XGA subsystem is capable of displaying 132
| alphanumeric characters on the display. The capabilities of the
| mode are defined below:

| **XGA** Each character is 8 PELs wide. VGA character attributes
| are available.

| **XGA-NI** Each character can be either 8 or 9 PELs wide. VGA or
| Main Frame Interactive (MFI) character attributes are
| available. See “Main Frame Interactive (MFI) Support” on page 3-17.

It is register compatible with the VGA, except for the following VGA CRT controller registers:

Horizontal Total

VGA requires that this register holds a value that is five less than the number of characters on a scan line. In 132-column text mode, this register requires a value that is one less than the number of characters on a scan line.

The End Horizontal Retrace

In 132-column text mode the End Horizontal Retrace field has no effect. Instead, Extended Graphics mode Horizontal Sync Pulse End register (index hex 1A) is used to give a larger horizontal count.

The Horizontal Retrace Delay field has no effect. Instead, Extended Graphics mode Horizontal Sync Pulse Position registers (index hex 1C and 1E) are used.

The End Horizontal Blanking, Bit 5 field continues to be effective.

See “132-Column Text Mode” on page 3-226 for setting 132-column text mode. See “Smooth Scrolling of VGA and 132 Column Text Modes” on page 3-225 for details on achieving smooth scrolling.

Main Frame Interactive (MFI) Support

MFI character attribute support is available on the XGA-NI subsystem. It is available when operating in VGA or 132 Column Text Mode.

To ensure future compatibility, Video BIOS should be used to select or deselect the MFI attribute support.

```
Interrupt 10h - Video BIOS

                (AH) = 12h - Alternate Select
                (BL) = 37h - MFI Alternate Attribute Set

                        (AL) = 1 - Enable MFI Attributes
                        (AL) = 0 - Disable MFI Attributes

                On Return:
                        (AL) = 12h - Function Supported
```

Figure 3-3. MFI Alternate Attribute Select BIOS Call

This function can also be enabled/disabled directly using the following registers:

- “Operating Mode Register (Address 21x0)” on page 3-36
- “MFI Control Register (Index 6D)” on page 3-86

This is not recommended however.

When the XGA-NI subsystem is in a text mode with the MFI attribute function enabled the attribute byte of a character in the display buffer is re-defined from the normal VGA format. This emulates the IBM 3270 and 5250 attributes.

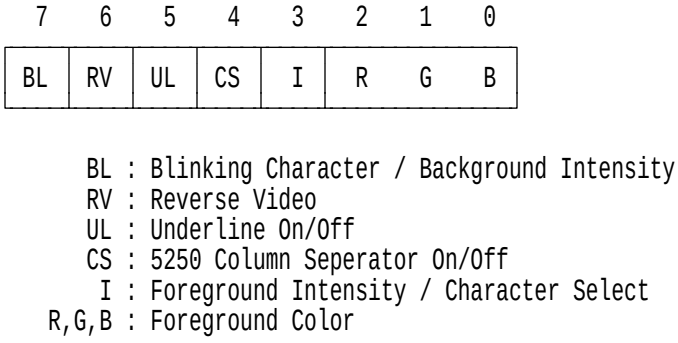


Figure 3-4. MFI Attribute Byte

The fields of the byte are defined as follows:

BL Operates in the same manner as it does in the VGA attribute definition. When Character Blink is enabled (VGA Register:3C0h Index:10h), BL will cause the character to blink at the MFI rate. When Background intensity is selected, BL will select color 8 as the background color instead of the normal color 0. This is used for the trim border feature of some 3270 emulators, for marking areas of text for printing and other purposes.

The MFI Blink rate for cursor and characters is defined below:

Mode	Blink Rate	Duty Cycle
VGA Cursor	VSYNC/16	50% On, 50% Off
VGA Characters	VSYNC/32	50% On, 50% Off
MFI Cursor	Vsync/32	50% On, 50% Off
MFI Characters	VSYNC/64	75% On, 25% Off

Figure 3-5. MFI Blink Rates

RV Reverse Video swaps the foreground and background Colors. When set to 1 Reverse Video is selected, when set to 0 Normal video is selected. When Normal Video

is selected, the Background Color is 0 (or 8 if Background Intensity is set). When Reverse Video is selected, the Foreground Color is 0(or 8 if Background Intensity is set).

UL When set to 1, the underline bar of the character cell is set to the foreground color (visible) as opposed to a decode of the foreground and background colors with VGA attributes. When set to 0, the underline bar of the character cell is disabled.

CS When CS is set to 1, visible Column Separator PELs are displayed in the 1st and last PEL positions of the character cell, on the same character scan line as the underline bar. If UL is set to 1, the Column Separator PELs are the background color, if UL is set to 0, the Column Separator PELs are the foreground Color. That is, the Column Separator PELs are always visible, even if Underline is selected.

When CS is set to 0, the Column Separator PELs are disabled.

I This bit selects between low and high intensity for the specified Foreground Color. When set to 0, the low intensity value is used, when set to 1, the high intensity value is used. It is also used to select between character sets. It operates in the same manner as the VGA equivalent bit. See “Alphanumeric Modes” on page 2-15

R,G,B These bits define the Foreground Color. The operate in the same manner as the VGA foreground color bits. See “Alphanumeric Modes” on page 2-15

When MFI function is enabled, control of the cursor type, blinking and color are controlled using “MFI Control Register (Index 6D)” on page 3-86.

Extended Graphics Mode

Extended Graphics mode provides applications with high-resolution, a wide range of colors, and high-performance. The XGA coprocessor provides hardware assistance in drawing and moving data in video memory and in system memory. Extended Graphics mode is controlled using a bank of 16 I/O registers, and the coprocessor is controlled by a bank of 128 memory-mapped registers.

See “XGA Adapter Identification, Location and XGA Mode Setting” on page 3-192 to locate the subsystem in I/O and memory space.

Display Controller

Video Memory Format

The XGA video memory appears to the system as a byte-addressable, packed array of PELs. The PELs may be 1, 2, 4, 8, or 16 bits long. The first PEL in memory is displayed at the top left corner of the screen. The next PEL is immediately to its right and so on. Addressing is not necessarily contiguous, going from one horizontal line to the next. Addressing depends on the values in the Display PEL Map Width registers. See “CRT Controller” on page 3-23.

Two orders of PELs are supported: Intel and Motorola.

To allow the XGA subsystem to function in either environment, the Memory Access Mode register (for display controller accesses) and the PEL Map n Format register (for coprocessor accesses) are used to make the PELs appear in the required order.

The two formats are described in the following paragraphs.

Intel Order: This table represents the first 3 bytes of the memory map in Intel order and shows the layout of the PELs within those bytes for all PEL sizes (bpp = bits-per-PEL).

PEL	Byte = n + 2								Byte = n + 1								Byte = n + 0							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Size = 1bpp																								
Number	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Bit significance	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Size = 2bpp																								
Number	11	11	10	10	9	9	8	8	7	7	6	6	5	5	4	4	3	3	2	2	1	1	0	0
Bit significance	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
Size = 4bpp																								
Number	5	5	5	5	4	4	4	4	3	3	3	3	2	2	2	2	1	1	1	1	0	0	0	0
Bit significance	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0
Size = 8bpp																								
Number	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
Bit significance	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Size = 16bpp																								
Number	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bit significance	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 3-6. Intel Order of the XGA Memory Map

Motorola Order: This table represents the first 3 bytes of the memory map in Motorola order and shows the layout of the PELs within those bytes for all PEL sizes (bpp = bits-per-PEL).

PEL	Byte = n + 0								Byte = n + 1								Byte = n + 2							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Size = 1bpp																								
Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Bit significance	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Size = 2bpp																								
Number	0	0	1	1	2	2	3	3	4	4	5	5	6	6	7	7	8	8	9	9	10	10	11	11
Bit significance	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
Size = 4bpp																								
Number	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3	4	4	4	4	5	5	5	5
Bit significance	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0
Size = 8bpp																								
Number	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2
Bit significance	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Size = 16bpp																								
Number	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
Bit significance	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8

Figure 3-7. Motorola Order of the XGA Memory Map

PEL Color Mapping

In 1, 2, 4, or 8 bits-per-PEL modes, the palette address is the numerical value of the PEL.

In 16 bits-per-PEL mode (direct color), the color mapping is 5 bits red, 6 bits green, and 5 bits blue. See “Direct Color Mode” on page 3-31.

Border Color Mapping

In the border area of the display, the palette is addressed by the Border Color register (index hex 55). The border area is defined in “CRT Controller” on page 3-23.

Direct Access to Video Memory

An application can use normal memory accesses to read or write PELs in video memory. All bits of one or more PELs can be accessed in a single memory cycle.

System Apertures into Video Memory: The XGA subsystem video memory is accessed in system memory address space through three possible apertures:

4MB Aperture

This allows up to 4MB of video memory to be addressed consecutively. If an access is made at an offset higher than the size of memory installed, no memory is written and undefined values are returned when read.

1MB Aperture

This allows up to 1MB of video memory to be addressed consecutively. If an access is made at an offset higher than the size of memory installed, no memory is written and undefined values are returned when read.

Note: To use the 1MB aperture, the Aperture Index register must be set to 0.

64KB Aperture

This allows up to 64KB of video memory to be addressed consecutively.

The aperture can be located at any 64KB section of the video memory using the Aperture Index register.

See “PS/2 System Video Memory Apertures” on page 3-232 for details on locating and using these apertures.

CRT Controller

The CRT controller generates all timing signals required to drive the serializer and the display. It consists of two counters, one for horizontal parameters and one for vertical parameters, and a series of registers. The counters run continuously, and when the count-value reaches that specified in one of the associated registers, the event controlled by that register occurs.

See “Effects of VGA & XGA Mode Setting on Video Memory” on page 3-229 for mode tables, including CRT controller register values.

CRT Controller Register Interpretations

A representation of the function of each of the CRT controller registers is given in the following figure.

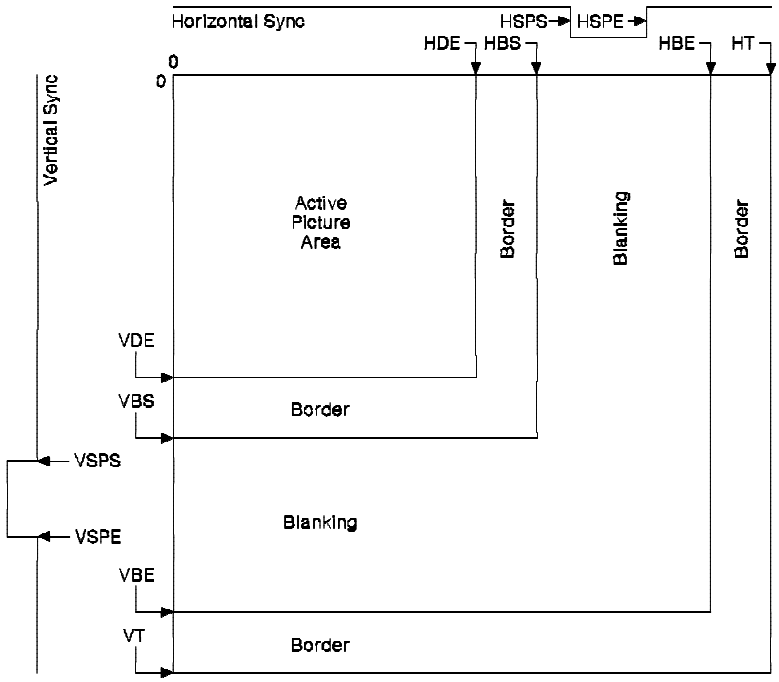


Figure 3-8. CRT Controller Register Definitions

The registers that control a horizontal scan of the display are:

HT Horizontal Total register
HDE Horizontal Display End register
HBS Horizontal Blanking Start register
HBE Horizontal Blanking End register
HSPS Horizontal Sync Pulse Start register
HSPE Horizontal Sync Pulse End register

The registers that control a vertical scan of the display are:

VT Vertical Total register
VDE Vertical Display End register
VBS Vertical Blanking Start register
VBE Vertical Blanking End register
VSPS Vertical Sync Pulse Start register

VSPE Vertical Sync Pulse End register

By using a system interrupt, the XGA subsystem can be programmed to inform the system microprocessor of the start and the end of the active picture area. An enable bit exists for each interrupt in the "Interrupt Enable Register (Address 21x4)" on page 3-39, and the "Interrupt Status Register (Address 21x5)" on page 3-41 contains a status bit for each interrupt.

Scrolling

Some or all of the displayed picture can be made to scroll. The first PEL displayed on the screen is controlled by the Display PEL Map Offset registers. These can be altered to a granularity of 8 bytes, giving coarse horizontal scrolling. Vertical scrolling is achieved by altering the Display PEL Map Offset registers in units of one line length. The line length is stored in the Display PEL Map Width registers. The value stored in the width registers is the amount of memory allocated to each line, not necessarily the physical length of the line being displayed.

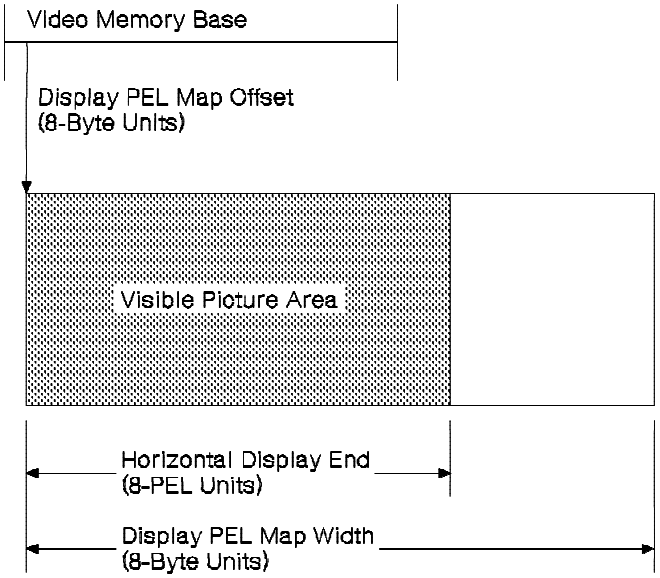


Figure 3-9. Display PEL Map Offset and Width Definitions

The Display PEL Map Width registers must be loaded with a value greater than or equal to the length of line being displayed. The

most efficient use of video memory is achieved when the width value is made equal to the length of the line being displayed. However, it is often more convenient to load a width value that specifies the start of each line on a suitable address boundary.

An area at the bottom of the display can be prevented from scrolling by using the Vertical Line Compare registers (index hex 2C and 2D).

Sprite

The sprite is a 64 x 64-PEL image stored in the XGA subsystem alpha/sprite buffer. When active, the sprite overlays the picture that is displayed. Each PEL in the sprite can take on four values that can be used to achieve the effect of a colored marker of arbitrary shape.

Sprite Color Mapping

The sprite is stored as 2-bit packed PELs, using Intel format, in the sprite buffer. Address zero is at the top left corner of the sprite.

These 2-bit PELs determine the sprite appearance as shown in the following figure:

Bits 1 0	Sprite Effect
0 0	Sprite color 0
0 1	Sprite color 1
1 0	Transparent
1 1	Complement

Figure 3-10. Sprite Appearance Defined by 2-bit PEL

The sprite effect definitions are as follows:

Sprite Colors 0 and 1
: These colors are set by writing to the Sprite Color registers (index hex 38 through 3D).

Transparent
: The underlying PEL color is displayed.

Complement
: The ones complement of the underlying PEL color is displayed.

Sprite Buffer Accesses

The sprite buffer is written to by loading a number into the Sprite Index High and Sprite/Palette Index Low registers. These registers indicate the location of the first group of four sprite PELs to be updated (2 bits-per-PEL implies 4 PELs-per-byte). Then the first four PELs are written to the Sprite Data register. This stores the sprite PELs in the sprite buffer and automatically increments the index registers. Subsequent writes to the Sprite Data register load the remaining sprite PELs, four at a time.

The prefetch function is used to read from the sprite buffer. The index or address of the first sprite buffer location to be read is loaded into the index registers. Writing to either the Sprite Prefetch Index High or the Sprite/Palette Prefetch Index Low registers increments both registers as a single value. The first byte of the index must be written to a non-prefetch index register, and the second byte to the other prefetch index register. For example, write to Sprite Index High, then Sprite/Palette Prefetch Index Low.

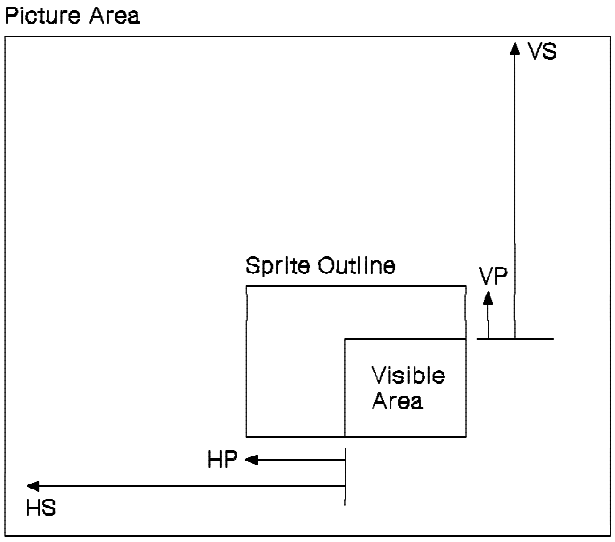
Writing to a prefetch index register loads the sprite data that is stored at the location specified in the index registers into a holding register, then increments the index registers as a single value. Reading the Sprite Data register returns the four sprite PELs that were prefetched, and loads the next four sprite PELs into the holding register. Subsequent reads from the Sprite Data register return the remaining sprite PELs, four at a time.

The sprite and the palette use the same hardware registers during reading and writing, so any task that is updating either the sprite or palette when an interrupt occurs must save and restore the following registers:

- Sprite/Palette Index Low register (index hex 60)
- Sprite Index High register (index hex 61)
- Palette Sequence register (index hex 66)
- Palette Red Prefetch register (index hex 67)
- Palette Green Prefetch register (index hex 68)
- Palette Blue Prefetch register (index hex 69)
- Sprite Prefetch register (index hex 6B).

Note: The Sprite/Palette Prefetch Index Low register (index hex 62) and Sprite Prefetch Index High register (index hex 63) must not be saved and restored.

Sprite Positioning



HS - Horizontal Sprite Start
HP - Horizontal Sprite Preset
VS - Vertical Sprite Start
VP - Vertical Sprite Preset

Figure 3-11. Sprite Positioning

The sprite position is controlled by Start and Preset registers. The Start registers control where the first displayed sprite PEL appears on the screen, and the Preset registers control which sprite PEL is first displayed within the 64 x 64 sprite definition. Using these registers, the sprite can be made to appear at any point in the picture area. If the sprite overlaps any edge, the part of the sprite outside the picture area is not visible (does not wrap). See “Sprite Handling” on page 3-265.

The XGA subsystem can be programmed to inform the system microprocessor when the last line of the sprite has been displayed on each frame using a sprite-display-complete system interrupt. An enable bit exists for each interrupt in the Interrupt Enable register, and the Interrupt Status register contains a status bit for each interrupt. See “Interrupt Enable Register (Address 21x4)” on page 3-39 and “Interrupt Status Register (Address 21x5)” on page 3-41 for the location of the bits.

Palette

The palette has 256 locations and each location contains three fields, one each for red, green, and blue. The palette is used to translate the PEL value into a displayed color.

Before the PEL value is used to address the palette, it is masked by the Palette Mask register. All bits in the PEL corresponding to 0's in the Palette Mask register are forced to 0 before reaching the palette.

Palette Accesses

The Palette Data register is 1 byte wide. Because each palette location is made up of three fields (red, green, and blue) three writes to the Palette Data register are required for each palette location. Palette data is held in a three-field holding register, and the contents are loaded into the palette RAM when all three fields have been filled. The Palette Sequence register controls the Holding Register field (red, green, or blue) selected for access with each write to the Palette Data register.

Two update sequences are possible:

1. Red, green, blue
2. Red, blue, green, no access.

Data is written to the palette by first loading the index, or address, of the first group of three palette-color locations into the non-prefetched Sprite/Palette Index Low register. Because the palette has only 256 locations, the Sprite Index High register is not used. The first color byte is then written to the Palette Data register. This stores the color byte in the Holding Register field indicated by the Palette Sequence register. The Palette Sequence register then increments to point to the next field as determined by the update order.

A second write to the Palette Data register loads the next Holding Register field, and the Palette Sequence register increments again. A third write to the Palette Data register loads the remaining Holding Register field. If update sequence 1 is selected, the palette location is loaded from the holding register and the Palette Sequence register increments again, returning to its starting value. If update sequence 2 is selected, a fourth write to the Palette Data register is necessary before the palette location is loaded. The

no-access data is ignored. Update sequence 2 allows the application to take advantage of the word or doubleword access possible with the XGA subsystem. See “Data Registers (Addresses 21xB to 21xF)” on page 3-47 and “XGA Display Controller Registers” on page 3-34 for more details.

The prefetch function is used to read from the palette. The index or address of the first palette location to be read is loaded into the Sprite/Palette Prefetch Index Low register.

Writing to this register loads the three color fields stored at the location specified in the index register into the palette holding register, then increments the index register.

A subsequent read from the Palette Data register returns the data from the holding register color field, indicated by the Palette Sequence register, and increments the sequence register to point to the next color field. When the last color field, indicated by the Palette Sequence register, is read, the holding register is loaded with the next palette location data, and the index is incremented.

Note: If the subsystem has a monochrome display attached, all of the palette red and blue locations must be loaded with 0's. Alternatively on the XGA-NI subsystem only, the Red and Blue DAC outputs can be blanked using the BRB field of the “Miscellaneous Control Register (Index 6C)” on page 3-85.

The sprite and the palette use the same hardware registers during reading and writing, so any task that is updating either the sprite or palette when an interrupt occurs must save and restore the following registers:

- Sprite/Palette Index Low register (index hex 60)
- Sprite Index High register (index hex 61)
- Palette Sequence register (index hex 66)
- Palette Red Prefetch register (index hex 67)
- Palette Green Prefetch register (index hex 68)
- Palette Blue Prefetch register (index hex 69)
- Sprite Prefetch register (index hex 6B).

Note: The Sprite/Palette Prefetch Index Low register (index hex 62) and Sprite Prefetch Index High register (index hex 63) must not be saved and restored.

Direct Color Mode

In direct color mode the PEL values in the video memory directly specify the displayed color.

The XGA subsystem can display direct color as a 16-bit PEL. The color fields provide the most significant bits of the inputs to the video DACs with the color value.

The bits in the 16-bit direct color data word are allocated to the DAC bits as follows:

Word bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
5R, 6G, 5B	msb		RED		lsb	msb		GREEN			lsb	msb		BLUE		lsb

Figure 3-12. Direct Color Mode Data Word

| On the XGA Subsystem, when selecting this mode, the palette must be loaded with data shown in Figure 3-13 on page 3-32. Only half of the palette should be loaded. Bit 7 of the Border Color register (index hex 55) specifies which half to load. If the Border Color register bit 7 = 0, load the upper half of the palette (locations hex 80 to FF). If the Border Color register bit 7 = 1, load the lower half (locations hex 00 to 7F).

| The values shown in the figure are written to the Palette Data register as a byte value.

| The XGA-NI Subsystem can be initialized in this manner, but there are algorithmic alternatives to loading the palette:

| **Zero Intensity Black** Undefined DAC bits held at 0. This is equivalent to the palette loading method specified above.

| **Full Intensity White** Undefined DAC bits held at 1.

| **Linearized Color** Undefined DAC bits made equal to most significant defined bits.

| **Non-Zero Color** Undefined DAC bits set to 1 if color is non-zero.

| See “Direct Color Control Register (Index 59)” on page 3-79 for control of undefined bits.

| **Note:** This algorithmic method is only available on the XGA-NI subsystem. For applications to function on both levels of

| XGA subsystem, the palette loading method of initializing
| the mode must be used. Alternatively the subsystem level
| can be determined and the appropriate method selected.
| See “XGA Level Identifier” on page 3-205.

Location (Hex)				
Border Color Bit 7 =		Red (Hex)	Green (Hex)	Blue (Hex)
0	1			
80	0	0	0	0
81	1	0	0	8
82	2	0	0	10
83	3	0	0	18
84	4	0	0	20
.	.	.	.	.
.	.	.	.	.
9E	1E	0	0	F0
9F	1F	0	0	F8
A0	20	0	0	0
A1	21	0	0	8
.	.	.	.	.
.	.	.	.	.
BE	3E	0	0	F0
BF	3F	0	0	F8
C0	40	0	0	0
C1	41	0	0	8
.	.	.	.	.
.	.	.	.	.
DE	5E	0	0	F0
DF	5F	0	0	F8
E0	60	0	0	0
E1	61	0	0	8
.	.	.	.	.
.	.	.	.	.
FE	7E	0	0	F0
FF	7F	0	0	F8

| *Figure 3-13. XGA Direct Color Palette Load*

| The values shown in Figure 3-13 have been chosen to ensure
| future compatibility.

| See “XGA Adapter Identification, Location and XGA Mode Setting”
| on page 3-192 and “Direct Color Mode” on page 3-271 for more
| details on this mode.

Coprocessor Functions

Full Coprocessor support in 16 Bits Per PEL mode is available on the XGA-NI subsystem only. The XGA subsystem coprocessor functions do not work in 16 bits-per-PEL mode. However, the coprocessor can function in 8 bits-per-PEL mode while data is being displayed in 16 bits-per-PEL. As a result, the coprocessor can be used to move data (in PxBlts) from one area of memory to another. See “XGA Level Identifier” on page 3-205 to identify different XGA levels.

When displaying in 16 Bits-per-PEL and using the Coprocessor in 8 bits-per-PEL mode, care should be taken when using any of the logical or arithmetic functions since each operation is performed on only 1 byte of data at a time, not the full 16-bit PEL.

If the coprocessor is used to move data into the Video Display Buffer in 8 bits-per-PEL format while displaying in 16 bits-per-PEL mode, the width of the destination map must be doubled.

See “Direct Color Mode” on page 3-271 for more information.

XGA Display Controller Registers

The display controller registers occupy 16 I/O addresses. The addresses are hex 21x0 through 21xF. The x is the Instance as defined in Figure 3-173 on page 3-173. “XGA Adapter Identification, Location and XGA Mode Setting” on page 3-192 provides details of locating and using these registers.

An indexed addressing scheme is used to select additional registers. The index of the registers is written to hex 21xA; the data can then be accessed using hex 21xB through 21xF. Because there are multiple addresses for the data port, writes to a single register are achieved in a single 16-bit instruction, the low byte containing the address, and the high byte the data. Registers that need to be accessed repeatedly (sprite data, palette data, and coprocessor save/restore data) are accessed by setting the index correctly, then performing string I/O instructions, either 2 or 4 bytes at a time. See “Data Registers (Addresses 21xB to 21xF)” on page 3-47.

The 16 I/O addresses are assigned as shown in the following figure.

Address (hex)	Function	Page Reference
21x0	Operating Mode register	3-36
21x1	Aperture Control register	3-38
21x2	Reserved	
21x3	Reserved	
21x4	Interrupt Enable register	3-39
21x5	Interrupt Status register	3-41
21x6	Virtual Memory Control register	3-42
21x7	Virtual Memory Interrupt Status register	3-42
21x8	Aperture Index register	3-43
21x9	Memory Access mode	3-44
21xA	Index	3-45
21xB	Data	3-47
21xC	Data	3-47
21xD	Data	3-47
21xE	Data	3-47
21xF	Data	3-47

Figure 3-14. Display Controller Register Addresses

Register Usage Guidelines

Unless specified otherwise, the following are guidelines when using the display controller registers:

- All registers are 8 bits wide.
- Registers can be read and written at the same address or index.
- When registers are read, they return the last written data for all implemented bits.
- Registers are *not* initialized by reset.
- Special reserved register bits must be used as follows:
  - Register bits marked with '– ' must be set to 0. These bits are undefined when read and should be masked off if the contents of the register is to be tested.
  - Register bits marked with '#' are reserved and the state of these bits must be preserved. When writing the register, read the register first and change only the bits that must be changed.
- Unspecified registers or registers marked as reserved in the XGA I/O address space are reserved. They must not be written to or read from.
- During a read, the values returned from write-only registers are reserved and unspecified.
- The contents of read-only registers must not be modified.
- Counters must not be relied upon to wrap from the high value to the low value.
- Register fields defined with valid ranges must not be loaded with a value outside the specified range.
- Register field values defined as reserved must not be written.
- The function that all XGA subsystem registers imply is only operative in XGA subsystem modes, even though the registers themselves are still readable and writable in VGA modes.

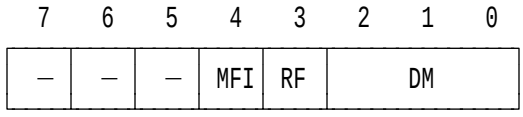
  Writing to the XGA subsystem registers when in VGA mode may cause the VGA registers to be corrupted.

Direct Access I/O Registers

The following registers are directly addressable in the I/O space hex 21x0 through 21xF.

Operating Mode Register (Address 21x0)

This read/write register has an address of hex 21x0.



– : Set to 0, Undefined on Read
MFI : Enable MFI Function
RF : Coprocessor Register Interface Format
DM : Display Mode

Figure 3-15. Operating Mode Register, Address Hex 21x0

The register fields are defined as follows:

MFI This bit is only available on the XGA-NI subsystem. On the XGA subsystem it should only be Set to 0 and assumed to be undefined on a read. When the Enable MFI Function field (bit 4) is set to 1 the display of MFI or VGA character attributes is controlled using “MFI Control Register (Index 6D).” When set to 0 this bit forces VGA character attributes to be displayed. See “Main Frame Interactive (MFI) Support” on page 3-17 for details on MFI attribute Byte.

This bit must be set before “MFI Control Register (Index 6D)” can be accessed.

RF The Coprocessor Register Interface Format field (bit 3) selects whether the coprocessor registers are arranged in Intel or Motorola format. When set to 0, Intel format is selected. When set to 1, Motorola format is selected. See “Coprocessor Registers” on page 3-132.

DM The Display Mode field (bits 2– 0) selects between the display modes available. Both VGA and 132-column text modes respond to VGA I/O and memory addresses. When the XGA subsystem is in either of these modes,

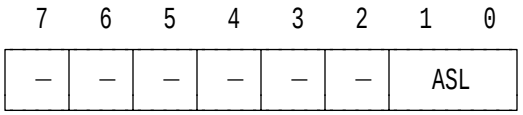
the addressing of the I/O registers and the video memory can be inhibited.

DM Field (binary)	Display Mode
0 0 0	VGA Mode (address decode disabled)
0 0 1	VGA Mode (address decode enabled)
0 1 0	132-Column Text Mode (address decode disabled)
0 1 1	132-Column Text Mode (address decode enabled)
1 0 0	Extended Graphics Mode
1 0 1	Reserved
1 1 0	Reserved
1 1 1	Reserved

Figure 3-16. Display Mode Bit Assignments

Aperture Control Register (Address 21x1)

This read/write register has address of hex 21x1.



– : Set to 0, Undefined on Read
ASL : Aperture Size And Location

Figure 3-17. Aperture Control Register, Address Hex 21x1

The register fields are defined as follows:

ASL The Aperture Size and Location field (bits 1, 0) controls a 64KB aperture through which XGA memory can be accessed in system address space. This aperture gives real mode applications and operating systems a means of accessing the XGA video memory. The 64KB area of the XGA video memory accessed by this aperture is selected using the Aperture Index register. By varying the value of the index register, the 64KB aperture is used to access the entire memory contents of the subsystem.

The aperture is controlled as follows:

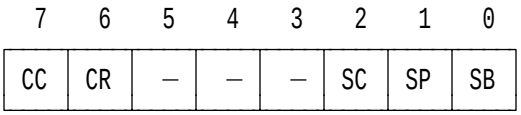
ASL Field (binary)	Aperture Size and Location
0 0	No 64KB Aperture
0 1	64KB at Address Hex 000A0000
1 0	64KB at Address Hex 000B0000
1 1	Reserved

Figure 3-18. Aperture Size and Location Bit Assignments

The 64KB aperture and a 1MB aperture cannot be used together because they are both paged using the Aperture Index register. See “System Apertures into Video Memory” on page 3-23.

Interrupt Enable Register (Address 21x4)

This read/write register has an address of hex 21x4.




```
 – : Set to 0, Undefined on Read
CC : Coprocessor Operation Complete Enable
CR : Coprocessor Access Rejected Enable
SC : Sprite Display Complete Enable
SP : Start Of Picture (End Of Blanking) Enable
SB : Start Of Blanking (End Of Picture) Enable
```

Figure 3-19. Interrupt Enable Register, Address Hex 21x4

The register fields are defined as follows:

CC The Coprocessor Operation Complete Enable field (bit 7) enables and disables the Coprocessor Operation Complete interrupt condition that can be generated by the subsystem. When set to 1, the interrupt is enabled. When set to 0, the interrupt is disabled. The status of the bit in this field has no effect on the interrupt status bits as defined in the Interrupt Status register, but prevents the interrupt condition from causing a system interrupt.

CR The Coprocessor Access Rejected Enable field (bit 6) enables and disables the Coprocessor Access Rejected interrupt condition that can be generated by the subsystem. When set to 1, the interrupt is enabled. When set to 0, the interrupt is disabled. The status of the bit in this field has no effect on the interrupt status bits as defined in the Interrupt Status register, but prevents the interrupt condition from causing a system interrupt.

SC The Sprite Display Complete Enable field (bit 2) enables and disables the Sprite Display Complete interrupt condition that can be generated by the subsystem. When set to 1, the interrupt is enabled. When set to 0, the interrupt is disabled. The status of the bit in this

field has no effect on the interrupt status bits as defined in the Interrupt Status register, but prevents the interrupt condition from causing a system interrupt.

SP The Start of Picture (End of Blanking) Enable field (bit 1) enables and disables the Start of Picture interrupt condition that can be generated by the subsystem. When set to 1, the interrupt is enabled. When set to 0, the interrupt is disabled. The status of the bit in this field has no effect on the interrupt status bits as defined in the Interrupt Status register, but prevents the interrupt condition from causing a system interrupt.

SB The Start of Blanking (End of Picture) Enable field (bit 0) enables and disables the Start of Blanking interrupt condition that can be generated by the subsystem. When set to 1, the interrupt is enabled. When set to 0, the interrupt is disabled. The status of the bit in this field has no effect on the interrupt status bits as defined in the Interrupt Status register, but prevents the interrupt condition from causing a system interrupt.

Interrupt Status Register (Address 21x5)

This read/write register has an address of hex 21x5.

7	6	5	4	3	2	1	0
CC	CR	–	–	–	SC	SP	SB

```
 – : Set to 0, Undefined on Read
CC : Coprocessor Operation Complete Status
CR : Coprocessor Access Rejected Status
SC : Sprite Display Complete Status
SP : Start Of Picture (End Of Blanking) Status
SB : Start Of Blanking (End Of Picture) Status
```

Figure 3-20. Interrupt Status Register, Address Hex 21x5

The register fields are defined as follows:

CC The Coprocessor Operation Complete Status field (bit 7) contains the interrupt status bit that can be generated by the subsystem to reset the Coprocessor Operation Complete interrupt. When read, 1 indicates that the interrupt condition has occurred, and 0 that it has not. Writing a 1 to the bit clears the interrupt condition, while writing a 0 has no effect. See “Programmer's View” on page 3-93 for more information.

CR The Coprocessor Access Rejected Status field (bit 6) contains the interrupt status bit that can be generated by the subsystem to reset the Coprocessor Access Rejected interrupt. When read, 1 indicates that the interrupt condition has occurred, and 0 that it has not. Writing a 1 to the bit clears the interrupt condition, while writing a 0 has no effect. See “Accesses to the Coprocessor During an Operation” on page 3-129 for more information.

SC The Sprite Display Complete Status field (bit 2) contains the interrupt status bit that can be generated by the subsystem to reset the Sprite Display Complete interrupt. When read, 1 indicates that the interrupt condition has occurred, and 0 that it has not. Writing a 1 to the bit clears the interrupt condition, while writing a

0 has no effect. See “Sprite” on page 3-26 for more information.

SP The Start of Picture (End of Blanking) Status field (bit 1) contains the interrupt status bit that can be generated by the subsystem to reset the Start of Picture interrupt. When read, 1 indicates that the interrupt condition has occurred, and 0 that it has not. Writing a 1 to the bit clears the interrupt condition, while writing a 0 has no effect. See “CRT Controller” on page 3-23 (End of blanking) for more information.

SB The Start of Blanking (End of Picture) Status field (bit 0) contains the interrupt status bit that can be generated by the subsystem to reset the Start of Blanking interrupt. When read, 1 indicates that the interrupt condition has occurred, and 0 that it has not. Writing a 1 to the bit clears the interrupt condition, while writing a 0 has no effect. See “CRT Controller” on page 3-23 (Start of blanking) for more information.

Virtual Memory Control Register (Address 21x6)

This read/write register has an address of hex 21x6. Full details of this register are in “Virtual Memory Control Register (I/O Address 21x6)” on page 3-188.

Virtual Memory Interrupt Status Register (Address 21x7)

This read/write register has an address of hex 21x7. Full details of this register are in “Virtual Memory Interrupt Status Register (I/O Address 21x7)” on page 3-190.

Aperture Index Register (Address 21x8)

This read/write register has an address of hex 21x8.

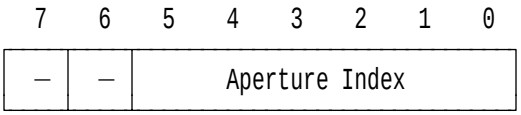


Figure 3-21. Aperture Index Register, Address Hex 21x8

The register field is defined as follows:

Aperture Index

The Aperture Index field (bits 5– 0) provides address bits to video memory when the aperture in the system address space being used is smaller than the size of video memory installed. They are used to move both the 64KB aperture and the 1MB aperture. All 6 bits are used to move the 64KB aperture in the video memory, with a granularity of 64KB. When moving the 1MB aperture, the granularity is restricted to 1MB and only bits 5 and 4 are used. In this case, the lower order bits must be written with 0's.

See “System Apertures into Video Memory” on page 3-23 for details on the use of video memory apertures.

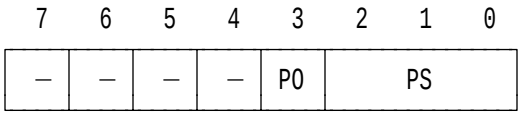
The bits are used as follows:

Aperture Size	Index Bits Used
64KB	5– 0
1MB	5– 4

Figure 3-22. Aperture Index Bit Assignments

Memory Access Mode Register (Address 21x9)

This read/write register has an address of hex 21x9.



– : Set to 0, Undefined on Read
PO : PEL Order
PS : PEL Size

Figure 3-23. Memory Access Mode Register, Address Hex 21x9

The register fields are defined as follows:

PO The PEL Order field (bit 3) controls PEL ordering when the video memory is being accessed by the system (not the coprocessor). Intel or Motorola order can be selected. When set to 0, Intel format is selected. When set to 1, Motorola format is selected.

PS The PEL Size field (bits 2– 0) selects the PEL size. The PEL size must be selected because this register is controlling a PEL swapper that converts from the external format specified to the internal format used by the adapter when the PELs are written, and converts back when they are read.

It is important to set this register correctly when accessing video memory with the system processor.

PEL size values are assigned as follows:

PS Field (binary)	PEL Size
0 0 0	1 Bit
0 0 1	2 Bits
0 1 0	4 Bits
0 1 1	8 Bits
1 0 0	16 Bits
1 0 1	Reserved
1 1 0	Reserved
1 1 1	Reserved

Figure 3-24. PEL Size Bit Assignments

Index Register (Address 21xA)

This read/write register has an address of hex 21xA.

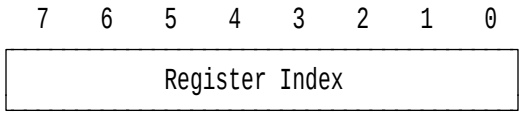


Figure 3-25. Index Register, Address Hex 21xA

The Register Index register selects the indexed Extended Graphics mode register accessed when any address with index hex B through F is read or written. Index values are assigned as shown in the following figure.

Register Index Field (hex)	Register
04	Auto-Configuration
0C	Coprocessor Save/Restore Data A
0D	Coprocessor Save/Restore Data B
10	Horizontal Total Low
11	Horizontal Total High
12	Horizontal Display End Low
13	Horizontal Display End High
14	Horizontal Blanking Start Low
15	Horizontal Blanking Start High
16	Horizontal Blanking End Low
17	Horizontal Blanking End High
18	Horizontal Sync Pulse Start Low
19	Horizontal Sync Pulse Start High
1A	Horizontal Sync Pulse End Low
1B	Horizontal Sync Pulse End High
1C	Horizontal Sync Position
1E	Horizontal Sync Position
20	Vertical Total Low
21	Vertical Total High
22	Vertical Display End Low
23	Vertical Display End High
24	Vertical Blanking Start Low
25	Vertical Blanking Start High
26	Vertical Blanking End Low
27	Vertical Blanking End High
28	Vertical Sync Pulse Start Low
29	Vertical Sync Pulse Start High
2A	Vertical Sync Pulse End
2C	Vertical Line Compare Low
2D	Vertical Line Compare High

Note: Undefined index values are reserved.

Figure 3-26. XGA Index Register Assignments (Part I)

Register Index Field (hex)	Register
30	Sprite Horizontal Start Low
31	Sprite Horizontal Start High
32	Sprite Horizontal Preset
33	Sprite Vertical Start Low
34	Sprite Vertical Start High
35	Sprite Vertical Preset
36	Sprite Control
38	Sprite Color 0 Red
39	Sprite Color 0 Green
3A	Sprite Color 0 Blue
3B	Sprite Color 1 Red
3C	Sprite Color 1 Green
3D	Sprite Color 1 Blue
40	Display PEL Map Offset Low
41	Display PEL Map Offset Middle
42	Display PEL Map Offset High
43	Display PEL Map Width Low
44	Display PEL Map Width High
50	Display Control 1
51	Display Control 2
52	Display ID and Comparator
54	Clock Frequency Select 1
55	Border Color
58	Programmable PEL Clock Frequency
59	Direct Color Control
60	Sprite/Palette Index Low
61	Sprite Index High
62	Sprite/Palette Prefetch Index Low
63	Sprite Prefetch Index High
64	Palette Mask
65	Palette Data
66	Palette Sequence
67	Palette Red Prefetch
68	Palette Green Prefetch
69	Palette Blue Prefetch
6A	Sprite Data
6B	Sprite Prefetch
6C	Miscellaneous Control
6D	MFI Control
70	Clock Frequency Select 2

Note: Undefined index values are reserved.

Figure 3-27. XGA Index Register Assignments (Part II)

Data Registers (Addresses 21xB to 21xF)

These read/write data registers have addresses of hex 21xB to 21xF. The data registers are used when reading and writing to the register indexed by the Index register (Address 21xA). The read/write operation can be of byte, word, or doubleword size.

To perform a byte write to an indexed register, a single 16-bit cycle to address hex 21xA can be used with the index in the lower byte and the data to be written in the upper byte. For indexed registers requiring successive writes, the index can be loaded using a byte write to address hex 21xA, followed by either a word or a doubleword access to address hex 21xC. Only the byte-wide register selected by the index is updated. Word or doubleword accesses result in two or four byte-wide accesses to the same indexed register.

Indexed Access I/O Registers

See “Index Register (Address 21xA)” on page 3-45 for a figure of the indexed registers.

Auto-Configuration Register (Index 04)

This read-only register has an index of hex 04. *Do not write to* this register.

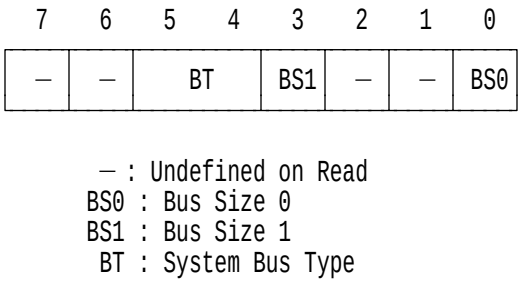


Figure 3-28. Auto-Configuration Register, Index Hex 04

The register field is defined as follows:

BS0 and BS1 The Bus Size field (bits 0 and 3) indicate whether the subsystem is interfaced to an 8-bit, a 16-bit or a 32-bit system, as defined in the following table:

BS1 Field	BS0 Field	System Interface Size
0	0	16 Bits
0	1	32 Bits
1	0	8 Bits
1	1	Reserved

Figure 3-29. System Interface Bus Size

The System Bus Type field (bits 5 and 4) indicates the bus type to which the subsystem is attached.

BT Field (binary)	System Bus Type
00	Micro Channel
01	ISA (AT Bus)
10	Reserved
11	Reserved

Figure 3-30. System Bus Type

Coprocessor Save/Restore Data Registers (Index 0C and 0D)

These read/write registers have indexes of hex 0C and 0D. The registers are an image of a port in the coprocessor. See “Coprocessor State Save/Restore” on page 3-129 for a description of their use.

Horizontal Total Registers (Index 10 and 11)

These read/write registers have indexes of hex 10 and 11.

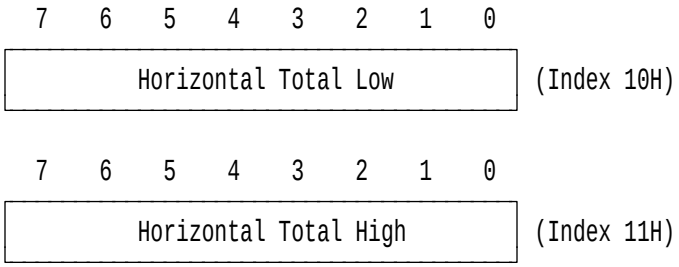


Figure 3-31. Horizontal Total Registers, Indexes Hex 10 and 11

The Horizontal Total Low and Horizontal Total High registers (bits 7– 0) define the total length of a scan line in units of eight PELs. They *must* be loaded as a 16-bit value in the range hex 0000 to 00FF. Values are assigned as shown in the following figure.

Value (hex)	Horizontal Total (PELs)
0000	8
0001	16
0002	24
.	
.	
00FF	2048

Figure 3-32. Horizontal Total Registers Value Assignments

Horizontal Display End Registers (Index 12 and 13)

These read/write registers have indexes of hex 12 and 13.

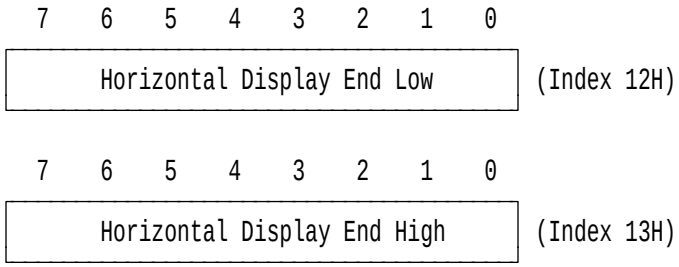


Figure 3-33. Horizontal Display End Registers, Indexes Hex 12 and 13

The Horizontal Display End Low and Horizontal Display End High registers (bits 7– 0) define the position of the end of the active picture area relative to the start of the active picture area in units of eight PELs. They *must* be loaded as a 16-bit value in the range hex 0000 to 00FF. Values are assigned as shown in the following figure.

Value (hex)	Display End (PELs)
0000	8
0001	16
0002	24
.	
.	
00FF	2048

Figure 3-34. Horizontal Display End Registers Value Assignments

Horizontal Blanking Start Registers (Index 14 and 15)

These read/write registers have indexes of hex 14 and 15.

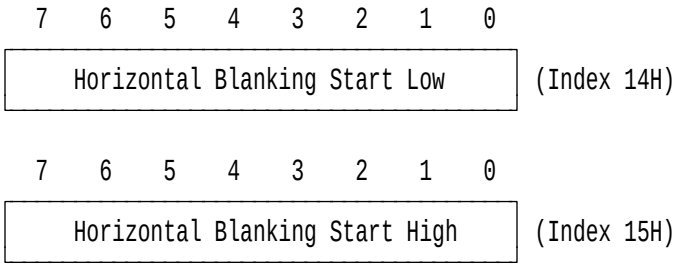


Figure 3-35. Horizontal Blanking Start Registers, Indexes Hex 14 and 15

The Horizontal Blanking Start Low and Horizontal Blanking Start High registers (bits 7– 0) define the position of the end of the picture border area relative to the start of the active picture area in units of eight PELs. They *must* be loaded as a 16-bit value in the range hex 0000 to 00FF. Values are assigned as shown in the following figure.

Value (hex)	Blanking Start (PELs)
0000	8
0001	16
0002	24
.	
.	
00FF	2048

Figure 3-36. Horizontal Blanking Start Registers Value Assignments

Horizontal Blanking End Registers (Index 16 and 17)

These read/write registers have indexes of hex 16 and 17.

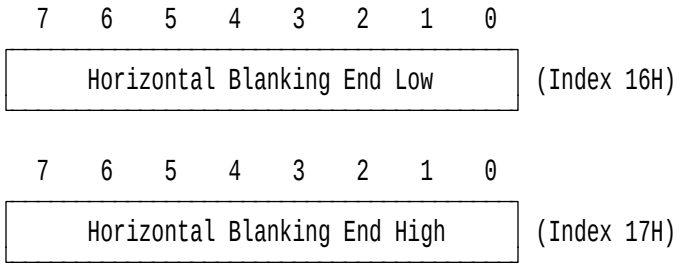


Figure 3-37. Horizontal Blanking End Registers, Indexes Hex 16 and 17

The Horizontal Blanking End Low and Horizontal Blanking End High registers (bits 7– 0) define the position of the start of the picture border area relative to (after) the start of the active picture area in units of eight PELs. They *must* be loaded as a 16-bit value in the range hex 0000 to 00FF. Values are assigned as shown in the following figure.

Value (hex)	Blanking End (PELs)
0000	8
0001	16
0002	24
.	
.	
00FF	2048

Figure 3-38. Horizontal Blanking End Registers Value Assignments

Horizontal Sync Pulse Start Registers (Index 18 and 19)

These read/write registers have indexes of hex 18 and 19.

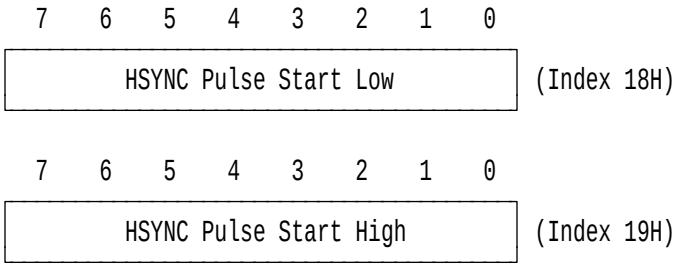


Figure 3-39. Horizontal Sync Pulse Start Registers, Indexes Hex 18 and 19

The Horizontal Sync Pulse Start Low and Horizontal Sync Pulse Start High registers (bits 7– 0) define the position of the start of horizontal sync pulse relative to the start of the active picture area in units of eight PELs. They *must* be loaded as a 16 bit-value in the range hex 0000 to 00FF. Values are assigned as shown in the following figure.

Value (hex)	Horizontal Pulse Start (PELs)
0000	8
0001	16
0002	24
.	
.	
00FF	2048

Figure 3-40. Horizontal Sync Pulse Start Registers Value Assignments

Horizontal Sync Pulse End Registers (Index 1A and 1B)

These read/write registers have indexes of hex 1A and 1B.

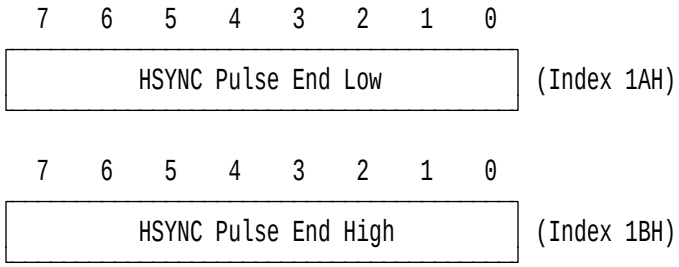


Figure 3-41. Horizontal Sync Pulse End Registers, Indexes Hex 1A and 1B

The Horizontal Sync Pulse End Low and Horizontal Sync Pulse End High registers (bits 7– 0) define the position of the end of the horizontal sync pulse relative to the start of the active picture area in units of eight PELs. They *must* be loaded as a 16-bit value in the range hex 0000 to 00FF.

This XGA subsystem register is also used in 132-column text mode in place of the VGA End Horizontal Retrace register. In 132-column text mode, each eight-PEL unit is equivalent to one eight-PEL character. Values are assigned as shown in the following figure.

Value (hex)	Horizontal Sync Pulse End (PELs)
0000	8
0001	16
0002	24
.	
.	
00FF	2048

Figure 3-42. Horizontal Sync Pulse End Registers Value Assignments

Horizontal Sync Pulse Position Registers (Index 1C and 1E)

These write-only registers have indexes of hex 1C and 1E.

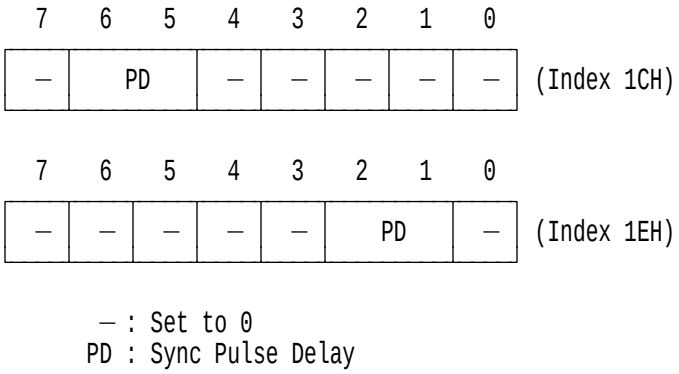


Figure 3-43. Horizontal Sync Pulse Position Registers (Index 1C and 1E)

The register field is defined as follows:

PD The Sync Pulse Delay field (bits 6, 5 or bits 2, 1) allows the 'horizontal sync' (HSYNC) signal to be delayed by up to four PELs. The same value *must* be written to both registers, as shown in the following figure.

PD Field (binary)	Sync Pulse Delay in PELs
0 0	0
0 1	Reserved
1 0	4
1 1	Reserved

Figure 3-44. Horizontal Sync Pulse Delay Bit Assignments

These XGA subsystem registers are also used in 132-column text mode in place of the HRD field in the VGA End Horizontal Retrace register.

Vertical Total Registers (Index 20 and 21)

These read/write registers have indexes of hex 20 and 21.

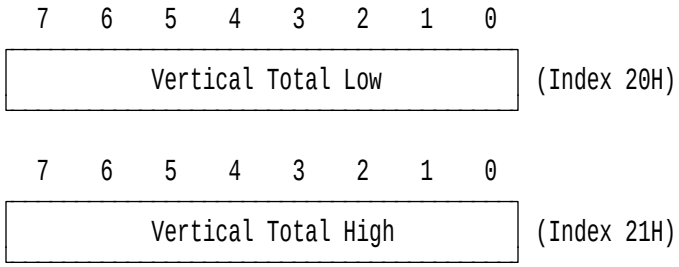


Figure 3-45. Vertical Total Registers, Indexes Hex 20 and 21

The Vertical Total Low and Vertical Total High registers (bits 7– 0) define the total length of a frame in units of one scan line. They *must* be written as a 16-bit value in the range hex 0000 to 07FF. Values are assigned as shown in the following figure.

Value (hex)	Total Length (Scan Lines)
0000	1
0001	2
0002	3
.	
.	
07FF	2048

Figure 3-46. Vertical Total Registers Value Assignments

Vertical Display End Registers (Index 22 and 23)

These read/write registers have indexes of hex 22 and 23.

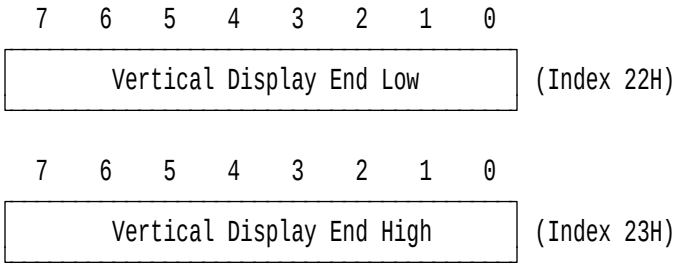


Figure 3-47. Vertical Display End Registers, Indexes Hex 22 and 23

The Vertical Display End Low and Vertical Display End High registers (bits 7– 0) define the position of the end of the active picture area relative to the start of the active picture area in units of one scan line. They *must* be written as a 16-bit value in the range hex 0000 to 07FF. Values are assigned as shown in the following figure.

Value (hex)	Display End (Scan Lines)
0000	1
0001	2
0002	3
.	
.	
07FF	2048

Figure 3-48. Vertical Display End Registers Value Assignments

Vertical Blanking Start Registers (Index 24 and 25)

These read/write registers have indexes of hex 24 and 25.

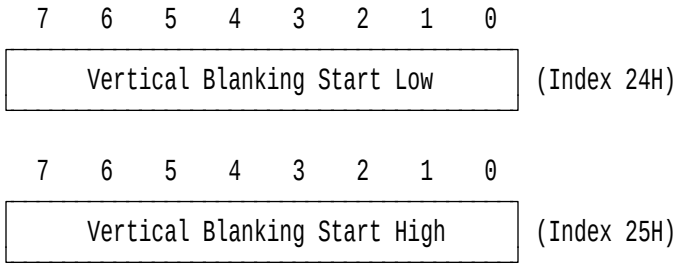


Figure 3-49. Vertical Blanking Start Registers, Indexes Hex 24 and 25

The Vertical Blanking Start Low and Vertical Blanking Start High registers (bits 7– 0) define the position of the end of the picture border area relative to the start of the active picture area in units of one scan line. They *must* be loaded as a 16-bit value in the range hex 0000 to 07FF. Values are assigned as shown in the following figure.

Value (hex)	Border End (Scan Lines) Blanking Start
0000	1
0001	2
0002	3
.	
.	
07FF	2048

Figure 3-50. Vertical Blanking Start Registers Value Assignments

Vertical Blanking End Registers (Index 26 and 27)

These read/write registers have indexes of hex 26 and 27.

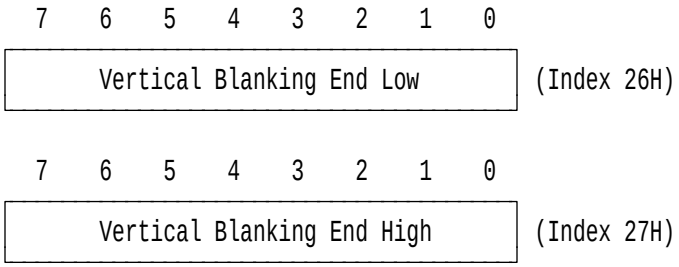


Figure 3-51. Vertical Blanking End Registers, Indexes Hex 26 and 27

The Vertical Blanking End Low and Vertical Blanking End High registers (bits 7– 0) define the position of the start of the picture border area relative to the start of the active picture area in units of one scan line. They *must* be loaded as a 16-bit value in the range hex 0000 to 07FF. Values are assigned as shown in the following figure.

Value (hex)	Border Start (Scan Lines) Blanking End
0000	1
0001	2
0002	3
.	
.	
07FF	2048

Figure 3-52. Vertical Blanking End Registers Value Assignments

Vertical Sync Pulse Start Registers (Index 28 and 29)

These read/write registers have indexes of hex 28 and 29.

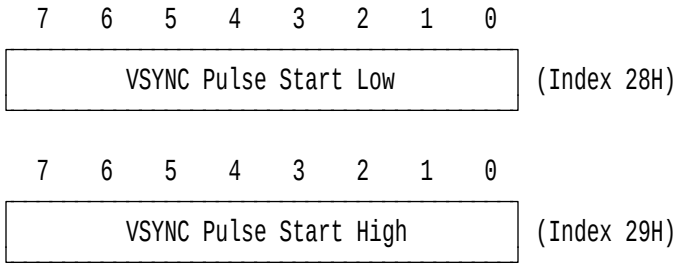


Figure 3-53. Vertical Sync Pulse Start Registers, Indexes Hex 28 and 29

The Vertical Sync Pulse Start Low and Vertical Sync Pulse Start High registers (bits 7– 0) define the position of the start of the vertical sync pulse relative to the start of the active picture area in units of one scan line. They *must* be loaded as a 16-bit value in the range hex 0000 to 07FF. Values are assigned as shown in the following figure.

Value (hex)	Sync Pulse Start (Scan Lines)
0000	1
0001	2
0002	3
.	
.	
07FF	2048

Figure 3-54. Vertical Sync Pulse Start Registers Value Assignments

Vertical Sync Pulse End Register (Index 2A)

This read/write register has an index of hex 2A.

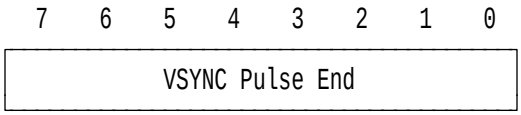


Figure 3-55. Vertical Sync Pulse End Register, Index Hex 2A

The Vertical Sync Pulse End register (bits 7– 0) defines the position of the end of the vertical sync pulse. The value loaded is the least significant byte of a 16-bit value that defines the end of the vertical sync pulse relative to the start of the active picture area in units of one scan line. The vertical sync end position *must* be within 31 scan lines of the vertical sync start position.

Note: Before setting the Operating Mode register (address 21x0) into VGA or 132-column text mode, bit 5 of this register must be set to 1.

This register may not return the value written, but the returned value is valid for save/restore operations.

Vertical Line Compare Registers (Index 2C and 2D)

These read/write registers have indexes of hex 2C and 2D.

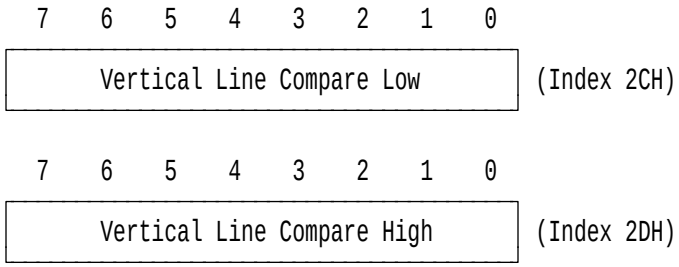


Figure 3-56. Vertical Line Compare Registers, Indexes Hex 2C and 2D

The Vertical Line Compare Low and Vertical Line Compare High registers (bits 7– 0) define the position of the end of the scrollable picture area relative to the start of the active picture area in units of one scan line. They *must* be loaded as a 16-bit value in the range hex 0000 to 07FF. Values are assigned as shown in the following figure.

Value (hex)	Scrollable End (scan lines)
0000	1
0001	2
0002	3
.	
.	
07FF	2048

Figure 3-57. Vertical Line Compare Registers Value Assignments

Sprite Horizontal Start Registers (Index 30 and 31)

These read/write registers have indexes of hex 30 and 31.

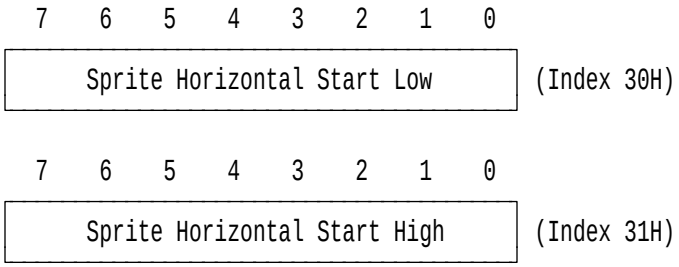


Figure 3-58. Sprite Horizontal Start Registers, Indexes Hex 30 and 31

The Sprite Horizontal Start Low and Sprite Horizontal Start High registers (bits 7– 0) define the position of the start of the sprite relative to the start of the active picture area in PELs. They *must* be loaded with a 16-bit value in the range hex 0000 to 07FF. See “Sprite Positioning” on page 3-28. Values are assigned as shown in the following figure.

Value (hex)	Sprite Start (PELs)
0000	0
0001	1
0002	2
.	
.	
07FF	2047

Figure 3-59. Sprite Horizontal Start Registers Value Assignments

Sprite Horizontal Preset Register (Index 32)

This read/write register has an index of hex 32.

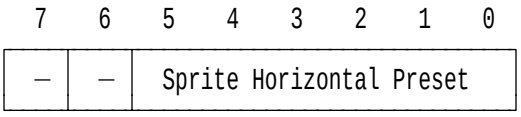


Figure 3-60. Sprite Horizontal Preset, Index Hex 32

The register fields are defined as follows:

Sprite Horizontal Preset
The Sprite Horizontal Preset field (bits 5– 0) defines the horizontal position within the 64 x 64-PEL sprite area where the sprite starts. The sprite always ends at position 63 (it does not wrap). See “Sprite Positioning” on page 3-28. Values are assigned as shown in the following figure.

Value (hex)	Sprite Start (PELs)
00	0
01	1
02	2
.	
.	
3F	63

Figure 3-61. Sprite Horizontal Preset Value Assignments

Sprite Vertical Start Registers (Index 33 and 34)

These read/write registers have indexes of hex 33 and 34.

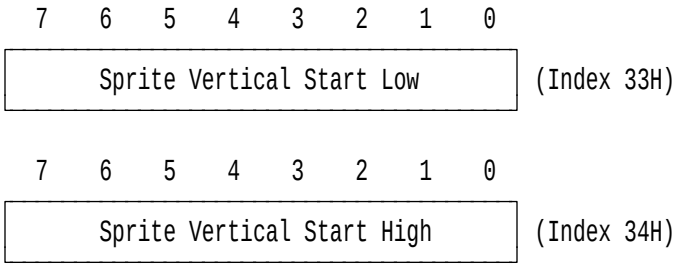


Figure 3-62. Sprite Vertical Start Registers, Indexes Hex 33 and 34

The Sprite Vertical Start Low and Sprite Vertical Start High registers (bits 7– 0) define the position of the start of the sprite relative to the start of the active picture area in units of one scan line. They *must* be loaded with a 16-bit value in the range hex 0000 to 07FF. See “Sprite Positioning” on page 3-28. Values are assigned as shown in the following figure.

Value (hex)	Sprite Start (Scan Lines)
0000	0
0001	1
0002	2
.	
.	
07FF	2047

Figure 3-63. Sprite Vertical Start Registers Value Assignments

Sprite Vertical Preset Register (Index 35)

This read/write register has an index of hex 35.

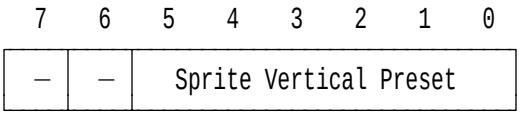


Figure 3-64. Sprite Vertical Preset, Index Hex 35

The register fields are defined as follows:

Sprite Vertical Preset
The Sprite Vertical Preset field (bits 5– 0) defines the vertical position within the 64 x 64-PEL sprite area where the sprite starts. The sprite always ends at position 63 (it does not wrap). See “Sprite Positioning” on page 3-28. Values are assigned as shown in the following figure.

Value (hex)	Sprite Start (PELs)
00	0
01	1
02	2
.	
.	
3F	63

Figure 3-65. Sprite Vertical Preset Value Assignments

Sprite Control Register (Index 36)

This read/write register has an index of hex 36.

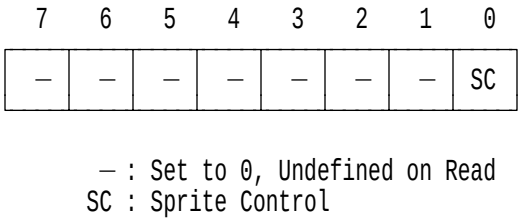


Figure 3-66. Sprite Control Register, Index Hex 36

The register fields are defined as follows:

SC The Sprite Control field (bit 0) controls the visibility of the sprite. When set to 1, the sprite appears on the screen at the location controlled by the sprite position registers. When set to 0, a sprite is not displayed. This bit must be set to 0 before any attempt is made to access the sprite image in the sprite buffer, otherwise the sprite buffer contents are corrupted.

Sprite Color Registers (Index 38– 3D)

These read/write registers have indexes of hex 38 through 3D.

Figure 3-67. Sprite Color Registers, Indexes Hex 38– 3D

The Sprite Color registers (bits 7– 0) define the red, green, and blue components of the PELs displayed when the sprite data for those PELs selects color 0 or color 1. These colors are passed directly to the DACs, not through the palette, and must be programmed to give the actual color required.

Note: The XGA-NI Subsystem uses all 8 bits of these registers. The XGA subsystem only uses the 6 most-significant bits.

Display PEL Map Offset Registers (Index 40– 42)

These read/write registers have indexes of hex 40 through 42.

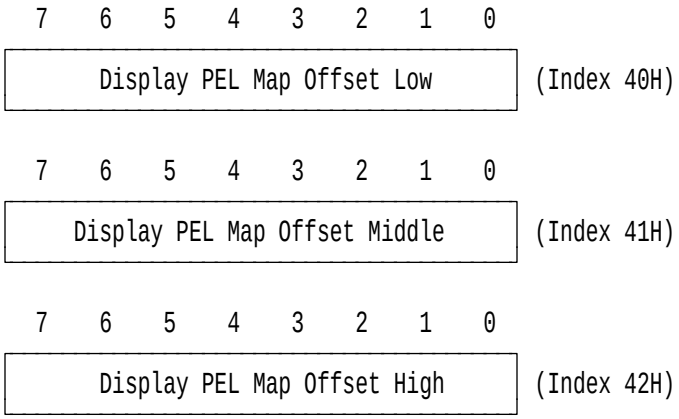


Figure 3-68. Display PEL Map Offset Registers, Indexes Hex 40– 42

The Display PEL Map Offset registers (bits 7– 0) define the address of the start of the visible portion of the video buffer in units of 8 bytes. They *must* be loaded as a single value in the range hex 00000 to 1FFFF. See “Scrolling” on page 3-25. Values are assigned as shown in the following figure.

Value (hex)	Display PEL Map Offset (bytes)
00000	0
00001	8
00002	16
.	
.	
1FFFF	1048568

Figure 3-69. Display PEL Map Offset Registers Value Assignments

Display PEL Map Width Registers (Index 43 and 44)

These read/write registers have indexes of hex 43 and 44.

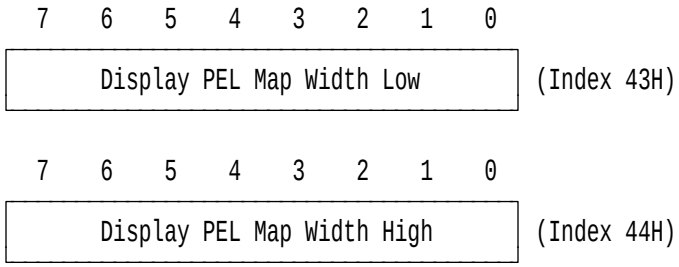


Figure 3-70. Display PEL Map Width Registers, Indexes Hex 43 and 44

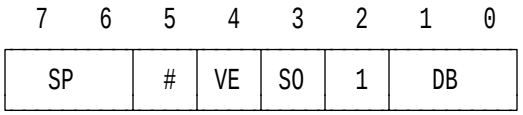
The Display PEL Map Width Low and Display PEL Map Width High registers (bits 7– 0) define the width of the display PEL map in units of 8 bytes. They *must* be loaded as a single value in the range hex 000 to 3FF. See “Scrolling” on page 3-25. Values are assigned as shown in the following figure.

Value (hex)	Display PEL Map Width (bytes)
000	0
001	8
002	16
.	
.	
7FF	16376

Figure 3-71. Display PEL Map Width Registers Value Assignments

Display Control 1 Register (Index 50)

This read/write register has an index of hex 50.




```
 # : Preserve Value Read When Writing
 1 : Set to 1, Undefined on Read
SP : SYNC Polarity
VE : Video Extension
SO : Display Scan Order
DB : Display Blanking
```

Figure 3-72. Display Control 1 Register, Index Hex 50

The register fields are defined as follows:

SP The SYNC Polarity field (bits 7, 6) value is assigned as shown in the following figure.

SP Field (binary)	Vertical	Horizontal	Lines
0 0	+	+	768
0 1	+	–	400
1 0	–	+	350
1 1	–	–	480

Figure 3-73. Sync Polarity Bit Assignments

VE The Video Extension field (bit 4) must be set to 1 (enabled) when the subsystem is in VGA mode. It must be be set to 0 (disabled) if the subsystem is in Extended Graphics or 132-column text mode.

SO The Display Scan Order field (bit 3) determines whether the display scan order is interlaced. When set to 0, the display scan order is not interlaced. When set to 1, the display scan order is interlaced.

DB The Display Blanking field (bits 1, 0) value is assigned as shown in the following figure.

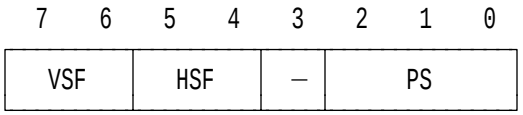
DB Field (binary)	Display Blanking
00	Display Blanked, CRT Controller Reset
01	Display Blanked, Prepare for Reset
10	Reserved
11	Normal Operation

Figure 3-74. Display Blanking Bit Assignments

When resetting the CRT controller, the display blanking bits must be set to 01 (prepare for reset) first, followed by 00 (CRT controller reset).

Display Control 2 Register (Index 51)

This read/write register has an index of hex 51.




```
  – : Set to 0, Undefined on Read
VSF : Vertical Scale Factor
HSF : Horizontal Scale Factor
 PS : PEL Size
```

Figure 3-75. Display Control 2 Register, Index Hex 51

The register fields are defined as follows:

VSF The Vertical Scale Factor field (bits 7, 6) controls how many times each line is replicated. Values are assigned as shown in the following figure.

VSF Field (binary)	Vertical Scale Factor
0 0	1
0 1	2
1 0	4
1 1	Reserved

Figure 3-76. Vertical Scale Factor Bit Assignments

HSF The Horizontal Scale Factor field (bits 5, 4) controls how many times each PEL is replicated horizontally. Values are assigned as shown in the following figure.

HSF Field (binary)	Horizontal Scale Factor
0 0	1
0 1	2
1 0	4
1 1	Reserved

Figure 3-77. Horizontal Scale Factor Bit Assignments

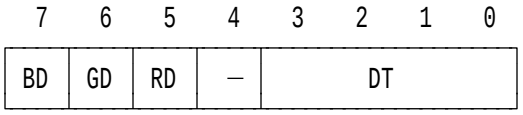
PS The PEL Size field (bits 2– 0) defines the PEL size (for the serializer, palette, and DAC), and the display scale factors (horizontal and vertical). Values are assigned as shown in the following figure.

PS Field (binary)	PEL Size
0 0 0	1 Bit
0 0 1	2 Bits
0 1 0	4 Bits
0 1 1	8 Bits
1 0 0	16 Bits (Direct Color Mode)
1 0 1	Reserved
1 1 0	Reserved
1 1 1	Reserved

Figure 3-78. Display Control 2 Register PEL Size Bit Assignments

Display ID and Comparator Register (Index 52)

This read-only register has an index of hex 52. *Do not write to* this register.



– : Undefined on Read
BD : Blue DAC Comparator Status
GD : Green DAC Comparator Status
RD : Red DAC Comparator Status
DT : Display Type

Figure 3-79. Display ID and Comparator, Index Hex 52

The register fields are defined as follows:

! **BD** The Blue DAC Comparator Status field (bit 7) indicates the state of the blue DAC output. When read as 1, the blue DAC output is low; when read as 0, the blue DAC output is high.

! **GD** The Green DAC Comparator Status field (bit 6) indicates the state of the green DAC output. When read as 1, the green DAC output is low; when read as 0, the green DAC output is high.

! **RD** The Red DAC Comparator Status field (bit 5) indicates the state of the red DAC output. When read as 1, the red DAC output is low; when read as 0, the red DAC output is high.

DT The Display Type field (bits 3– 0) indicates the type of display attached. Bit values are defined by displays.

Clock Frequency Select 1 Register (Index 54)

This read/write register has an index of hex 54.

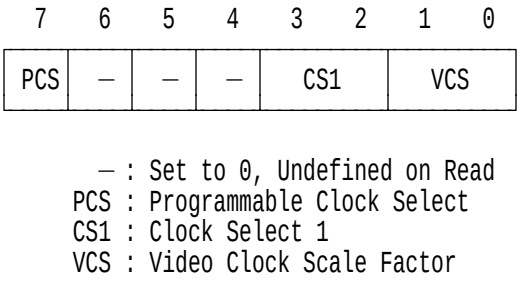


Figure 3-80. Clock Frequency Selector Register, Index Hex 54

The Clock Frequency Select 1 register must be used in conjunction with the Clock Frequency Select 2 register. It is defined under “Clock Frequency Select 2 Register (Index 70)” on page 3-87.

Border Color Register (Index 55)

This read/write register has an index of hex 55.

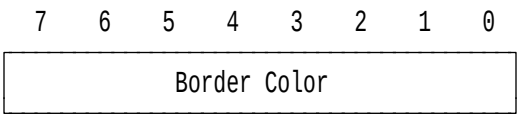


Figure 3-81. Border Color Register, Index Hex 55

The Border Color register (bits 7– 0) holds the border color palette index, which is the index of the palette location selected to be displayed in the picture border area of the display.

The inverse of bit 7 is used for palette address bit 7 in direct color mode. See “Direct Color Mode” on page 3-31.

Programmable PEL Clock Register (Index 58)

This read/write register has an index of hex 58. It is only available on the XGA-NI Subsystem.

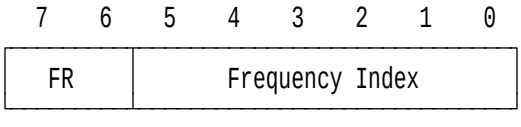


Figure 3-82. Programmable PEL Clock register, Index Hex 58

The register is defined as follows:

FR The FR field (bits 7– 6) defines the range of frequencies that can be programmed by the Frequency Index field. The value in this field defines the Division Factor used in the formula below. Values are assigned as shown in the following figure.

FR Field (binary)	Division Factor	Frequency Range
0 0	4	16.25MHz to 32.00MHz in 0.25MHz increments
0 1	2	32.50MHz to 64.00MHz in 0.50MHz increments
1 0	1	65.00MHz to 128.00MHz in 1.00MHz increments
1 1	–	Reserved

Figure 3-83. Programmable Frequency Ranges

Frequency Index The Frequency Index field (bits 5– 0) in conjunction with the Division Factor, defines the PEL frequency. The value loaded in the Frequency Index field is derived from the following formula:

$$\text{Frequency Index} = (f \times \text{Division Factor}) - 65$$

where f is the PEL Frequency required.

Note: The maximum PEL Frequency that should be programmed for the XGA-NI Subsystem is 90MHz.

Direct Color Control Register (Index 59)

This read/write register has an index of hex 59. It is only available on the XGA-NI Subsystem.

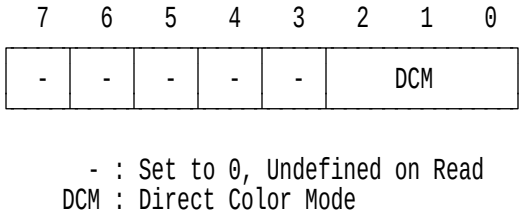


Figure 3-84. Direct Color Control register, Index Hex 59

The register is defined as follows:

DCM The DCM field (bits 2– 0) selects which algorithm is used to define the low order DAC bits which are not defined in the 16 bit Color Value. This field only takes effect when the XGA subsystem is displaying in Direct Color mode.

DCM Field (binary)	Algorithm
0 0 0	Palette Defined (see “Direct Color Mode” on page 3-31)
0 0 1	Missing bits set to 1 if Color is Non Zero
0 1 0	Missing bits set to 0
0 1 1	Missing bits set to 1
1 0 0	Missing bits equal to the most significant bits of same color field
1 0 1	Reserved
1 1 0	Reserved
1 1 1	Reserved

Figure 3-85. Direct Color Modes

Sprite/Palette Index Registers (Index 60 and 61)

These read/write registers have indexes of hex 60 and 61.

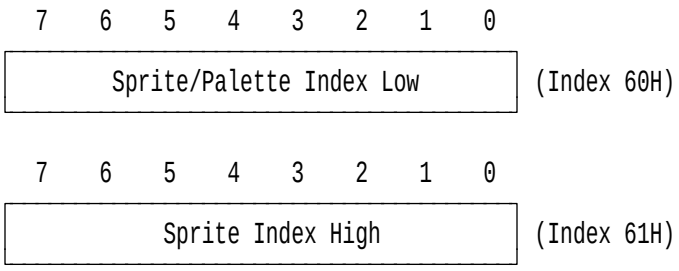


Figure 3-86. Sprite/Palette Index Registers, Indexes Hex 60 and 61

The Sprite/Palette Index Low and Sprite Index High registers (bits 7– 0) specify the index when writing to the sprite or the palette. See “Sprite Buffer Accesses” on page 3-27 and “Palette Accesses” on page 3-29 for details of these registers.

The Sprite/Palette Index Low register is used for the 256 locations of the palette that are available. It can be loaded with any palette index value in the range hex 00 to FF.

The Sprite/Palette Index Low and the Sprite Index High registers are both used to access the sprite. The registers can be loaded with any sprite index value in the range hex 0000 to 3FFF.

Accessing these registers does not cause any action other than loading or returning the value of the next index.

The registers must be saved, and subsequently restored, by any interrupting task that uses the palette or sprite registers.

Sprite/Palette Prefetch Index Registers (Index 62 and 63)

These read/write registers have indexes of hex 62 and 63.

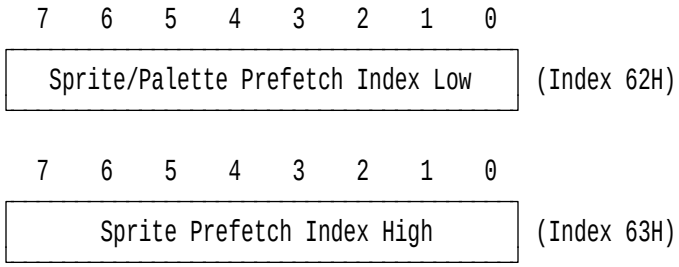


Figure 3-87. Sprite/Palette Prefetch Index Registers, Indexes 62 and 63

The Sprite/Palette Prefetch Index Low and Sprite Prefetch Index High registers (bits 7– 0) specify the index when reading from the sprite or the palette. See “Sprite Buffer Accesses” on page 3-27 and “Palette Accesses” on page 3-29 for details of these registers.

When reading from the palette, the Sprite/Palette Prefetch Index Low register must be used. Writing the Sprite/Palette Prefetch Index Low register also causes the palette prefetch registers to be loaded, and the index value to be incremented.

When reading from the sprite, use either the Sprite/Palette Prefetch Index Low register or the Sprite Prefetch Index High register. Writing to either register also causes the sprite prefetch registers to be loaded, and the index value to be incremented as a single value.

These registers must *not* be saved and subsequently restored in hardware task switches.

Palette Mask Register (Index 64)

This read/write register has an index of hex 64.

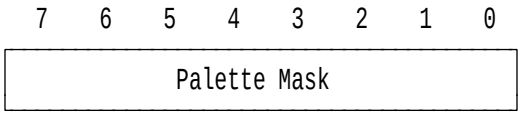


Figure 3-88. Palette Mask Register, Index Hex 64

The contents of the Palette Mask register (bits 7– 0) are ANDed with each display memory PEL value, and the result is used to index the palette.

Palette Data Register (Index 65)

This read/write register has an index of hex 65.

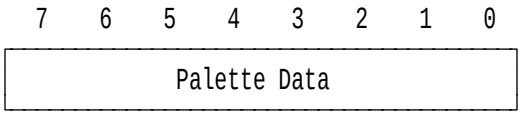


Figure 3-89. Palette Data Register, Index Hex 65

The Palette Data register (bits 7– 0) is an image of the currently selected palette RAM location. The data returned on read may not be that last written because of the selection mechanism described in “Palette Accesses” on page 3-29.

For monochrome displays, all of the palette red and blue locations *must* be loaded with 0's. Alternatively on the XGA-NI subsystem only, the Red and Blue DAC outputs can be blanked using the BRB field of the “Miscellaneous Control Register (Index 6C)” on page 3-85.

Note: The XGA-NI Subsystem uses all 8 bits of these registers. The XGA subsystem only uses the 6 most-significant bits.

Palette Sequence Register (Bits 2– 0 only) (Index 66)

This read/write register has an index of hex 66.

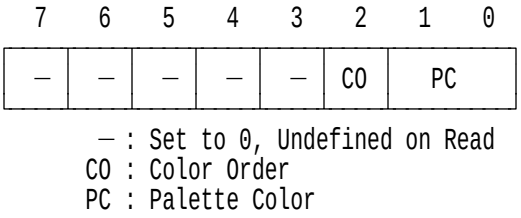


Figure 3-90. Palette Sequence Register, Index Hex 66

The register fields are defined as follows:

CO The Color Order field (bit 2) defines the sequence to be followed for selecting the red, green, and blue elements during successive Palette Data register accesses. The color order is shown in the following figure.

CO Field (binary)	Color Order
0	R,G,B,R,G,B,...
1	R,B,G,x,R,B,G,x,..

Note: x = discarded data.

Figure 3-91. Palette Sequence Register Color Order Bit Assignment

PC The Palette Color field (bits 1, 0) defines which of the red, green, or blue elements of the currently selected palette location is the current one for the Palette Data register. The palette color selection is shown in the following figure.

PC Field (binary)	Color
0 0	R
0 1	G
1 0	B
1 1	x

Note: x = discarded data.

Figure 3-92. Palette Sequence Register Palette Color Bit Assignments

See “Palette Accesses” on page 3-29 for more information.

Palette Red Prefetch Register (Index 67)

This read/write register has an index of hex 67.

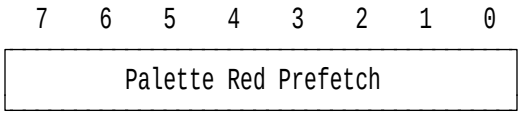


Figure 3-93. Palette Red Prefetch Register, Index Hex 67

The Palette Red Prefetch register (bits 7– 0) is not used for any normal function but must be saved, and subsequently restored, by any interrupting code that uses the sprite or palette registers.

Palette Green Prefetch Register (Index 68)

This read/write register has an index of hex 68.

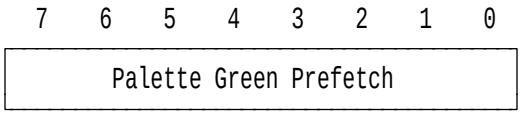


Figure 3-94. Palette Green Prefetch Register, Index Hex 68

The Palette Green Prefetch register (bits 7– 0) is not used for any normal function but must be saved, and subsequently restored, by any interrupting code that uses the sprite or palette registers.

Palette Blue Prefetch Register (Index 69)

This read/write register has an index of hex 69.

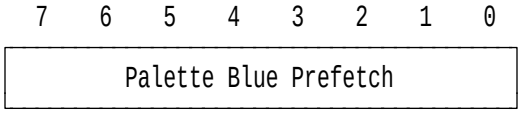


Figure 3-95. Palette Blue Prefetch Register, Index Hex 69

The Palette Blue Prefetch register (bits 7– 0) is not used for any normal function but must be saved, and subsequently restored, by any interrupting code that uses the sprite or palette registers.

Sprite Data Register (Index 6A)

This read/write register has an index of hex 6A.

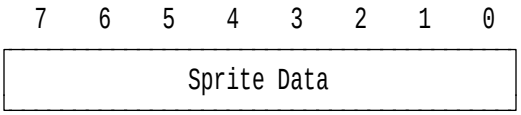


Figure 3-96. Sprite Data Register, Index Hex 6A

The Sprite Data register (bits 7– 0) is an image of the currently selected sprite buffer location. The data returned on read may not be that last written because of the selection mechanism described in “Sprite Buffer Accesses” on page 3-27.

When used for writing sprite data, the sprite PELs are Intel format packed PELs.

Sprite Prefetch Register (Index 6B)

This read/write register has an index of hex 6B.

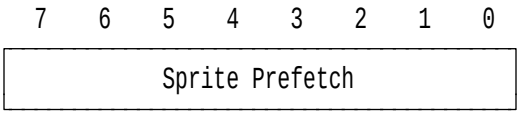


Figure 3-97. Sprite Prefetch Register, Index Hex 6B

The Sprite Prefetch register (bits 7– 0) is not used for any normal function but must be saved, and subsequently restored, by any interrupting code that uses the sprite or palette registers.

Miscellaneous Control Register (Index 6C)

This read/write register has an index of hex 6C. It is only available on the XGA-NI Subsystem.

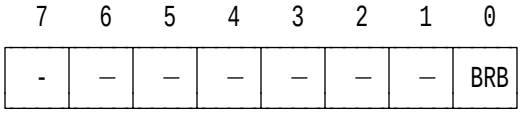


Figure 3-98. Miscellaneous Control Register, Index 6C

The register is defined as follows:

BRB When the BRB field (bit 0) is set to 1, the outputs of the Red and Blue DACs are forced to zero, regardless of the DAC inputs. This should be used when a monochrome display is connected to the subsystem.

When the BRB field is set to 0, the DACs function as normal.

MFI Control Register (Index 6D)

This read/write register has an index of hex 6D. It is only available on the XGA-NI Subsystem. This register can only be accessed (written or read) when the MFI function has been enabled. See "Main Frame Interactive (MFI) Support" on page 3-17 and "Operating Mode Register (Address 21x0)" on page 3-36.

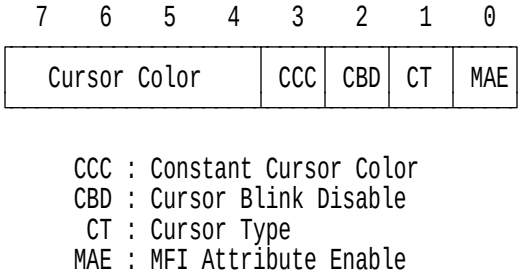


Figure 3-99. MFI Control Register, Index 6D

The register is defined as follows:

| **Cursor Color** The Cursor Color field (bits 7– 4) defines the cursor to be one of 16 colors when the CCC field is set to 1. It is defined in the same manner as the I, R, G and B fields in the attribute byte. See “Main Frame Interactive (MFI) Support” on page 3-17

| **CCC** When the CCC field (bit 3) is set to 1, a constant color cursor is displayed as defined by the Cursor Color field. When the CCC field is set to 0, the cursor color adopts the foreground color of the character upon which is it placed.

| **CBD** When the CBD field (bit 2) is set to 1, a non-blinking cursor is displayed. When the CBD field is set to 0, the cursor will blink at the MFI rate. See “Main Frame Interactive (MFI) Support” on page 3-17.

| **CT** When the CT field (bit 1) is set to 0, the cursor forces the foreground color of the character upon which it is placed. When the CT field is set to 1, the cursor forces the reverse video of the character upon which it is placed.

| **MAE** When the MAE field (bit 0) is set to 0, normal VGA characters attributes are displayed and the other fields in this register have no effect. When set to 1 the MFI character attibutes are displayed and the other fields of this register become active.

| Clock Frequency Select 2 Register (Index 70)

| This read/write register has an index of hex 70. It is used with the | Clock Frequency Select 1 Register (index hex 54) for clock | selection.

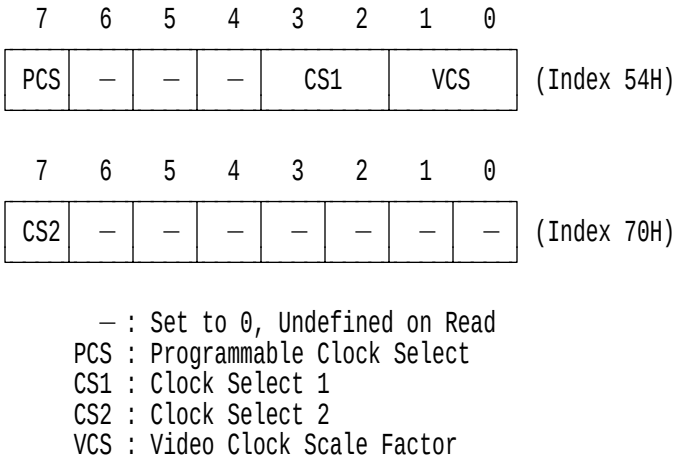


Figure 3-100. Clock Frequency Select Registers

The clock frequency select registers fields are defined as follows:

PCS The Programmable Clock Select field (bit 7 of the Clock Frequency Select 1 register) is used in conjunction with the Clock Select 1 and Clock Select 2 fields. See the description of the Clock Select 1 field to learn how it is used.

CS2 The Clock Select 2 field (bit 7 of the Clock Frequency Select 2 register) is used in conjunction with the Clock Select 1 and Programmable Clock Select fields. See the description of the Clock Select 1 field to learn how it is used.

CS1 The Clock Select 1 field (bits 3, 2 of the Clock Frequency Select 1 register) must be used in conjunction with the Clock Select 2 and Programmable Clock Select fields Clock selection is shown in the following figure.

PCS Field (binary)	CS2 Field (binary)	CS1 Field (binary)	Selected Clock
0	0	0 0	VGA 8-PEL Character Mode and 640 x 480 Graphics Mode Clock
0	0	0 1	VGA 9-PEL Character Mode Clock
0	0	1 0	Clock Sourced from Video Extension Interface
0	0	1 1	1024 x 768 Graphics Mode Clock
0	1	0 0	132-Column Text Mode Clock (8 PEL Characters)
0	1	0 1	Reserved
0	1	1 0	Reserved
0	1	1 1	Reserved
1	0	0 0	Programmed PEL Clock
1	0	0 1	Reserved
1	0	1 0	Reserved
1	0	1 1	Reserved
1	1	0 0	Reserved
1	1	0 1	Reserved
1	1	1 0	Reserved
1	1	1 1	Reserved

Figure 3-101. Clock Selected Bit Assignments

Note: See "Effects of VGA & XGA Mode Setting on Video Memory" on page 3-229 for details of mode setting.

VCS The Video Clock Scale Factor field (bits 1, 0 of the Clock Frequency Select 1 register) controls the divide ratio of the selected video clock before it is used by the CRT controller. The operation of the video clock scale factor is invisible to the programmer, but it must be set as shown for correct operation of the hardware.

VCS Field (binary)	Video Clock Scale Factor	Mode
0 0	1	VGA and 640 x 480 Graphics Modes
0 1	2	1024 x 768 Graphics and 132-Column Text Modes
1 0	Reserved	
1 1	Reserved	

Figure 3-102. Video Clock Scale Factor Bit Assignments

Coprocessor Description

The XGA coprocessor provides autonomous drawing functions for the video subsystem. Autonomous drawing functions means that the coprocessor draws into memory (either video memory or system memory) independently of the system microprocessor, while the system microprocessor is performing some other operation.

The coprocessor supports 1, 2, 4, 8 or 16 bits-per-PEL on the XGA-NI subsystem. Support is limited to 1, 2, 4, or 8 bits-per-PEL on the XGA Subsystem. See “Direct Color Mode” on page 3-271 for details of using the coprocessor when displaying in 16 bits-per-PEL (direct color) mode.

The execution of an operation using the coprocessor involves the following steps:

1. The system microprocessor sets up the coprocessor registers to perform a particular operation.
2. The system microprocessor writes to the PEL Operations register to start the coprocessor operation.
3. The coprocessor performs the drawing operation. The system microprocessor can be performing some other function at this time.
4. The coprocessor completes the drawing operation, informs the system microprocessor, and becomes idle.
5. The process is repeated.

The coprocessor operates on PELs within PEL maps. A PEL map is an area of memory at a given address with a defined height, width, and PEL format (see “PEL Maps” on page 3-96).

PELs from a source are combined with PELs from a destination under the control of a pattern and mask, and the result is written back to the destination.

After each access, the source, destination, pattern, and mask addresses are updated according to the function being performed, and the operation is repeated until a programmed limit is encountered.

The drawing operation can be a PEL block transfer (PxBlt), Bresenham line draw, or draw and step.

The function performed to combine the source and destination data can be a logical or arithmetic operation. One of two possible operations is selected for each PEL by the value of the corresponding pattern PEL. A mask PEL for each PEL protects the destination from update.

Pattern data can be generated automatically from source data by detecting PELs with a value of 0.

A color compare function allows the modifying of the destination PEL to be dependent on the value of the destination PEL, compared to a programmable value.

Three general purpose PEL maps can be defined in memory. Each map has a defined start address, width and height in PELs, and number of bits-per-PEL. Source, pattern, and destination data can reside in any combination of these maps. There is also a mask map with its defined start address, width, height, and format. Mask data is always taken from this map.

Source, pattern, and destination data are each addressed by unique X,Y pointers. Mask data is addressed by the destination X,Y pointers (see Figure 3-104 on page 3-98). If the source or pattern X,Y pointers move outside the defined extremities of their PEL maps, they are reset automatically to wrap round to the opposite side of the PEL map. If the destination X,Y pointers move outside the extremities of the destination map, update of the destination map is inhibited until the X,Y pointers move back inside the map.

Figure 3-103 on page 3-92 represents the coprocessor graphics data flow for the passage of one PEL. In reality, multiple PELs are processed in one cycle.

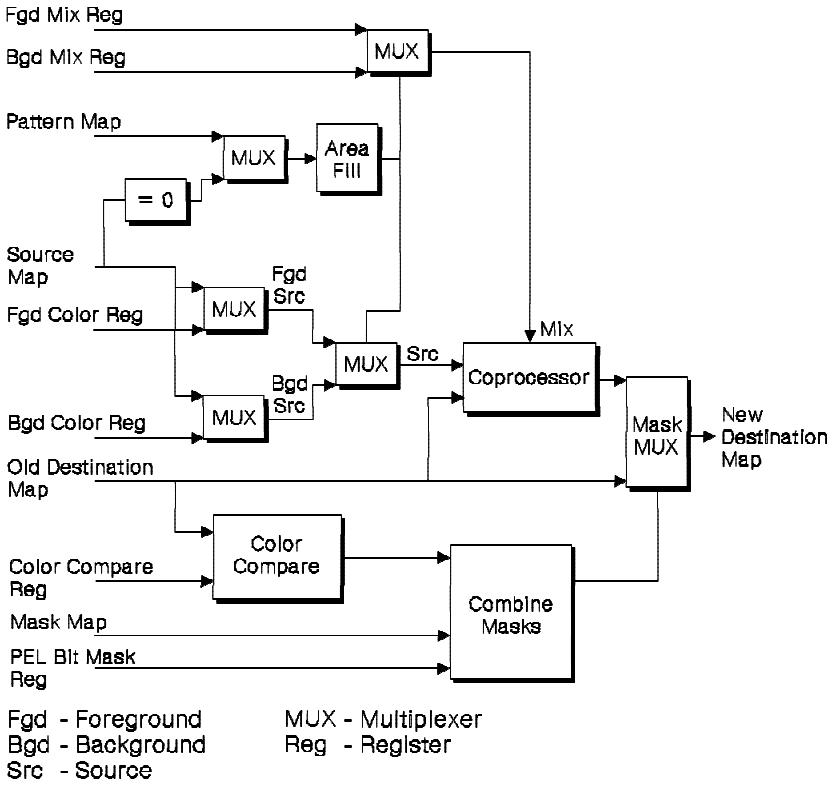


Figure 3-103. Coprocessor Data Flow

Programmer's View

An operation is defined as the execution of a single PxBlt, line draw, or draw and step function.

An operation is set up by first loading the registers of the coprocessor with appropriate data, such as X,Y coordinates, function mixes, and dimensions. Then the operation is initiated by writing to the PEL Operations register. This defines the flow of data in the operation and starts the operation. The coprocessor then executes the operation and completes it when some programmed limit is reached.

There is one exception to this sequence of initiating operations: the draw and step function (see the section “Draw and Step” on page 3-108).

The XGA can be programmed to inform the system microprocessor of the completion of an operation using a system interrupt. This interrupt is called the coprocessor-operation-complete interrupt. An enable bit and status bit exist for this interrupt in the “Interrupt Enable Register (Address 21x4)” on page 3-39 and “Interrupt Status Register (Address 21x5)” on page 3-41.

A mechanism is provided to let the system microprocessor suspend or terminate an operation before it is completed. The suspension of operations is required to allow task switches, while termination of operations can be used to recover from errors.

PEL Formats

| On the XGA-NI subsystem, the coprocessor can manipulate images
| with 1, 2, 4, 8 or 16 bits-per-PEL. On the XGA subsystem, it can
| manipulate images with 1, 2, 4, or 8 bits-per-PEL.

| It manipulates packed-PEL data, so each data doubleword (32 bits)
| contains:

| **XGA-NI Subsystem** 32, 16, 8, 4 or 2 PELs respectively.

| **XGA Subsystem** 32, 16, 8 or 4 PELs respectively.

The PELs can be in Motorola or Intel format. See Figure 3-6 on page 3-21 and Figure 3-7 on page 3-22 for Intel and Motorola formats.

Each PEL map manipulated by the coprocessor can be defined as either Motorola or Intel format. If the destination map has a different format than the source, pattern, or mask maps, the coprocessor automatically translates between the two formats.

Motorola or Intel format is controlled by a bit in the PEL Map Format register.

PEL Fixed and Variable Data

When executing an operation, the coprocessor reads source, pattern, and mask data, and reads and writes destination data. The source, pattern, and mask data can be fixed throughout the operation, or it can vary from PEL to PEL.

If fixed data is used, it is written to the relevant fixed data register in the coprocessor before the operation is started (Foreground Color and Background Color registers).

If variable data is required, the data is read from memory by the coprocessor during the operation. The coprocessor only allows variable data from memory, and does not let the system unit system microprocessor supply variable data.

The Coprocessor View of Memory

To the programmer, the coprocessor treats video memory and system memory the same. Data can be moved between system memory and video memory by defining PEL maps at the appropriate addresses.

Accesses to the XGA video memory are faster than accesses to system memory.

The coprocessor can address all of the video memory.

The Video Memory Base Address register and Instance indicate the base address where the video memory appears in system address space. This base address is on a 4MB address boundary. The coprocessor assumes that the whole 4MB of address space above this boundary is reserved for its own video memory. All addresses outside this 4MB block are treated as system memory. See “POS Register 4 (Base + 4)” on page 3-175.

“Direct Access to Video Memory” on page 3-22 describes video memory addressing.

PEL Maps

PEL Maps A, B, and C (General Maps)

The coprocessor defines three general purpose PEL maps in memory, called PEL maps A, B, and C. Each map is defined by four registers:

PEL Map Base Pointer	Specifies the linear start address of the map in memory.
PEL Map Width	Specifies the width of the map in PELs. The value programmed must be one less than the required width.
PEL Map Height	Specifies the height of the map in PELs. The value programmed must be one less than the required height.
PEL Map Format	Specifies the number of bits-per-PEL of the map, and whether the PELs are stored in Motorola or Intel format.

Source, pattern, and destination data reside in any of PEL maps A, B, or C, determined by the contents of the PEL Operations register.

These maps may be defined as any arbitrary size up to 4096 by 4096 PELs. Individual PELs within the maps are addressed using X,Y pointers. See “X and Y Pointers” on page 3-98.

PEL maps can be located in video memory and in system memory.

There are two restrictions on map usage: the source and destination maps must have the same number of bits-per-PEL, and the pattern map must be 1 bit-per-PEL.

PEL Map M (Mask Map)

In addition to the three general purpose maps, the coprocessor defines a mask map. This map is closely related to the destination map. It protects the destination from update on a PEL-by-PEL basis and can be used to provide a scissoring-type function on any arbitrary shaped area. See “Scissoring with the Mask Map” on page 3-103.

The mask map is described by a set of registers similar to the general purpose PEL maps A, B, and C, but it is fixed at 1 bit-per-PEL.

The mask map differs from the source, pattern, and destination maps as follows:

- The mask map uses the destination X and Y pointers.
- The position of the mask map origin relative to the destination is defined by the mask map origin X and Y offsets.

See “X and Y Pointers” on page 3-98 for more information.

Map Origin

The origin of a PEL map is the point where X = 0 and Y = 0.

The coprocessor defines the origin of all its PEL maps as being at the top left corner of the map. The direction of increasing X is to the right; the direction of increasing Y is downward. Figure 3-104 on page 3-98 illustrates the X,Y addressing of an XGA map.

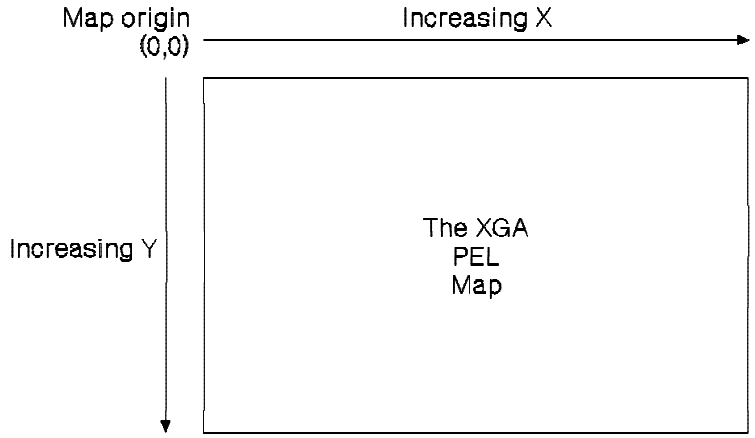


Figure 3-104. XGA PEL Map Origin

In storage, PELs to the right of and below the origin are stored in ascending, contiguous memory locations.

X and Y Pointers

The characteristics of X and Y pointers vary depending on the type of PEL map.

Source and Pattern Maps

These maps each have X and Y pointers that determine the PEL accessed for that map. The two sets of pointers are completely independent, and are modified as the operation proceeds.

If, in the course of an operation, the source or pattern pointers are moved beyond the extremities of the PEL map containing the source or pattern data, they are reset to the opposite edge of the PEL map. Source and pattern maps can be regarded as continuous, as they wrap round at their extremities. This allows a single operation to repeat a small pattern over a large area in the destination map. This is known as pattern tiling, shown in Figure 3-105 on page 3-99.

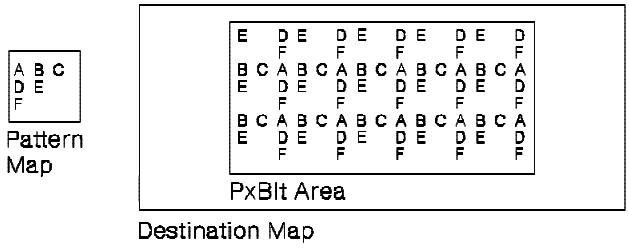


Figure 3-105. Repeating Pattern (Tiling)

Destination Map

If a destination X or Y pointer is moved beyond the extremity of the PEL map containing the destination, the pointers are not wrapped. Updates to the destination are disabled until the pointers are moved to within the defined PEL map. This mechanism is effectively a fixed scissor window around the destination PEL map.

A guardband exists around the destination map to ensure that the destination X and Y pointers do not wrap when they move outside the limits of the map. The guardband is 2048 PELs wide on all sides of the largest definable destination map.

The guardband is illustrated in Figure 3-106.

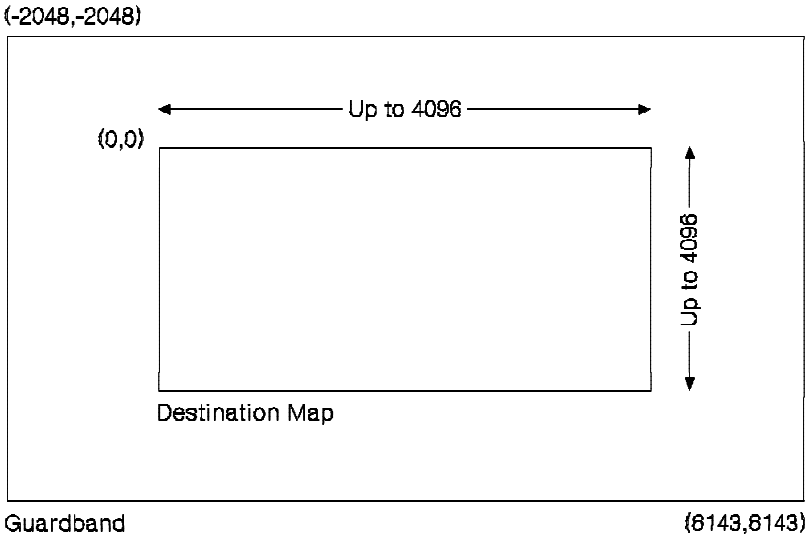


Figure 3-106. Destination Map Guardband

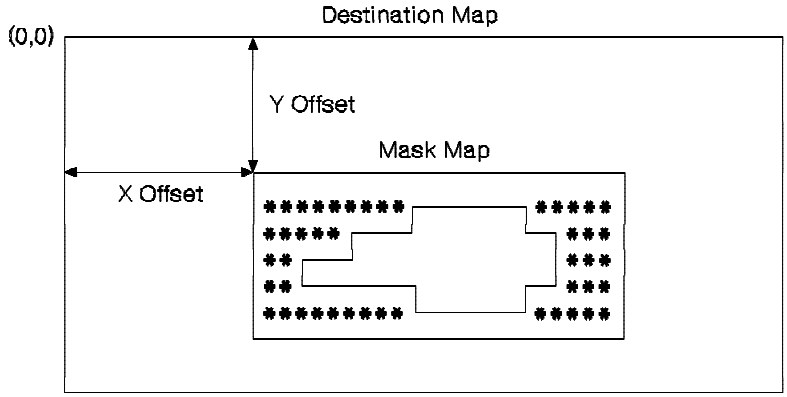
The guardband allows the destination X and Y addresses to range (– 2048 to 6143). All PELs within the destination map can be updated, but updates to PELs within the guardband are inhibited. The size of the destination map is determined by the map width and height, so PELs within the range (0,0) to (width – 1, height – 1) can be updated. The guardband occupies PEL X addresses (– 2048 to – 1),

and width to 6143, and Y addresses (– 2048 to – 1), and height to 6143.

To address the destination map correctly and take advantage of the scissor capability of the coprocessor destination boundary, X and Y destination addresses can be calculated using 16-bit twos complement numbers. All X and Y addresses generated by the operation must be within the range (– 2048 to 6143), and all PELs drawn must be inside the bounds of the destination map. Any X and Y addresses generated that are outside the range (– 2048 to 6143) cause the X and Y pointers to wrap and produce erroneous results.

Mask Map

The mask map width and height can be any size less than or equal to the dimensions of the destination map. If the mask map is smaller than the destination map, the hardware needs to know where the mask map is positioned relative to the destination map. Two pointers, the mask map origin X offset and mask map origin Y offset, specify the X,Y position where the mask map origin in the destination is located. The following figure illustrates the use of these pointers.



Note: The Mask Map is forced to have the same map origin as the Destination Map.

Figure 3-107. Mask Map Origin X and Y Offsets

The mask map takes its X and Y pointers from the destination map X and Y pointers. For every PEL in the

destination map, the corresponding PEL in the mask map is read and, depending on the value of the mask PEL, update of the destination enabled or disabled.

Scissoring with the Mask Map

Hardware scissoring is provided in the coprocessor using the mask map. The mask map can be used for any operation in three ways, as follows:

Disabled
: Contents of the mask map and boundary position are ignored.

Boundary Enabled
: Contents of the mask map are disabled, but the boundary of the mask map acts as a rectangular scissor window on the destination map. No memory is required to store the map contents in this mode.

Enabled
: Contents of the mask map can be used to provide a nonrectangular scissor window. The boundary of the mask map also provides a rectangular scissor window at the extremities of the mask map.

The mask map mode is controlled by a bit in the PEL Operation register. PELs located on a scissor boundary are treated as if they are inside it. The modes are described in the following text.

Mask Map Disabled

When the mask map is disabled, updates to the destination are performed regardless of the position or contents of the mask map. No memory must be reserved for the mask map. The contents of the PEL map M Base Pointer, Width, Height, and Format registers are ignored.

If the current operation attempts to draw outside the boundary of the destination map, the update is automatically inhibited. The destination X and Y pointers are incremented as normal, but destination update is not enabled until the pointers move back inside the bounds of the destination map. A fixed hardware scissor window then exists around the boundary of the destination map. This destination boundary scissor is enabled regardless of the mask map mode.

Figure 3-108 illustrates the destination boundary scissor operation when the mask map is disabled.

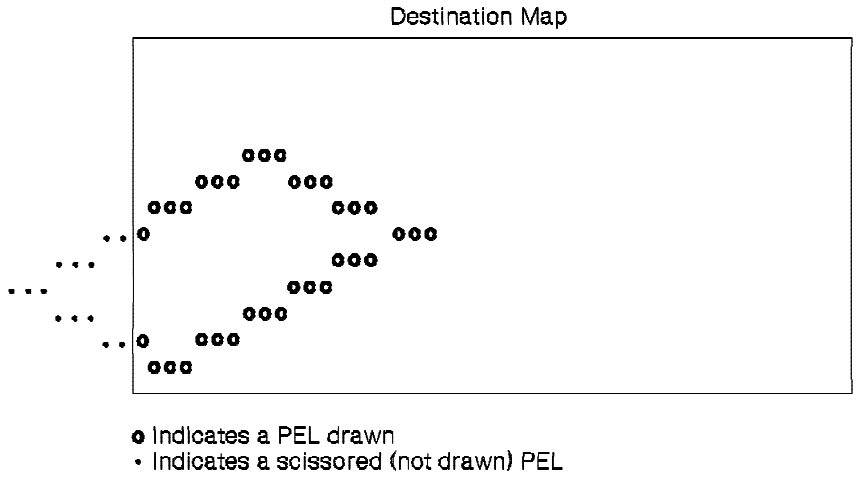


Figure 3-108. Destination Boundary Scissor

Mask Map Boundary Enabled

Mask map boundary enabled mode provides a single rectangular scissor window within the destination map. The contents of the mask map are ignored, so no memory must be reserved for the mask map.

In boundary enabled mode, the size and position of the mask map must be specified. The PEL map M Base Pointer, Width, Height, and Format registers must be defined. These four registers define a rectangular boundary within the destination map. Updates to the destination map inside this boundary take place as normal. Updates outside this boundary are inhibited.

The following figure illustrates a mask map boundary enabled scissoring operation.

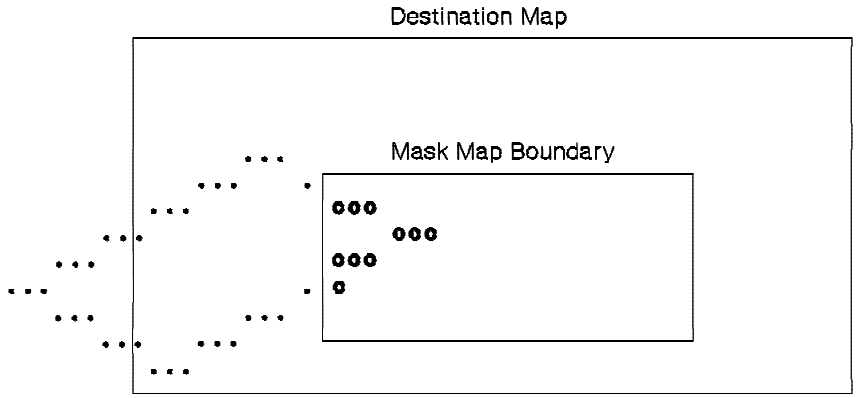


Figure 3-109. Mask Map Boundary Scissor

Mask Map Enabled

When the mask map is enabled, both the mask map boundary and contents provide scissoring action. Memory must be reserved to hold the mask map PELs. The PEL map M Base Pointer, Width, Height, and Format registers must be set up to point to the mask data, describing its size and position relative to the destination map.

Any PEL in the destination that is about to be updated has its corresponding mask map PEL examined. If the mask PEL is inactive (0), the destination PEL update is inhibited. If the mask PEL is active (1), the destination PEL is updated as normal. This mode allows drawing nonrectangular scissor windows in the mask map prior to an operation, then in a single execution of an operation, applying a nonrectangular scissor window to that operation.

Memory must be reserved to hold the mask map contents. The mask data is fixed at 1 bit-per-PEL. For a full screen mask map on a 1024 x 768 PEL screen, 96KB of memory are required. If the scissor operation does not cover the whole destination map, a mask map smaller than the destination map can be used to save memory. Applications with no memory available for the mask map contents must use the mask map boundary enabled mode.

The following figure illustrates a mask map enabled scissor operation.

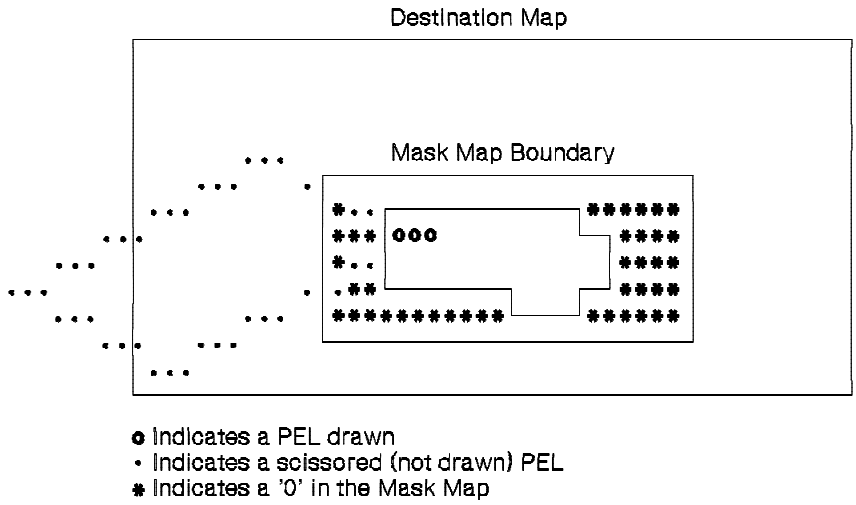


Figure 3-110. Mask Map Enabled Scissor

Before performing an operation that requires a nonrectangular scissor, the nonrectangular mask into the mask map must be drawn. Windowing systems only permit rectangular windows, so the mask can be drawn using a sequence of PxBlt operations that have fixed source data. For more complex shapes, the line draw and draw and step functions can be used to draw area outlines that can then be filled.

A large number of operations can be performed, all using the same mask, keeping the overhead per operation in setting up the mask small. The use of the mask to perform nonrectangular scissors improves the performance of a given drawing operation over a single rectangular scissor that is provided by the hardware.

Drawing Operations

The coprocessor provides four drawing operations:

- Draw and step
- Line draw
- PEL block transfer (PxBlt)
- Area fill.

The operations can be either one-dimensional or two-dimensional. Draw and step and line draw are one-dimensional while the PxBlts are two-dimensional. Draw and step and line draw are collectively called draw operations in the following text.

Either of the draw operations can be read or write. Qualifiers to the operation are described in “Line Draw” on page 3-112.

Draw and Step

This operation draws a PEL at the destination, then updates the X,Y pointers to one of the eight neighbors of the PEL according to a 3-bit code.

Up to 15 address steps can be specified in a fixed direction by each draw and step code. An 8-bit code describes the vector, as shown in Figure 3-111.

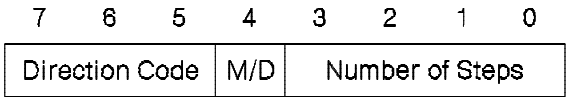


Figure 3-111. Draw and Step Code

Number of Steps

This field indicates how many steps are taken, from 0 to 15. The X,Y pointers are updated after the PEL is drawn, so a draw and step function always attempts to draw at least one PEL.

The number of steps taken in the draw and step operation is one less than the number of PELs that the hardware attempts to draw. When the number of steps is programmed to five, six PELs are drawn; when zero steps are specified, one PEL is drawn. After the draw and step operation, the X,Y pointers point to the last PEL that the operation attempted to draw (this PEL may not be drawn if the last PEL null drawing mode is active).

For example, a draw and step code of hex 35 moves X,Y pointers starting at coordinates (17,10) to coordinates (22,5), as shown in Figure 3-112.

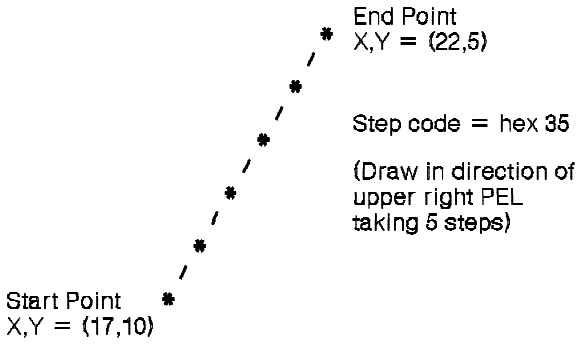


Figure 3-112. Draw and Step Example

M/D

This field specifies if the current operation is a move operation or a draw operation. When set to 1, PELs are drawn. When set to 0, X and Y pointers are modified as normal, but no PELs are drawn.

Direction Code

This field indicates the direction of drawing relative to the current PEL, as shown in Figure 3-113.

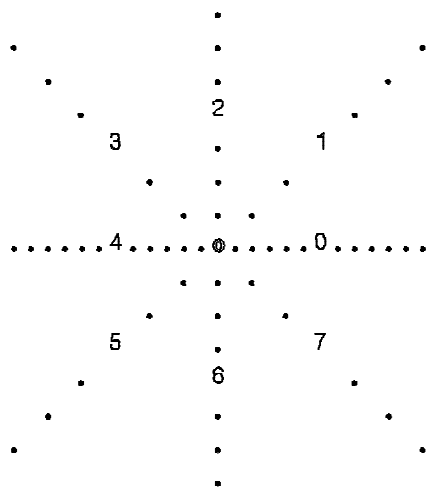


Figure 3-113. Draw and Step Direction Codes

Draw and step codes must be written to the Direction Step register. Each write to the register can load up to four draw and step codes in one access. The draw and step codes are executed starting with the least significant byte. Each group of up to four codes written to the Direction Step register is treated as one operation. All codes are executed before the coprocessor indicates that the operation is complete. However, for the purposes of first and last PEL null drawing, each code describes a distinct line.

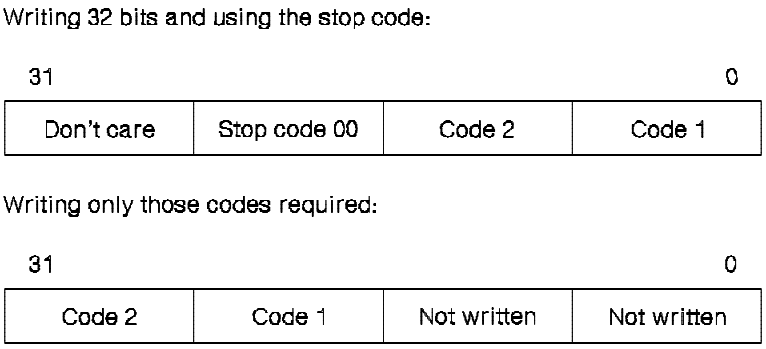
The draw and step operation differs from other operations because it is not initiated through the PEL Operations register. Writing a draw and step code to the most significant byte of the Direction Step register initiates the draw and step operation.

Before any data is written to the Direction Step register, the PEL Operation register must be loaded to specify the particular draw and step function and the data flow for the operation. Writing the PEL Operation register with a function of draw and step does not initiate a draw and step operation, but sets up the parameters for the operation. Writing steps to the Direction Step register initiates the draw and step operation. If the PEL Operation register

specifies a function other than draw and step when the Direction Step register is written, no operation takes place.

The XGA treats a draw and step code of 00 as a stop code. If a stop code is encountered as one of the four codes in the Direction Step register, the draw and step operation completes after that code has been executed. The completion of the operation is indicated in the normal way through the Coprocessor Control register. The coprocessor busy bit in the Coprocessor Control register indicates the operation has completed because a stop code was encountered. This mechanism allows software to load sequences of draw and step codes to the coprocessor without monitoring the number of codes that make up the figure being drawn.

There are two ways to program fewer than four codes to the Direction Step register. The first unwanted step code can be set to 00 (stop) and all 32 bits of the register written, or only the required number of codes can be written to the Direction Step register. In the latter case, the codes must be written to the most significant bytes of the register. The two methods are shown in Figure 3-114.



Note: The figure shows the case when only two Step codes are required. The second method requires the I/O address programmed to change depending on the number of steps written.

Figure 3-114. Programming Fewer Than Four Step Codes

Line Draw

The line draw function uses the Bresenham line drawing algorithm to draw a line of PELs into the destination. The Bresenham line drawing algorithm operates with all parameters normalized to the first octant (octant 0). The octant code for the octant in which the line lies must be specified in the Octant field of the PEL Operation register. This contains a 3-bit code made up of three 1-bit flags called DX, DY, and DZ.

DX is 1 for negative X direction, 0 for positive X direction
DY is 1 for negative Y direction, 0 for positive Y direction
DZ is 1 for $|X| \leq |Y|$, 0 for $|X| > |Y|$ ($|X|$ is the magnitude of X, the value ignoring the sign)

The Octant field is formed by concatenating DX, DY, and DZ. See Figure 3-170 on page 3-168 for octant bit value assignments.

Figure 3-115 shows the encoding of octants.

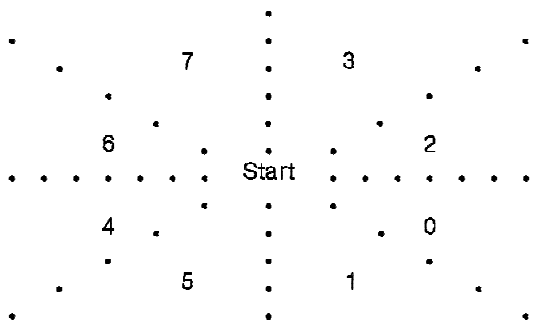


Figure 3-115. Bresenham Line Draw Octant Encoding

The length of the line (delta X when normalized) must be specified in the Operation Dimension 1 register.

The coprocessor provides the following registers to control the draw line address stepping:

- Bresenham Error Term E = 2 × deltaY − deltaX
- Bresenham Constant K1 = 2 × deltaY
- Bresenham Constant K2 = 2 × (deltaY − deltaX).

When the drawing operation has completed, X and Y pointers point at the last PEL of the line.

The coprocessor draw operations that take source data from a PEL map apply the specified address update to either the source or destination map. The X,Y address in the other map is always incremented in X only. There are two possible draw operations, Read Draw and Write Draw.

Write Draw After every PEL drawn, the source X,Y pointers are incremented in X only. The destination X,Y pointers are updated according to the current function specified (Bresenham line draw or draw and step).

Read Draw After every PEL drawn, the source X,Y pointers are updated according to the current function specified (Bresenham line draw or draw and step). The destination X,Y pointers are incremented in X only.

The read and write in the terms read draw and write draw refer to the direction of data transfer of the map having its addresses updated by the specified function. During a read line draw, the map where data is read (the source) has its addresses updated by the Bresenham line draw function. During a write draw and step, the map where data is written (the destination) has its addresses updated by the draw and step function.

Note: To draw a fixed color line (by taking the source from the Foreground Color or Background Color register), a write draw function must be used.

Figure 3-116 on page 3-114 illustrates the stepping of X and Y pointers during a read line draw and write line draw.

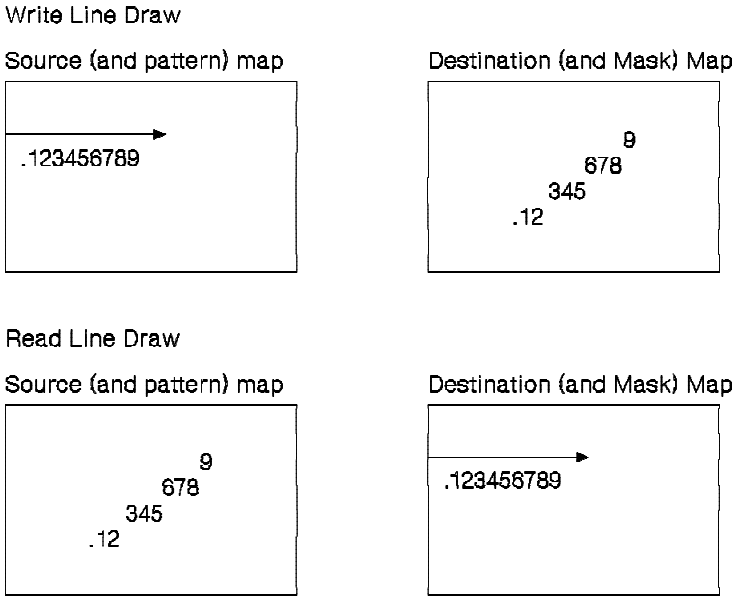


Figure 3-116. Memory to Memory Line Draw Address Stepping

In the map that is not having the current addressing function applied, the X pointer is always incremented regardless of the direction of X in the current addressing function. The Y pointer for the same map is not updated during the operation.

The above description refers to the source and destination maps. The pattern map X and Y pointers are updated in the same way as the source pointers. The mask map X and Y pointers (that are not directly accessible), are updated in the same manner as the destination pointers.

If an attempt is made to move any of the map pointers outside the bounds of their current map, the rules set out in “X and Y Pointers” on page 3-98 apply: the source and pattern pointers wrap, and the mask and destination scissor. To draw a line with a repeating color scheme and pattern, the source map width and pattern map width must be set to the required run-length of the repeating colors and pattern respectively. The coprocessor automatically draws the repeating run of colors and pattern. Conversely, if a line with a long nonrepeating color scheme or

pattern is required, the source and pattern map widths must be set equal to, or greater than, the line length, otherwise wrapping occurs.

Drawing with Null Endpoint PELs: It is common to draw a series of lines, one after the other, with the endpoint of one line being the starting point of the next line. Such composite lines are called polylines. A problem can arise because the common endpoint of the two abutting lines is drawn twice, once as the last PEL of the first line, and once as the first PEL of the second line. If a mix of XOR is active, the common PEL is drawn and removed. Similar problems arise with different mixes.

To avoid drawing the endpoints of polylines twice, the coprocessor provides functions that inhibit the drawing of the end PEL. Depending on the function selected, either the first PEL or the last PEL of individual lines is not drawn (drawn null). The choice of whether to draw first or last PEL null is arbitrary, as long as one or the other is used for the whole figure being drawn. It is usually a convention of the graphics application whether first or last PEL null is used.

First and last PEL null drawing functions are provided for both the Bresenham line draw function and the draw and step function. In all cases, the programming of parameters is the same as for normal line draw and draw and step. Only the contents of the Drawing Mode field in the PEL Operations register are different.

Area Boundary Drawing: The outline of an object is drawn using Bresenham line draw, draw and step functions, or a combination of the two. The outline is created by observing the following rules.

- If a line is drawn from screen top-to-bottom, draw with last PEL null and draw only the last PEL in every horizontal run of PELs.
- If a line is drawn from screen bottom-to-top, draw with first PEL null, and draw only the first PEL in every horizontal run of PELs.
- If a line is horizontal, draw none of the PELs.
- Always draw with a mix of XOR.

The coprocessor implements these drawing rules in hardware. A shape drawn as an area outline must be drawn as a normal line draw or draw and step operation, with the draw area boundary

drawing mode selected in the PEL Operation register and a mix of XOR.

Area Outline Scissoring: It is important during area outline drawing to ensure that the correct outline is drawn when the outline intersects the scissor boundary. In particular, when the outline is scissored by a vertical boundary at the left of a map, a PEL is drawn in the outline to activate filling at that boundary.

Using the combination of mask map and fixed destination boundary scissoring available in XGA, area outlines are incorrectly scissored by the mask map, but correctly scissored by destination map boundary scissoring. The correct area can be filled by ensuring that the mask map scissoring is disabled when the outline is drawn and enabled or boundary enabled when the scan/fill part of the area fill is drawn. This results in the correct, scissored figure being drawn. See “Scissoring with the Mask Map” on page 3-103.

PEL Block Transfer (PxBlt)

The PxBlt function transfers a rectangular block of PELs from the source to the destination. The width and height of the rectangle are specified in the Operation Dimension 1 and Operation Dimension 2 registers. The transfer can be programmed to start at any of the four corners of the rectangle, and proceed toward the diagonally opposite corner. The address is stepped in the X direction until the edge of the rectangle is encountered, then X is reset and the Y direction is stepped. This process is repeated until the entire rectangle is transferred.

PxBlts can be implemented in normal write mode or in read/modify/write mode, depending on the number of bits-per-PEL and the mix being used.

| If the PxBlt is being implemented in read/modify/write mode (that
| is, 1, 2, or 4 bits-per-PEL with *any* mix or 8 or 16 bits-per-PEL with
| a read/modify/write mix), then do one of the following:

- Ensure that the destination map has a base address that is on a doubleword (4 byte) address boundary, and is an exact number of doublewords wide.
- If the destination map is not doubleword aligned, ensure that the destination map boundary is not crossed during the PxBlt operation.

PxBlt Direction: The PxBlt direction indicates in which direction the X,Y address is stepped across the rectangle. It also defines the starting corner of the transfer. This is significant if the destination rectangle overlaps the source rectangle, so the PxBlt direction must be programmed correctly to achieve the required result.

The direction octant bits in the PEL Operations register determine the direction that the PxBlt is drawn in.

The encoding is as follows:

binary 000 or 001	Start at top left corner of area, increasing right and down.
binary 100 or 101	Start at top right corner of area, increasing left and down.
binary 010 or 011	Start at bottom left corner of area, increasing right and up.
binary 110 or 111	Start at bottom right corner of area, increasing left and up.

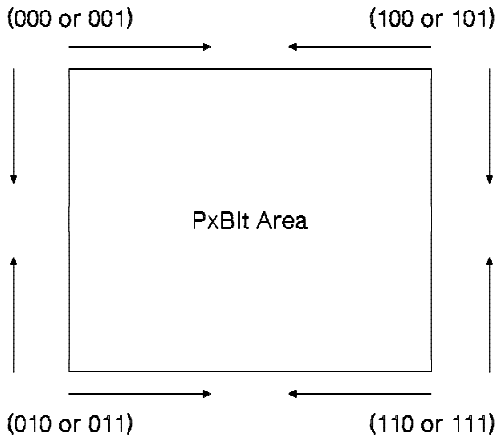


Figure 3-117. PxBlt Direction Codes

After a PxBlt operation has completed, the X and Y pointers are set so the X pointer contains its original value at the start of the PxBlt and the Y pointer points to its value on the last line of the PxBlt plus or minus 1, depending on the Y direction that the PxBlt was programmed.

See “Overlapping PxBlts” on page 3-260 for details on PxBlts where the source and destinations overlap.

Inverting PxBlt: As detailed in “Map Origin” on page 3-97, the coprocessor assumes that the origin of a PEL map is at the top left corner of the map, with Y increasing downward. Applications that use an origin at the bottom left of the map (Y increasing upward) use either of the following:

- Modify all Y coordinates by subtracting the map height from them before passing the modified coordinates to the display hardware.
- Use the coprocessor inverting PxBlt operation.

Inverting PxBlt use requires the application to draw into an off-screen PEL map without any Y coordinate modification, and then use the inverting PxBlt operation to move the data to the destination map.

Figure 3-118 on page 3-119 illustrates the X,Y addressing of the inverting PxBlt operation, and shows how the result of the inverting PxBlt appears the same as the original when displayed as an inverted PEL map (that is, with the origin at the bottom left).

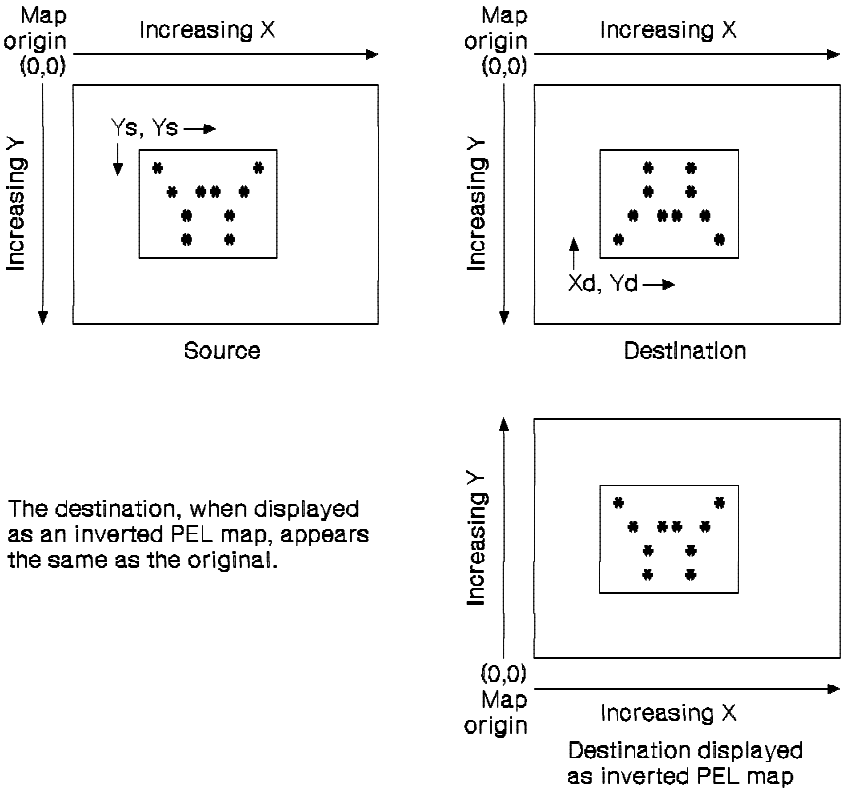


Figure 3-118. Inverting PxBlt

An inverting PxBlt is set up in the same manner as a standard PxBlt with the following notes:

- The PxBlt direction set applies to the updating of the source X and Y addresses.
- The destination Y pointer must be programmed to the opposite (in Y) corner of the destination rectangle.
- The Function field in the PEL Operation register must be set to inverting PxBlt as opposed to PxBlt.

See “Overlapping PxBlts” on page 3-260 for details on PxBlts when the source and destinations overlap.

Area Fill

The following steps are required to perform an area fill operation without a user pattern:

1. Draw the closed outline of the area to be filled using the area boundary drawing mode. Typically, a unique, off-screen PEL map would be defined to draw the area boundary into. This PEL map must be initialized to contain 0-value PELs before the boundary is drawn. This PEL map must be in a 1 bit-per-PEL format.
2. Designate the PEL map where the area boundary was drawn as the pattern map.
3. Specify the desired destination.
4. Select the desired foreground mix and source.
5. Specify the background mix as Destination (code 5).
6. Specify the operation direction as any direction with X increasing (Codes 0 or 1, 2 or 3), because the pattern data is scanned from left to right. Selection of a negative X direction code for area fill operations results in fill errors.
7. Initiate the area fill operation.

During the area fill operation, the coprocessor applies a filling function to the pattern PELs before they are used to select background and foreground sources and mixes in the usual way. The filling function modifies the pattern PELs horizontally line by line. It scans the pattern from left to right, and when encountering the first foreground (1) PEL, sets all subsequent PELs to foreground (1) until the next foreground PEL is encountered.

This process is illustrated in Figure 3-119.

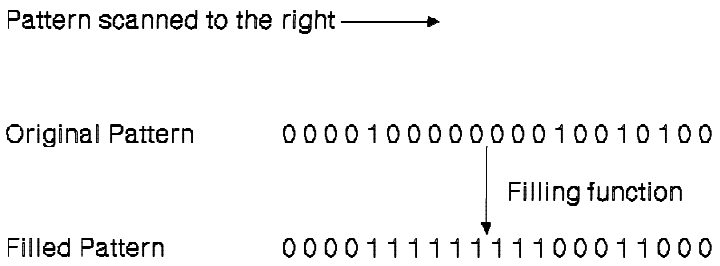


Figure 3-119. Pattern Filling

The filled pattern is generated internally to the coprocessor. It is then used exactly as the pattern in any normal operation, with foreground (1) PELs selecting foreground source and mix, and background (0) PELs selecting background source and mix. During area fill operations, it is required to fill the specified area and leave all other PELs unchanged. This is why the background mix was specified to be destination. Figure 3-103 on page 3-92 shows the position of the pattern-filling circuitry in the coprocessor data flow.

Area fill operations that require a pattern fill must be performed in two stages. This is because area fill PxBlt operations use the pattern map to perform the area fill function and cannot include a user pattern in a single operation. However, by first combining the contents of the mask map with a mask of the filled area, a full pattern PxBlt of an area can be achieved as follows:

1. Both the pattern map and the destination map must be defined as the map containing the previously drawn area boundary. The source map must be defined as the map that would normally supply the mask map for the operation. The mask map facility must be disabled. An area fill PxBlt must be performed with the following conditions:

   - Foreground source = source PEL map
   - Foreground mix = Source (code 3)
   - Background source = background color
   - Background color = 0
   - Background mix = Source (code 3).

   Note: All the maps in this operation must be 1 bit-per-PEL.

   This operation combines the mask data for the pattern area fill with a mask of the filled area.

2. A second non-area fill PxBlt must be performed with the combined mask generated in step 1 defined as the mask map. All other maps can be used as normal with no restrictions.

Logical and Arithmetic Functions

During an operation in the coprocessor, source data is combined with destination data, under the control of pattern data, and the result is written back to the destination. Mask data can be included in the operation to selectively inhibit updating of destination data.

Source data can be either foreground source or background source on a PEL-by-PEL basis. The foreground source is combined with the destination using the foreground mix; the background source is combined with the destination using the background mix. The pattern determines if the source and mix are foreground or background for a particular PEL. If the pattern PEL is 1, source and mix are foreground; if it is 0, they are background.

The foreground and background sources can each be either a fixed color over the whole operation or PEL data taken from the source PEL map. The background source and foreground source bits in the PEL Operations register determine whether fixed colors or source PEL map data is used in an operation. The fixed color that is used as the foreground source is called the foreground color, and is stored in the Foreground Color register. The fixed color that is used as the background source is called the background color, and is stored in the Background Color register.

The possible combinations of source, destination and pattern are shown below:

- Pattern PEL = 1 (foreground source)
  - New destination PEL **=** old destination PEL **Fgd OP** Foreground color
  - New destination PEL **=** old destination PEL **Fgd OP** PEL map source.
- Pattern PEL = 0 (background source)
  - New destination PEL **=** old destination PEL **Bgd OP** Background color
  - New destination PEL **=** old destination PEL **Bgd OP** PEL map source.

Fgd OP is the logical or arithmetic function specified in the Foreground Mix register. **Bgd OP** is the logical or arithmetic function specified in the Background Mix register.

These operations can be inhibited by the contents of the mask map. If the mask PEL is 0, the destination PEL is not modified. If the mask PEL is 1, the selected operation is applied to the destination PEL.

Mixes

The foreground and background mixes provided by the XGA are independent. The XGA provides all logical mixes of two operands and six arithmetic mixes. The mixes provided are as follows:

Code (hex)	Function
00	Zeros
01	Source AND Destination
02	Source AND NOT Destination
03	Source
04	NOT Source AND Destination
05	Destination
06	Source XOR Destination
07	Source OR Destination
08	NOT Source AND NOT Destination
09	Source XOR NOT Destination
0A	NOT Destination
0B	Source OR NOT Destination
0C	NOT Source
0D	NOT Source OR Destination
0E	NOT Source OR NOT Destination
0F	Ones
10	Maximum
11	Minimum
12	Add With Saturate
13	Subtract (Destination – Source) With Saturate
14	Subtract (Source – Destination) With Saturate
15	Average

Note: Mix codes hex 16 to FF are reserved.

Figure 3-120. Foreground and Background Mixes

Saturate means that if the result of an arithmetic operation is greater than all 1's, the final result remains all 1's. If the result of an arithmetic operation is less than 0, the final result remains at 0.

Breaking the Coprocessor Carry Chain

To limit the operation of the coprocessor to certain bits in a PEL (for example, to perform an operation on both the upper and lower 4 bits of an 8-bit PEL independently), it is not desirable for the arithmetic operations to propagate a carry from one group of bits in the PEL to the next. One solution is to use the XGA PEL bit mask to ensure only one component of the PEL is processed at a time. The disadvantage of this technique is that the operation must be repeated once for each component in the PEL.

The XGA provides an alternative mechanism that allows PELs with component fields to process correctly in one pass. A carry chain mask can be specified that determines how carry bits are propagated in the coprocessor. By loading the appropriate mask in the Carry Chain Mask register before performing an operation
! involving an arithmetic operation , the PEL is effectively divided into independent fields. The mask prevents the coprocessor carry being propagated across the field boundaries.

Each bit in the mask enables or disables the propagation of the carry from the corresponding bit in the coprocessor to its more significant neighbor. The mask is n – 1 bits wide for a PEL n bits wide, and the carry from the most significant bit of the coprocessor is not propagated.

An example for a carry chain mask for an 8-bit PEL with two 4-bit fields follows:

7	6	5	4	3	2	1	0
-	1	1	1	0	1	1	1

Figure 3-121. Carry Chain Mask for an 8-bit PEL

Bits outside the required mask size for a given PEL size need not be written in the register.

Generating the Pattern from the Source

Pattern data for an operation can be supplied by PEL maps A, B, or C, or it can be fixed to 1 (foreground source) throughout the operation. Pattern data can also be internally generated by the coprocessor from source PEL map data. A comparison operation is performed on each source PEL and the pattern data is generated depending on the result.

The comparison operation compares the source PEL to 0. For any source PEL with a value of 0, a 0 (background) pattern PEL is generated. For any nonzero source PEL, a 1 (foreground) pattern PEL is generated. The internally generated pattern is then used to select between foreground and background sources, and mixes in the usual way. When the pattern is internally generated, the coprocessor ignores the pattern PEL map contents.

This capability allows the background source data and mix to be forced for all 0 value PELs in the source. In particular, it provides a transparency function, where a multibit character can be drawn onto a destination with the destination data showing through any 0 (black) PEL in the source character definition.

Color Expansion

If the source PELs for an operation have fewer bits per PEL than the destination PELs, the source PELs must be expanded to the same size as those in the destination before they are combined. This process is referred to as color expansion.

The major use of color expansion is to draw 1-bit-per-PEL character sets on n-bits-per-PEL destinations. The coprocessor performs this function in hardware, but does not have a color expansion look-up table. Instead, the 1-bit-per-PEL character map must be defined as the pattern map. The PEL Operation register must be programmed to use the Foreground Color and Background Color registers, not the source map. The Foreground Color and Background Color registers then act as a two-entry color expansion look-up table, and the character map is expanded to the number of bits per PEL in the destination.

PEL Bit Masking

The PEL bit mask allows any combination of bits in a destination PEL to be protected from update (being written). A mask value must be loaded in the PEL Bit Mask register to enable or disable updating of PEL bits selectively as required.

This mask is the same as the plane mask in subsystems that are plane oriented, as opposed to packed-PEL.

When the destination bits-per-PEL is less than 8 bits, only the low order bits of the PEL Bit Mask register are significant.

A bit that is not write enabled is prevented from affecting arithmetic or compare operations. In effect, masked bits are completely excluded from the operation or comparison.

Color Compare

The value that the destination PELs are compared with is stored in the Destination Color Compare Value register. The Destination Color Compare Condition register indicates the condition when the destination update is inhibited. The possible conditions are as follows:

Condition Code	Condition
0	Always True (disable update)
1	Destination Data > Color Compare Value
2	Destination Data = Color Compare Value
3	Destination Data < Color Compare Value
4	Always False (enable update)
5	Destination Data > = Color Compare Value
6	Destination Data < > Color Compare Value
7	Destination Data < = Color Compare Value

Figure 3-122. Color Compare Conditions

Note: A comparison result of true prevents update to the destination.

Controlling Coprocessor Operations

Starting a Coprocessor Operation

Coprocessor operations are started by writing the most significant byte of the PEL Operations register. One exception to this is the draw and step function. For details, see “Draw and Step” on page 3-108.

Suspending a Coprocessor Operation

Coprocessor operations can be suspended before they have completed. The state of the coprocessor, including internal register contents, can then be read rapidly to allow task state saving. A previous task can be restored through the same data port and the restored operation can be restarted.

The suspend operation bits in the Coprocessor Control register are used to suspend and restart coprocessor operations.

If a coprocessor operation is suspended, a terminate operation is required before starting a new coprocessor operation.

Terminating a Coprocessor Operation

Operations can be terminated before they have completed. The state of the coprocessor registers that are updated as the operation proceeds is undefined after the operation is terminated and their contents must not be relied upon. “Coprocessor Registers” on page 3-132 details the registers that are updated as an operation proceeds.

The terminate operation bit in the Coprocessor Control register is used to terminate operations.

Coprocessor Operation Completion

There are two methods for the system microprocessor to detect the completion of a coprocessor operation:

- Receive an operation-complete interrupt from the XGA.
- Poll the Coprocessor Busy bit in the Coprocessor Control register or (on the XGA-NI Subsystem) poll the Auxiliary Coprocessor Busy bit in the Auxilliary Coprocessor Status register.

Coprocessor-Operation-Complete Interrupt

The coprocessor provides an operation-complete interrupt that can interrupt the system on completion of an operation. The interrupt is enabled by a bit in the Interrupt Enable register and its status is indicated by a bit in the Interrupt Status register. See “Interrupt Enable Register (Address 21x4)” on page 3-39 and “Interrupt Status Register (Address 21x5)” on page 3-41 for bit locations.

Regardless of the state of the operation-complete interrupt enable bit, the status bit is always set to 1 on completion of an operation. The application must ensure that this bit is reset before starting an operation. This is done by writing a 1 back to the status bit.

If the interrupt enable bit is 1, the completion of an operation not only sets the interrupt status bit, but also causes an interrupt. The system microprocessor must reset the interrupt by writing a 1 back to the status bit after servicing the interrupt.

Coprocessor Busy Bit

The coprocessor busy bit in the Coprocessor Control register, or (on the XGA-NI Subsystem) the Auxiliary Coprocessor Busy bit in the Auxilliary Coprocessor Status register, indicates if the coprocessor is executing an operation. It is set to 1 by the hardware when the coprocessor is executing an operation and reset to 0 when the operation completes. Applications can read this bit to determine if the coprocessor is busy. See “Waiting for Hardware Not Busy” on page 3-259 for more information.

Accesses to the Coprocessor During an Operation

When the coprocessor is executing an operation, the system processor can only perform read accesses to the coprocessor registers. Write accesses are not permitted because they could corrupt operation data.

If the system processor attempts to write data to the coprocessor registers during an operation, the coprocessor allows the access to complete, but the executing operation may be corrupted. The coprocessor interrupts the system microprocessor to indicate a write access occurred during an active operation and the operation may have been corrupted. This interrupt is called the coprocessor-access-rejected interrupt. An enable bit is in the Interrupt Enable register and a status bit is in the Interrupt Status register. See “Interrupt Enable Register (Address 21x4)” on page 3-39 and “Interrupt Status Register (Address 21x5)” on page 3-41 for bit locations.

There is one exception to this rule. The Coprocessor Control register can be written during an operation without corrupting the operation. See “Coprocessor Control Register (Offset 11)” on page 3-136 for more information.

Coprocessor State Save/Restore

When operating in a multitasking environment it is necessary to save and restore the state of the display hardware when switching tasks.

Sometimes a task switch is required when the coprocessor is in the course of executing an operation. Not only the contents of registers visible to the system microprocessor, but also contents of internal registers (the state of the coprocessor) must be saved and, later, restored. The coprocessor has special hardware that lets it suspend the execution of an operation and efficiently save and restore task states.

Suspending Coprocessor Operations

At any time during the execution of a coprocessor operation, the operation can be suspended by writing to a bit in the Coprocessor Control register. Any executing memory cycle is completed before the coprocessor suspends the operation. The system can then save and restore the coprocessor contents and restart the restored operation by clearing the bit in the Coprocessor Control register.

If a coprocessor operation is suspended, a terminate operation is required before starting a new coprocessor operation.

Save/Restore Mechanism

The coprocessor provides two special 32-bit save/restore data ports. All the coprocessor state data passes through these ports when the state is being saved or restored. The number of doublewords read or written is determined by two read-only registers (State Length registers A and B). The amount of data saved or restored is less than 1KB. State-saving software must perform string I/O read instructions, reading data from the two save/restore data ports in turn. The coprocessor hardware automatically provides successive doublewords of data on successive reads. After the state has been saved, the coprocessor is in a reset state.

If a coprocessor operation is suspended, a terminate operation is required before starting a new coprocessor operation.

Restoring the state of the coprocessor uses a similar process. State data must be moved back into the coprocessor using string I/O write instructions. The state data must be written back into the coprocessor in the same order as it was read (first out, first in).

The exact number of doublewords specified in the State Length registers must be read or written when saving or restoring the coprocessor state. Failure to do this leaves the coprocessor in an indeterminate state.

Coprocessor Registers

| “XGA Adapter Identification, Location and XGA Mode Setting” on
| page 3-192 provides details of locating and using these registers.

The XGA coprocessor supports two register interface formats. The type of interface required (Intel or Motorola) is set when selecting Extended Graphics mode in the Operating Mode register.

The difference between Intel and Motorola formats is that, with two exceptions, the bytes within each 4 bytes of register space are reversed (byte 0 becomes byte 3). The two exceptions are the Direction Steps register and the PEL Operations register. The bytes within these registers are not reversed because the existing byte order is required by the operation being performed.

Most of the coprocessor registers are not directly readable by the system microprocessor. Registers that cannot be read directly can be read indirectly using the coprocessor state save and restore mechanism. See “Save/Restore Mechanism” on page 3-131 for more information. Only the following registers are readable directly by the system microprocessor:

- State Save/Restore Data Ports register
- State Length registers
- Coprocessor Control register
- Virtual Memory Control register
- Virtual Memory Interrupt Status register
- Current Virtual Address register
- Bresenham Error Term E register
- Source X Address and Source Y Address registers
- Pattern X Address and Pattern Y Address registers
- Destination X Address and Destination Y Address registers.

The contents of most coprocessor registers are not changed during a coprocessor operation and therefore do not need to have their contents reloaded before starting another similar operation. The registers with contents that change during an operation are:

Bresenham Error Term E
: The error term is updated throughout line draw operations.

Source X Address and Source Y Address
: Any operation that uses the source PEL map updates these pointers.

Pattern X Address and Pattern Y Address

The pattern map X and Y pointers are updated during any operation that does not have the Pattern field in the PEL Operation register set to foreground.

Destination X Address and Destination Y Address

The destination map X and Y pointers are updated during all operations.

The following figures show the coprocessor register space in Intel and Motorola formats.

Coprocessor Address Space				
Byte 3	Byte 2	Byte 1	Byte 0	
Page Directory Base Address				0
Current Virtual Address				4
		Auxiliary Coprocessor Status		8
		State B length	State A length	C
	PEL Map Index	Coprocessor Control		10
PEL Map n Base Pointer				14
PEL Map n Height		PEL Map n Width		18
			PEL Map n Format	1C
		Bresenham Error Term		20
		Bresenham K1		24
		Bresenham K2		28
Direction Steps				2C
				30
				34
				38
				3C
				40
				44
	Destination Color Compare Condition	Background Mix	Foreground Mix	48
Destination Color Compare Value				4C
PEL Bit Mask				50
Carry Chain Mask				54
Foreground Color Register				58
Background Color Register				5C
Operation Dimension 2		Operation Dimension 1		60
				64
				68
Mask Map Origin Y Offset		Mask Map Origin X Offset		6C
Source Map Y Address		Source Map X Address		70
Pattern Map Y Address		Pattern Map X Address		74
Destination Map Y Address		Destination Map X Address		78
PEL Operation				7C

Figure 3-123. XGA Coprocessor Register Space, Intel Format

Note: All unused and undefined offsets are reserved.

Coprocessor Address Space				
Byte 0	Byte 1	Byte 2	Byte 3	
Page Directory Base Address				0
Current Virtual Address				4
		Auxiliary Coprocessor Status		8
		State B length	State A length	C
	PEL Map Index	Coprocessor Control		10
PEL Map n Base Pointer				14
PEL Map n Height		PEL Map n Width		18
			PEL Map n Format	1C
		Bresenham Error Term		20
		Bresenham K1		24
		Bresenham K2		28
Direction Steps				2C
				30
				34
				38
				3C
				40
				44
	Destination Color Compare Condition	Background Mix	Foreground Mix	48
Destination Color Compare Value				4C
PEL Bit Mask				50
Carry Chain Mask				54
Foreground Color Register				58
Background Color Register				5C
Operation Dimension 2		Operation Dimension 1		60
				64
				68
Mask Map Origin Y Offset		Mask Map Origin X Offset		6C
Source Map Y Address		Source Map X Address		70
Pattern Map Y Address		Pattern Map X Address		74
Destination Map Y Address		Destination Map X Address		78
PEL Operation				7C

Figure 3-124. XGA Coprocessor Register Space, Motorola Format

Note: All unused and undefined offsets are reserved.

Register Usage Guidelines

Unless otherwise stated, the following are guidelines to be used when accessing the coprocessor registers:

- Special reserved register bits must be used as follows:
  - Register bits marked with '– ' must be set to 0. These bits are undefined when read and should be masked off if the contents of the register is to be tested.
  - Register bits marked with '#' are reserved and the state of these bits must be preserved. When writing the register, read the register first and change only the bits that must be changed.
- Unspecified registers or registers marked as reserved in the XGA coprocessor address space are reserved. They must not be written to or read from.
- During a read, the values returned from write-only registers are reserved and unspecified.
- The contents of read-only registers must not be modified.
- Counters must not be relied upon to wrap from the high value to the low value.
- Register fields defined with valid ranges must not be loaded with a value outside the specified range.
- Register field values defined as reserved must not be written.

The following sections describe the coprocessor registers in detail. Unless stated otherwise, the register definitions are in Intel format.

Virtual Memory Registers

The XGA coprocessor virtual memory implementation is given in "Virtual Memory Description" on page 3-177.

State Save/Restore Registers

The following registers allow the internal state of the coprocessor to be saved and restored. “Coprocessor State Save/Restore” on page 3-129 describes this mechanism.

Auxiliary Coprocessor Status Register (Offset 09)

This read only register has an offset of hex 09. It is only available for use on the XGA-NI Subsystem. The Auxiliary Coprocessor Status register indicates if the coprocessor is currently executing an operation.

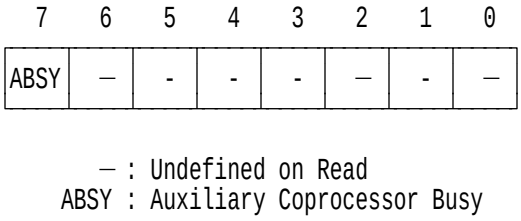


Figure 3-125. Auxiliary Coprocessor Status Register, Offset Hex 09

The register fields are defined as follows:

ABSY When the Auxiliary Coprocessor Busy field (bit 7) is read as 1, the coprocessor is currently executing an operation. When read as 0, the coprocessor is idle.

This bit provides the same information as the BSY field of “Coprocessor Control Register (Offset 11)” but it should be used in preference to it when operating on a XGA-NI Subsystem as it may increase coprocessor performance. See “Waiting for Hardware Not Busy” on page 3-259

Coprocessor Control Register (Offset 11)

This read/write register has an offset of hex 11. The Coprocessor Control register indicates if the coprocessor is currently executing an operation. The current coprocessor operation can be terminated or suspended by writing to this register.

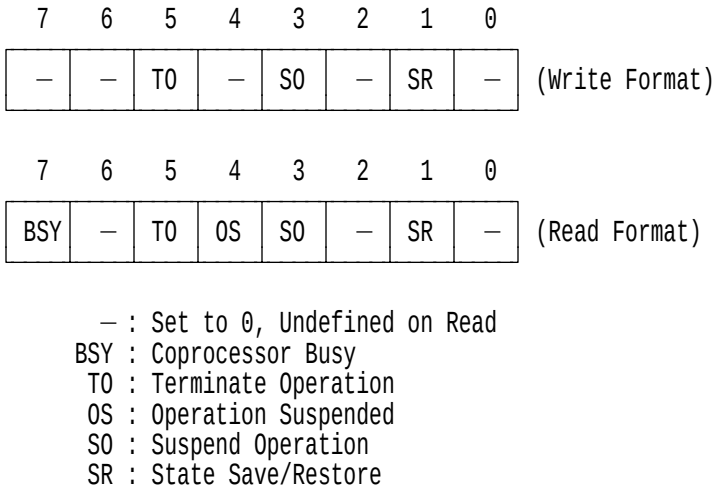


Figure 3-126. Coprocessor Control Register, Offset Hex 11

The register fields are defined as follows:

BSY When the Coprocessor Busy field (bit 7) is read as 1, the coprocessor is currently executing an operation. When read as 0, the coprocessor is idle. On a XGA-NI subsystem "Auxiliary Coprocessor Status Register (Offset 09)" should be used in preference to this field. See "Waiting for Hardware Not Busy" on page 3-259

TO Coprocessor operations can be terminated by writing a 1 to the Terminate Operation field (bit 5). The application must ensure that the operation has terminated before proceeding. Termination is ensured by waiting for the operation-complete interrupt (if enabled), or by reading the Coprocessor Busy field until the coprocessor goes not busy (bit 7 = 0).

After the coprocessor has terminated the operation, it automatically sets the Terminate Operation field to 0. The coprocessor is returned to its initial power-on state, with coprocessor interrupts masked off and certain other register bits reset. All registers must be assumed invalid and must be reprogrammed before another operation is initiated.

OS The Operation Suspended field (bit 4) is set to 1 by the coprocessor when it has suspended the operation. This bit must be read by the system microprocessor to ensure that an operation has been suspended before saving/restoring is started.

SO When the Suspend Operations field (bit 3) is set to 1, coprocessor operations are suspended.

When set to 0, the suspended processor operations are restarted. This must be done to restart a restored operation after a task switch. When the operation restarts, the coprocessor resets the Operations Suspended field to 0.

Suspending an operation flushes the translate look-aside buffer.

If a coprocessor operation is suspended, a terminate operation is required before starting a new coprocessor operation.

SR The State Save/Restore field (bit 1) selects whether to save or restore the coprocessor state. When set to 0, a state restore can be performed; when set to 1, a state save can be performed. The Coprocessor Control register must be written with the Suspend Operation field set and the Save/Restore field appropriately set before each state save or state restore.

State Length Registers (Offset C and D)

These read-only registers have offsets of hex C and D. *Do not write to* these registers. The State Length registers return the length, in doublewords, of parts A and B of the coprocessor state for save and restore.

Save/Restore Data Ports Register (I/O Index C and D)

These read/write registers have I/O indexes of hex C and D. The Save/Restore registers are directly mapped to I/O address space and do not appear in the coprocessor register summary. However, they are coprocessor registers and are described here.

These registers are used to save and restore the two parts, A and B, of the internal state of the coprocessor. After a state save or restore is initiated, string I/O reads or writes must be executed from or to these registers. The data can be read or written using any combination of byte, word, or doubleword accesses, provided that the exact number of doublewords specified in the State Length registers is read or written. Failure to read or write the correct amount of data leaves the coprocessor in an undefined state.

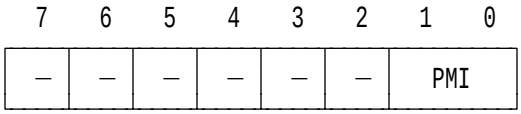
Data must be written back to this port in the order it was read (first out, first in).

PEL Interface Registers

The following is a detailed description of the coprocessor PEL interface registers.

PEL Map Index Register (Offset 12)

This write-only register has an offset of hex 12.



– : Set to 0
PMI : PEL Map Index

Figure 3-127. PEL Map Index Register, Offset Hex 12

The register field is defined as follows:

PMI The PEL Map Index field (bits 1, 0) selects which PEL map registers will be used.

Each PEL map used in the XGA is described by four registers, as follows:

- The PEL Map n Base Pointer register
- The PEL Map n Width register
- The PEL Map n Height register
- The PEL Map n Format register.

Each PEL map has its own copy of these registers, so there are four copies of these registers in the XGA, one each for:

- The mask map
- PEL map A
- PEL map B
- PEL map C.

Only one of these banks of PEL map registers is visible to the system microprocessor at any time. The PEL Map Index register is used to select the maps the registers apply to. The encoding of the PEL Map Index register is shown in the following figure.

PMI Field (binary)	PEL Map Register
0 0	Mask Map
0 1	PEL Map A
1 0	PEL Map B
1 1	PEL Map C

Figure 3-128. PEL Map Index

Before loading the PEL map base pointer, width, height, and format for a particular map, the PEL Map Index register must be set up to point to the registers of the required map. For example, to set up the register for map B, first load the PEL map index with binary 10.

PEL Map n Base Pointer Register (Offset 14)

This write-only register has an offset of hex 14. The n is selected using the “PEL Map Index Register (Offset 12)” on page 3-140.

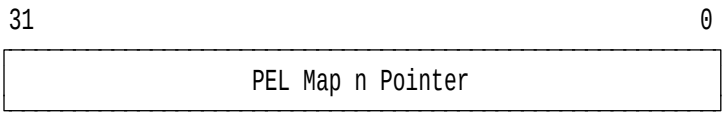


Figure 3-129. PEL Map n Base Pointer Register, Offset Hex 14

The PEL Map n Pointer register (bits 31– 0) specifies the byte address in memory of the start of a PEL map. If virtual address mode is enabled, this address is a virtual address, otherwise it is a physical address.

PEL Map n Width Register (Offset 18)

This write-only register has an offset of hex 18. The PEL Map n Width register can be loaded with any value in the range (0 to 4095). The n is selected using the “PEL Map Index Register (Offset 12)” on page 3-140.

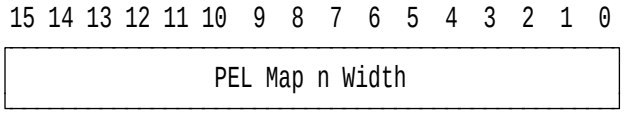


Figure 3-130. PEL Map n Width Register, Offset Hex 18

The PEL Map n Width register (bits 15– 0) specifies the width of a PEL map. The width of a PEL map is measured in PELs, that is, independent of the number of bits-per-PEL.

Widths are used during address stepping to specify the width of the PEL map. Steps with a Y direction component are achieved by the hardware adding or subtracting the width ± 0/1.

The PEL map width is also used for wrapping the source and pattern maps, or to implement the fixed scissor boundary around the destination map.

The value loaded in the width register must be 1 less than the bit map width. For a bit map that is 1024 PELs wide, the width register must be loaded with 1023 (hex 03FF).

PEL Map n Height Register (Offset 1A)

This write-only register has an offset of hex 1A. The PEL Map n Height register can be loaded with any value in the range (0 to 4095). The n is selected using the “PEL Map Index Register (Offset 12)” on page 3-140.

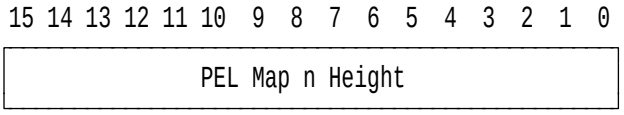


Figure 3-131. PEL Map n Height Register, Offset Hex 1A

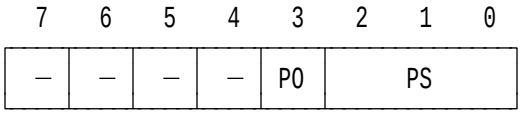
The PEL Map n Height register (bits 15– 0) specifies the height of a PEL map. The height of the PEL map is measured in PELs, that is, independent of the number of bits-per-PEL.

The PEL map height is used for wrapping the source and pattern maps, or to implement the fixed scissor boundary around the destination map.

The value loaded in the height register must be one less than the PEL map height. For a bit map that is 768 PELs high, the height register must be loaded with 767 (hex 02FF).

PEL Map n Format Register (Offset 1C)

This write-only register has an offset of hex 1C. The n is selected using the “PEL Map Index Register (Offset 12)” on page 3-140.



– : Set to 0
PO : PEL Order
PS : PEL Size

Figure 3-132. PEL Map n Format Register, Offset Hex 1C

The register fields are defined as follows:

PO The PEL Order field (bit 3) selects the format for the memory-to-screen mapping. When set to 0, the PEL map is Intel-ordered; when set to 1, the PEL map is Motorola-ordered. “PEL Formats” on page 3-93 describes the difference in formats.

PS The PEL Size field (bits 2– 0) specifies the number of bits-per-PEL in the PEL map. PEL maps occupied by the source or destination map can be 1, 2, 4, or 8 bits-per-PEL on the XGA Subsystem and 1, 2, 4, 8, or 16 bits=per-PEL on the XGA-NI subsystem. The PEL map occupied by the pattern map must be 1 bit-per-PEL. Programming the pattern to be taken from a PEL map that does not contain 1-bit PELs produces undefined results. The PEL size field definitions are shown in the following figure.

PS Field (binary)	PEL Size
0 0 0	1 Bit
0 0 1	2 Bits
0 1 0	4 Bits
0 1 1	8 Bits
1 0 0	16 Bits On XGA-NI Subsystem Only. Reserved on XGA Subsystem
1 0 1	Reserved
1 1 0	Reserved
1 1 1	Reserved

| *Figure 3-133. PEL Size Value Assignments*

PEL Maps A, B, and C

PEL maps A, B, and C are all described by similar registers. The different maps are merely three instances of PEL maps that can have different locations in memory, sizes, and formats.

The pattern map used by the XGA must be 1 bit-per-PEL. The pattern map must reside in a PEL map that is 1 bit-per-PEL. Failure to do this produces undefined results.

Mask Map

The mask map has a base pointer, width, and height that are similar to those of PEL maps A, B, and C.

The Mask Map Format register differs from maps A, B, and C in that only the Motorola/Intel format bit of the mask map is programmable. This register bit operates the same as the bit for maps A, B, and C. The number of bits-per-PEL is assumed to be 1 bit-per-PEL. The bits-per-PEL must always be set to 1 bit-per-PEL to ensure future compatibility.

Bresenham Error Term E Register (Offset 20)

This read/write register has an offset of hex 20.

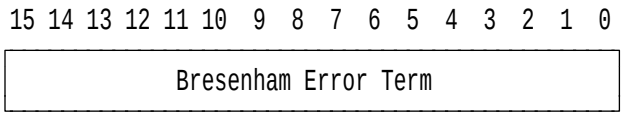


Figure 3-134. Bresenham Error Term E Register, Offset Hex 20

The Bresenham Error Term register (bits 15– 0) specifies the Bresenham error term for the draw line function. The error term value is a signed quantity, calculated as ((2 × deltaY) – deltaX) after normalization to the first octant.

This register must be written as a 16-bit sign-extended twos complement number in the range (– 8192 to 8191).

Bresenham Constant K1 Register (Offset 24)

This write-only register has an offset of hex 24.

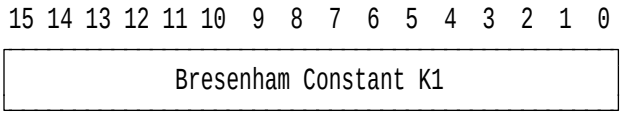


Figure 3-135. Bresenham Constant K1 Register, Offset Hex 24

The Bresenham Constant K1 register (bits 15– 0) specifies the Bresenham constant, K1, for the draw line function. The K1 value is a signed quantity, calculated as (2 × deltaY) after normalization to the first octant.

This register must be written as a 16-bit sign-extended twos complement number in the range (– 8192 to 8191).

Bresenham Constant K2 Register (Offset 28)

This write-only register has an offset of hex 28.

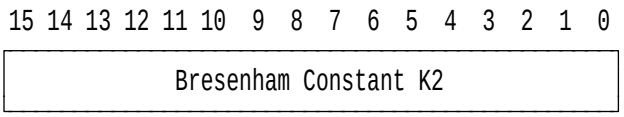


Figure 3-136. Bresenham Constant K2 Register, Offset Hex 28

The Bresenham Constant K2 register (bits 15– 0) specifies the Bresenham constant, K2, for the draw line function. The K2 value is a signed quantity, calculated as (2 × (deltaY – deltaX)) after normalization to the first octant.

This register must be written as a 16-bit sign-extended twos complement number in the range (– 8192 to 8191).

Direction Steps Register (Offset 2C)

This write-only register has an offset of hex 2C.

The byte order of this register is independent of whether the Intel or Motorola register interface is enabled. The following figure shows the Intel and Motorola views of the Direction Steps register.

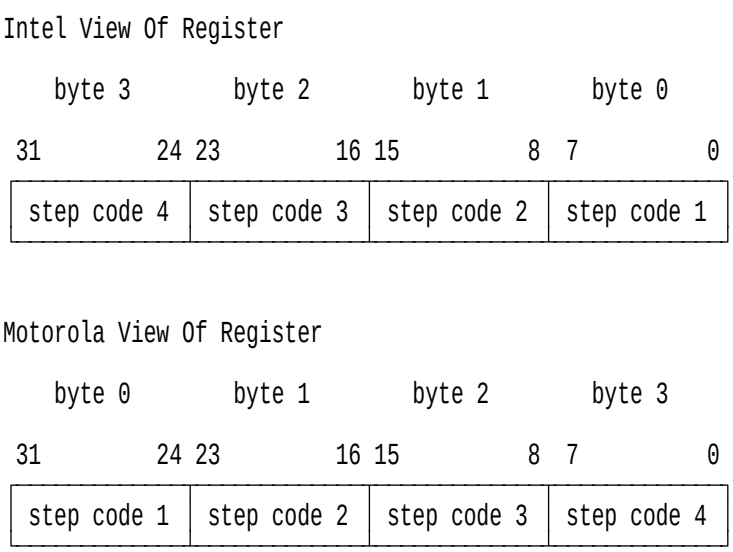


Figure 3-137. Direction Steps Register, Offset Hex 2C

This register is used to specify up to four draw and step codes to the coprocessor, and to initiate a draw and step operation.

Writing data to byte 3 of this register initiates a draw and step operation. A draw and step operation can be initiated by a single 32-bit access, by two 16-bit accesses where bytes 2 and 3 are written last, or by four 1-byte accesses where byte 3 is written last. If multiple draw and step operations are required with the same draw and step codes, the operation can be initiated by writing to byte 3.

Before initiating a draw and step operation, the PEL Operation register must be configured to set up the data path and flags for

the draw and step operation. See “Draw and Step” on page 3-108 for full details.

Foreground Mix Register (Offset 48)

This write-only register has an offset of hex 48.

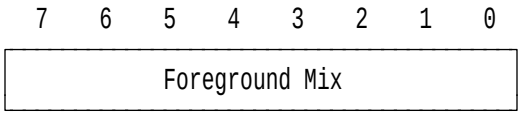


Figure 3-138. Foreground Mix Register, Offset Hex 48

The Foreground Mix register (bits 7– 0) holds the foreground mix value that specifies a logic or arithmetic function to be performed between the destination and function 1 second operand PELs, during an operation where the pattern PEL value is 1.

See “Logical and Arithmetic Functions” on page 3-122 for details and mix functions available.

Background Mix Register (Offset 49)

This write-only register has an offset of hex 49.

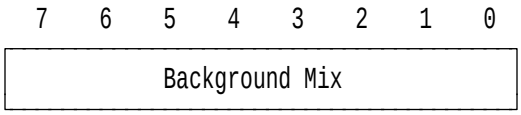


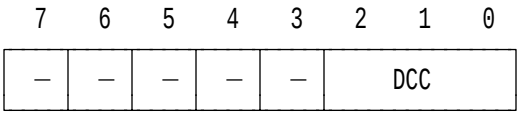
Figure 3-139. Background Mix Register, Offset Hex 49

The Background Mix register (bits 7– 0) holds the background mix value that specifies a logic or arithmetic function to be performed between the destination and function 0 second operand PELs during an operation where the pattern PEL value is 0.

See “Logical and Arithmetic Functions” on page 3-122 for details and mix functions available.

Destination Color Compare Condition Register (Offset 4A)

This write-only register has an offset of hex 4A.



– : Set to 0
DCC : Destination Color Compare Condition

Figure 3-140. Destination Color Compare Condition Reg, Offset Hex 4A

The register field is defined as follows:

DCC The Destination Color Compare Condition field (bits 2– 0) specifies the destination color compare condition when the destination update is inhibited.

The DCC field definitions are shown in the following figure.

DCC Field (binary)	Destination Color Compare Condition
0 0 0	Always True (disable update)
0 0 1	Destination > Color Compare Value
0 1 0	Destination = Color Compare Value
0 1 1	Destination < Color Compare Value
1 0 0	Always False (enable update)
1 0 1	Destination > = Color Compare Value
1 1 0	Destination < > Color Compare Value
1 1 1	Destination < = Color Compare Value

Figure 3-141. Destination Color Compare Condition Bit Definition

Destination Color Compare Value Register (Offset 4C)

This write-only register has an offset of hex 4C.

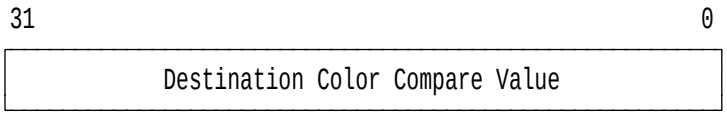


Figure 3-142. Destination Color Compare Value Register, Offset Hex 4C

The Destination Color Compare Value register (bits 31– 0) contains the comparison value for the destination PELs to be compared when color compare is enabled. Only the corresponding number of bits-per-PEL in the destination are required in this register (for example, if the destination is 4 bits-per-PEL, only the 4 low-order bits of this register are used). The bits of this register that are more significant than the number of bits-per-PEL need not be written.

See “Color Compare” on page 3-126 for details of the color compare function.

PEL Bit Mask (Plane Mask) Register (Offset 50)

This write-only register has an offset of hex 50.

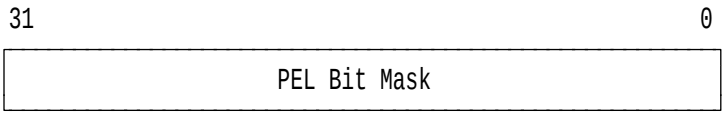


Figure 3-143. PEL Bit Mask (Plane Mask) Register, Offset Hex 50

The PEL Bit Mask register (bits 31– 0) determines the bits within each PEL that are subject to update by the coprocessor. A 1 means the corresponding bit is enabled for updates. A 0 means the corresponding bit is not updated.

A bit that is not write enabled is prevented from affecting either arithmetic operations or the destination color compare comparison, so masked bits are excluded from the operation or comparison. Only the corresponding number of bits-per-PEL in the destination are required in this register (for example, if the destination is 4 bits-per-PEL, only the 4 low-order bits of this register are used). The bits of this register that are more significant than the number of bits-per-PEL need not be written.

See “PEL Bit Masking” on page 3-126 for details of the PEL bit mask function.

Carry Chain Mask Register (Offset 54)

This write-only register has an offset of hex 54.

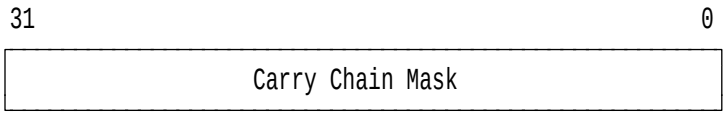


Figure 3-144. Carry Chain Mask Field Register, Offset Hex 54

The Carry Chain Mask register (bits 31– 0) contains a mask up to 31 bits wide. The mask is used to specify how the carry chain of the coprocessor is propagated when performing arithmetic update mixes and color compare operations.

A 0 in the mask means that the carry out of this bit-position of the coprocessor is not to be propagated to the next significant bit-position. A 1 in the mask means that propagation is to take place. The PEL value can be split into sections within the PEL.

Only the corresponding number of bits-per-PEL in the destination are required in this register (for example, if the destination is 4 bits-per-PEL, only the 4 low-order bits of this register are used). The bits of this register that are more significant than the number of bits-per-PEL, need not be written. There is no carry out of the most-significant bit of the PEL irrespective of the setting of the corresponding carry chain mask bit.

See “Breaking the Coprocessor Carry Chain” on page 3-124 for details on the carry chain function.

Foreground Color Register (Offset 58)

This write-only register has an offset of hex 58.

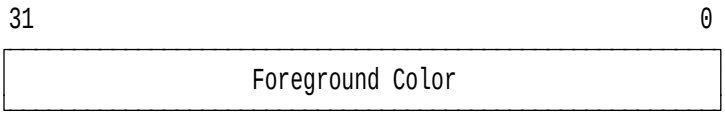


Figure 3-145. Foreground Color Register, Offset Hex 58

The Foreground Color register (bits 31– 0) holds the foreground color to be used during coprocessor operations. The foreground color can be specified as the foreground source by setting up the appropriate field in the PEL Operations register.

Only the corresponding number of bits-per-PEL in the destination are required in this register (for example, if the destination is 4 bits-per-PEL, only the 4 low-order bits of this register are used). The bits of this register that are more significant than the number of bits-per-PEL need not be written.

Background Color Register (Offset 5C)

This write-only register has an offset of hex 5C.

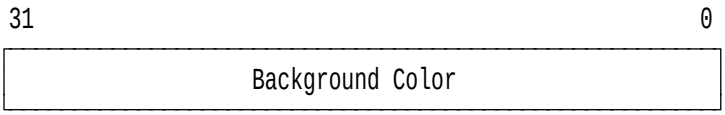


Figure 3-146. background Color Register, Offset Hex 5C

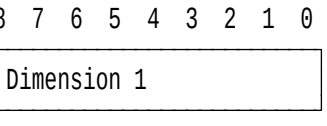
The Background Color register (bits 31– 0) holds the background color to be used during coprocessor operations. The background color can be specified as the background source by setting up the appropriate field in the PEL Operation register.

Only the corresponding number of bits-per-PEL in the destination are required in this register (for example, if the destination is 4 bits-per-PEL, only the 4 low-order bits of this register are used).

The bits of this register that are more significant than the number of bits-per-PEL need not be written.

! **Operation Dimension 1 Register (Offset 60)**

! This write-only register has an offset of hex 60.



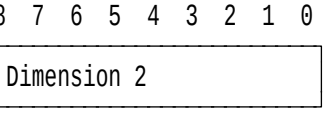
! *Figure 3-147. Operation Dimension 1 Register, Offset Hex 60*

! The Operation Dimension 1 register (bits 15– 0) specifies the width
! of the rectangle to be drawn by the PxBlt function, or the length of
! line in a line draw operation. The value is an unsigned quantity,
! and must be one less than the required width. To draw a line 10
! PELs long, the value 9 must be written to this register.

! The value written to this register must be within the range
! (0 to 4095).

! **Operation Dimension 2 Register (Offset 62)**

! This write-only register has an offset of hex 62.



! *Figure 3-148. Operation Dimension 2 Register, Offset Hex 62*

! The Operation Dimension 2 register (bits 15– 0) specifies the
! height of the rectangle to be drawn by the PxBlt function. The
! value is an unsigned quantity, and must be one less than the
! required height. To draw a rectangle 10 PELs high, the value 9
! must be written to this register.

! The value written to this register must be within the range
! (0 to 4095).

Mask Map Origin X Offset Register (Offset 6C)

This write-only register has an offset of hex 6C.

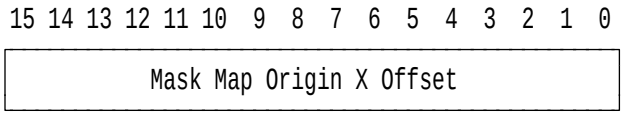


Figure 3-149. Mask Map Origin X Offset Register, Offset Hex 6C

The Mask Map Origin X Offset register (bits 15– 0) specifies the X offset of the mask map origin, relative to the origin of the destination map.

The value written to this register must be within the range (0 to 4095).

Mask Map Origin Y Offset Register (Offset 6E)

This write-only register has an offset of hex 6E.

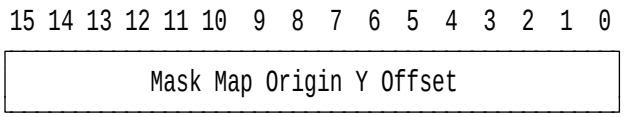


Figure 3-150. Mask Map Origin Y Offset Register, Offset Hex 6E

The Mask Map Origin Y Offset register (bits 15– 0) specifies the Y offset of the mask map origin, relative to the origin of the destination map.

The value written to this register must be within the range (0 to 4095).

Source X Address Register (Offset 70)

This read/write register has an offset of hex 70.

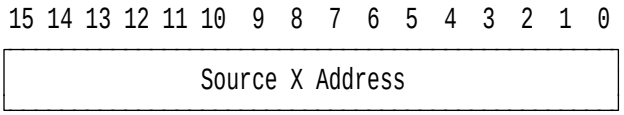


Figure 3-151. Source X Address Register, Offset Hex 70

The Source X Address register (bits 15– 0) specifies the X coordinate of the coprocessor operation source PEL.

The value written to this register must be within the range (0 to 4095).

Source Y Address Register (Offset 72)

This read/write register has an offset of hex 72.

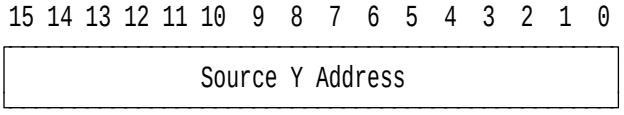


Figure 3-152. Source Y Address Register, Offset Hex 72

The Source Y Address register (bits 15– 0) specifies the Y coordinate of the coprocessor operation source PEL.

The value written to this register must be within the range (0 to 4095).

Pattern X Address Register (Offset 74)

This read/write register has an offset of hex 74.

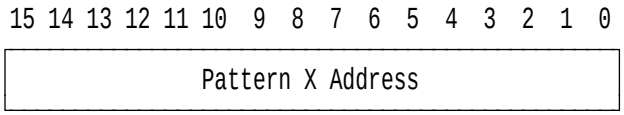


Figure 3-153. Pattern C Address Register, Offset Hex 74

The Pattern X Address register (bits 15– 0) specifies the X coordinate of the coprocessor operation pattern PEL.

The value written to this register must be within the range (0 to 4095).

Pattern Y Address Register (Offset 76)

This read/write register has an offset of hex 76.

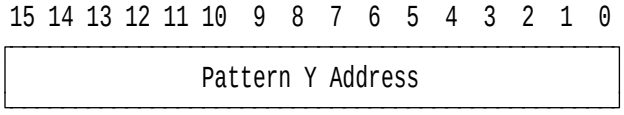


Figure 3-154. Pattern Y Address Register, Offset Hex 76

The Pattern Y Address register (bits 15– 0) specifies the Y coordinate of the coprocessor operation pattern PEL.

The value written to this register must be within the range (0 to 4095).

Destination X Address Register (Offset 78)

This read/write register has an offset of hex 78.

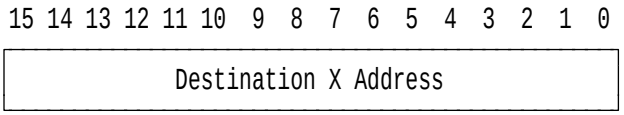


Figure 3-155. Destination X Address Register, Offset Hex 78

The Destination X Address register (bits 15– 0) specifies the X coordinate of the coprocessor operation destination PEL.

This register must be written as a 16-bit sign-extended twos complement number in the range (– 2048 to 6143).

Destination Y Address Register (Offset 7A)

This read/write register has an offset of hex 7A.

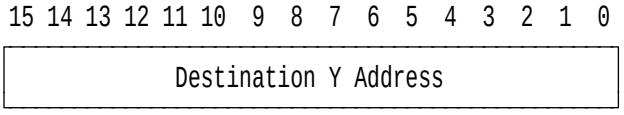


Figure 3-156. Destination Y Address Register, Offset Hex 7A

The Destination Y Address register (bits 15– 0) specifies the Y coordinate of the coprocessor operation destination PEL.

This register must be written as a 16-bit sign-extended twos complement number in the range (– 2048 to 6143).

PEL Operations Register (Offset 7C)

This write-only register has an offset of hex 7C.

The PEL Operations register is used to define the flow of data during an operation. It specifies the address update function that is to be performed, and initiates PxBlt and line draw operations.

The contents of the PEL Operation register are preserved throughout an operation.

The byte order of this register is dependent on whether the Intel or Motorola register interface is enabled. The following figure shows the Intel and Motorola views of the PEL Operations register.

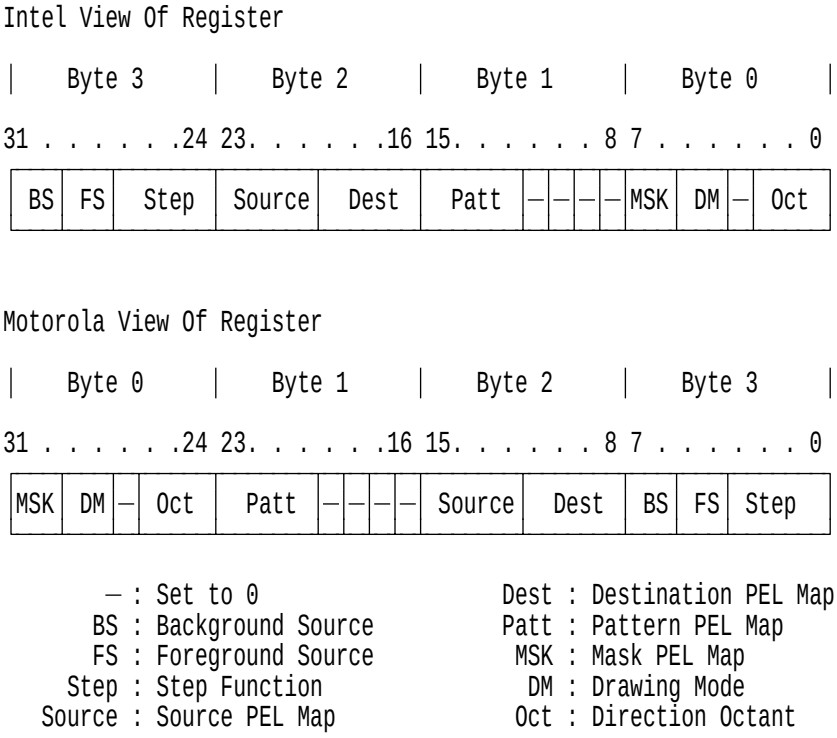


Figure 3-157. PEL Operations Register, Offset Hex 7C

All operations, with the exception of draw and step (see “Draw and Step” on page 3-108 and “Direction Steps Register (Offset 2C)” on page 3-148), are initiated by writing to the *most-significant byte* of this register. Therefore, an operation can be initiated by a single 32-bit write, two 16-bit writes when bytes 2 and 3 are written last, or four 1-byte accesses where byte 3 is written last.

The fields in the PEL Operation register are shown, by bytes, in the following figures.

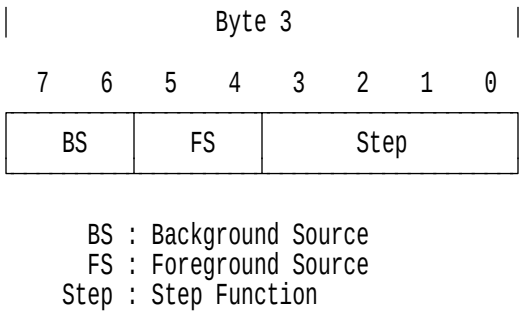


Figure 3-158. PEL Operations Register, Byte 3

BS The Background Source field (byte 3; bits 7, 6) determines the background source that is to be combined with the destination when the pattern PEL equals 0 (background mix).

BS Field (binary)	Background Source
0 0	Background Color
0 1	Reserved
1 0	Source PEL Map
1 1	Reserved

Figure 3-159. PEL Operations Register Background Source Value Assignments

FS The Foreground Source field (byte 3; bits 5, 4) determines the foreground source that is to be combined with the destination when the pattern PEL equals 1 (foreground mix).

FS Field (binary)	Foreground Source
0 0	Foreground Color
0 1	Reserved
1 0	Source PEL Map
1 1	Reserved

Figure 3-160. PEL Operations Register Foreground Source Value Assignments

Step The Step Function field (byte 3; bits 3– 0) determines how the coprocessor address is modified as PEL data is manipulated. This field can be regarded as the coprocessor function code. Writing to this field starts the coprocessor operation, except for draw and step functions. Draw and step operations are started by writing to the Direction Steps register.

Step Field (binary)	Step Function
0 0 0 0	Reserved
0 0 0 1	Reserved
0 0 1 0	Draw and Step Read
0 0 1 1	Line Draw Read
0 1 0 0	Draw and Step Write
0 1 0 1	Line Draw Write
0 1 1 0	Reserved
0 1 1 1	Reserved
1 0 0 0	PxBlt
1 0 0 1	Inverting PxBlt
1 0 1 0	Area Fill PxBlt
1 0 1 1	Reserved
.	.
.	.
1 1 1 1	Reserved

Figure 3-161. PEL Operations Register Step Function Value Assignments

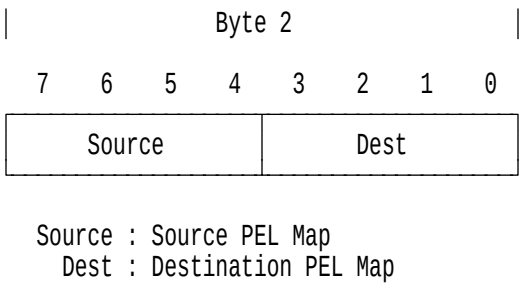


Figure 3-162. PEL Operations Register, Byte 2

Source The Source PEL Map field (byte 2; bits 7– 4) determines the location of PEL map source data. The combination of this field and the Foreground and Background Source fields determine the data that is to be used as the source data for coprocessor functions.

Source Field (binary)	Source PEL Map
0 0 0 0	Reserved
0 0 0 1	PEL Map A
0 0 1 0	PEL Map B
0 0 1 1	PEL Map C
0 1 0 0	Reserved
.	.
.	.
1 1 1 1	Reserved

Figure 3-163. PEL Operations Register Source PEL Map Value Assignments

Dest The Destination PEL Map field (byte 2; bits 3– 0) determines the location of the destination data to be modified during an operation.

Dest Field (binary)	Destination PEL Map
0 0 0 0	Reserved
0 0 0 1	PEL Map A
0 0 1 0	PEL Map B
0 0 1 1	PEL Map C
0 1 0 0	Reserved
.	.
.	.
1 1 1 1	Reserved

Figure 3-164. PEL Operations Register Destination PEL Map Value

Assignments

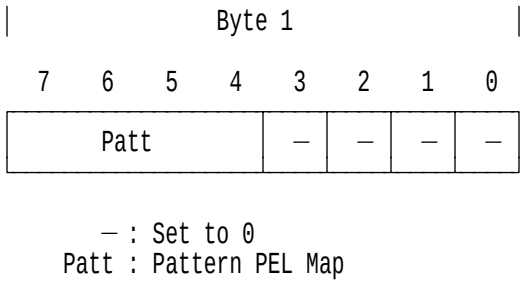


Figure 3-165. PEL Operations Register, Byte 1

Patt The Pattern PEL Map field (byte 1; bits 7– 4) determines the pattern data to be used during an operation. Code 1000 causes the coprocessor to assume that the pattern is 1 across the whole operation, and to use the foreground function on all PELs. This effectively turns off the use of the pattern. Code 1001 causes the pattern to be generated from source data. Every 0 PEL in the source generates a background pattern PEL; every nonzero PEL in the source generates a foreground pattern PEL.

Patt Field (binary)	Pattern PEL Map
0 0 0 0	Reserved
0 0 0 1	PEL map A
0 0 1 0	PEL map B
0 0 1 1	PEL map C
0 1 0 0	Reserved
0 1 0 1	Reserved
0 1 1 0	Reserved
0 1 1 1	Reserved
1 0 0 0	Foreground (fixed)
1 0 0 1	Generated from Source
1 0 1 0	Reserved
.	.
.	.
1 1 1 1	Reserved

Figure 3-166. PEL Operations Register Pattern PEL Map Value Assignments

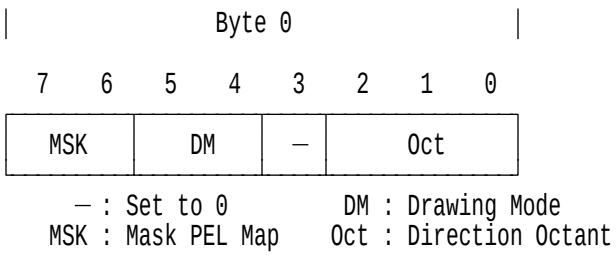


Figure 3-167. PEL Operations Register, Byte 0

MSK The Mask PEL Map field (byte 0; bits 7, 6) determines how the mask map is used. See “Scissoring with the Mask Map” on page 3-103 for details of the mask map modes.

MSK Field (binary)	Mask PEL Map
0 0	Mask Map Disabled
0 1	Mask Map Boundary Enabled
1 0	Mask Map Enabled
1 1	Reserved

Figure 3-168. PEL Operations Register Mask PEL Map Value Assignments

DM The Drawing Mode field (byte 0; bits 5, 4) determines the attributes of line draw and draw and step operations.

DM Field (binary)	Drawing Mode
0 0	Draw All PELs
0 1	Draw First PEL Null
1 0	Draw Last PEL Null
1 1	Draw Area Boundary

Figure 3-169. PEL Operations Register Drawing Mode Value Assignments

Oct The Direction Octant field (byte 0; bits 2– 0) is comprised of three bits, DX, DY, and DZ.

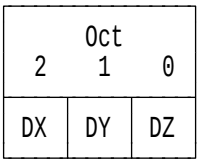


Figure 3-170. PEL Operations Register Direction Octant Values

XGA System Interface

Multiple Instances

Up to eight Instances of an XGA subsystem can be installed in a system unit. Addressing the I/O registers, memory-mapped registers, and video memory for each Instance is controlled by the contents of the XGA POS registers. See “XGA POS Registers” on page 3-170 for more information.

Multiple XGA Subsystems in VGA Mode

The VGA has only one set of addresses allocated to it. It is not possible to have multiple XGA subsystems in VGA mode responding to update requests simultaneously. However, more than one XGA subsystem can be in VGA mode if only one has VGA address decoding enabled using the Operating Mode register. Subsystems with VGA address decoding disabled continue to display the correct picture. See “XGA Adapter Coexistence with VGA” on page 3-223 for further information.

Note: The XGA *must not* be disabled using the Subsystem Enable field in XGA POS Register 2.

Multiple XGA Subsystems in 132-Column Text Mode

In 132-column text mode, the XGA responds to VGA address decodes, and the same rules apply as for multiple XGA subsystems in VGA mode. See “XGA Adapter Coexistence with VGA” on page 3-223 for further information.

Multiple XGA Subsystems in Extended Graphics Mode

Extended Graphics modes are controlled by a bank of 16 I/O registers that are located in one of eight possible locations. Up to eight XGA subsystems can be installed in a system unit. Each Instance of XGA installed is positioned at a unique I/O and memory location, so each can be used independently in the system. See “Multiple XGA Subsystems” on page 3-223 for details on controlling multiple XGAs.

The XGA coprocessor memory-mapped registers occupy a bank of 128 contiguous register addresses that are mapped in memory

space. These registers can also be relocated, allowing up to eight Instances of the XGA coprocessor to coexist in a system.

The locations of these registers are controlled by the XGA POS registers. See “XGA POS Registers” for the register details, and see“XGA Adapter Identification, Location and XGA Mode Setting” on page 3-192 for programming considerations on reading and using the data contained in them.

XGA POS Registers

The XGA subsystem has movable I/O addresses for the display controller, allowing more than one XGA subsystem to be installed in a system unit.

All POS registers detailed in this section are set up during system configuration and must never be written. All registers are specified relative to a base address. Details of how to locate the base address and read the registers are given in “XGA Adapter Identification, Location and XGA Mode Setting” on page 3-192.

Register Usage Guidelines

Unless otherwise stated, the following are guidelines to be used when accessing the POS registers:

- All registers are 8 bits long.
- All registers are read only.
- Special reserved register bits must be used as follows:
  - Register bits marked with ‘– ’ must be masked off if the contents of the register is to be tested.
  - Register bits marked with ‘#’ are reserved and the state of these bits must be preserved. This register must be masked off if the contents of the register is to be tested.

Subsystem Identification Low Byte Register (Base + 0)

This read-only register is located at base address + 0. *Do not write to* this register. When read, this register returns the low byte of the POS ID.

Subsystem Identification High Byte Register (Base + 1)

This read-only register is located at base address + 1, *do not write to* this register. When read, this register returns the high byte of the POS ID.

POS Register 2 (Base + 2)

This read-only register is located at base address + 2. *Do not write to* this register.

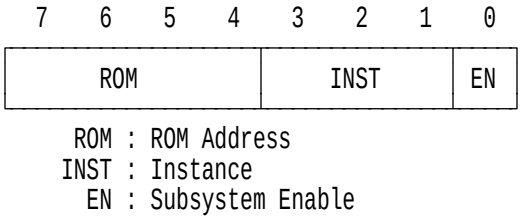


Figure 3-171. POS Register 2, Base Address + 2

The fields in this register are defined as follows:

ROM The ROM Address field (bits 7– 4) specifies which of 16 possible 8KB memory locations has been assigned to the XGA ROM. The ROM occupies the first 7KB of this 8KB block; the other 1KB is occupied by the coprocessor memory-mapped registers.

The Instance field specifies the 128-byte section within this 1KB block that is allocated to the subsystem. See the following figure. For example, Instance 2 has its coprocessor registers located in the third 128-byte section of the 1KB block.

ROM Address		Coprocessor Register Base Address (hex)							
		Instance:							
Field	Range (hex)	0	1	2	3	4	5	6	7
0000	C0000 : C1BFF	C1C00	C1C80	C1D00	C1D80	C1E00	C1E80	C1F00	C1F80
0001	C2000 : C3BFF	C3C00	C3C80	C3D00	C3D80	C3E00	C3E80	C3F00	C3F80
0010	C4000 : C5BFF	C5C00	C5C80	C5D00	C5D80	C5E00	C5E80	C5F00	C5F80
0011	C6000 : C7BFF	C7C00	C7C80	C7D00	C7D80	C7E00	C7E80	C7F00	C7F80
0100	C8000 : C9BFF	C9C00	C9C80	C9D00	C9D80	C9E00	C9E80	C9F00	C9F80
0101	CA000 : CBBFF	CBC00	CBC80	CBD00	CBD80	CBE00	CBE80	CBF00	CBF80
0110	CC000 : CDBFF	CDC00	CDC80	CDD00	CDD80	CDE00	CDE80	CDF00	CDF80
0111	CE000 : CFBFF	CFC00	CFC80	CFD00	CFD80	CFE00	CFE80	CFF00	CFF80
1000	D0000 : D1BFF	D1C00	D1C80	D1D00	D1D80	D1E00	D1E80	D1F00	D1F80
1001	D2000 : D3BFF	D3C00	D3C80	D3D00	D3D80	D3E00	D3E80	D3F00	D3F80
1010	D4000 : D5BFF	D5C00	D5C80	D5D00	D5D80	D5E00	D5E80	D5F00	D5F80
1011	D6000 : D7BFF	D7C00	D7C80	D7D00	D7D80	D7E00	D7E80	D7F00	D7F80
1100	D8000 : D9BFF	D9C00	D9C80	D9D00	D9D80	D9E00	D9E80	D9F00	D9F80
1101	DA000 : DBBFF	DBC00	DBC80	DBD00	DBD80	DBE00	DBE80	DBF00	DBF80
1110	DC000 : DDBFF	DDC00	DDC80	DDD00	DDD80	DDE00	DDE80	DDF00	DDF80
1111	DE000 : DFBFF	DFC00	DFC80	DFD00	DFD80	DFE00	DFE80	DFF00	DFF80

Figure 3-172. XGA ROM, Memory-Mapped Register Assignments

Note: The Coprocessor registers may also be able to be accessed using alternative addresses in the Protect Mode address range. See "XGA Adapter Identification, Location and XGA Mode Setting" on page 3-192 for general details of locating and using the XGA registers, and "Alternative XGA Coprocessor Register Set" on page 3-269 for specific details on the alternative "Shadowed" Coprocessor Registers.

INST The value in this field indicates the *Instance* of this XGA subsystem. Coexisting XGA Subsystems each have unique Instance values.

The Instance field (bits 3– 1) specifies the set of I/O addresses allocated to the display controller registers. See Figure 3-173. The lowest address of each set of addresses is referred to as the I/O base address.

INST Field (binary)	Instance Number	Instance I/O Base Address (hex)
000	0	2100
001	1	2110
010	2	2120
011	3	2130
100	4	2140
101	5	2150
110	6	2160
111	7	2170

Figure 3-173. I/O Device Address Bit Assignment

EN The Subsystem Enable field (bit 0) indicates whether the subsystem is enabled. When read as 1, the subsystem is enabled for address decoding for all non-POS addresses. When read as 0, only POS registers can be accessed; all other accesses to the subsystem have no effect.

POS Register 4 (Base + 4)

This read-only register is located at base address + 4. *Do not write to* this register.

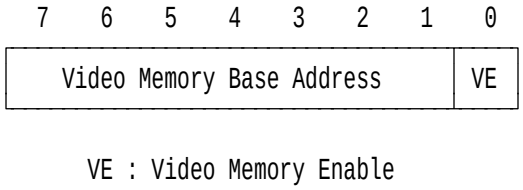


Figure 3-174. POS Register 4, Base Address + 4

The fields in this register are defined as follows:

Video Memory Base Address
This field (bits 7– 1) contains the most significant 7 bits of the address where the XGA memory is located. Three more bits are provided by the Instance in POS byte 1. This gives a video memory base address on a 4MB boundary. See the following figure.

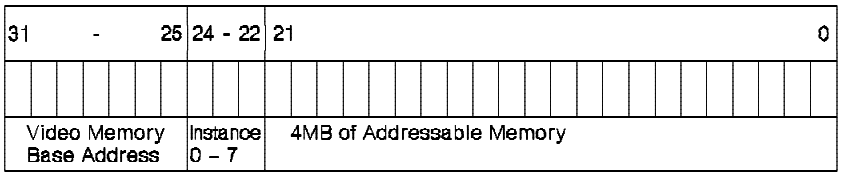


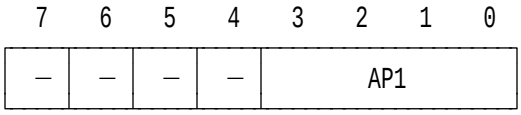
Figure 3-175. XGA Video Memory Base Address.

For example, if the video memory base address is set to 1 and Instance 6 has been selected, the XGA video memory is located at hex 03800000.

VE When the Video Memory Enable field (bit 0) is set to 0, the 4MB aperture is disabled; when set to 1, the 4MB aperture is enabled.

POS Register 5 (Base + 5)

This read-only register is located at base address + 5. *Do not write to* this register.



– : Undefined On Read
AP1 : 1MB Aperture Base Address

Figure 3-176. POS Register 5, Base Address + 5

The field in this register is defined as follows:

AP1 The 1MB Aperture Base Address field (bits 3– 0) indicates where the 1MB aperture is placed in system address space, or if the aperture has been disabled. The following figure describes the use of this field.

AP1 Field (binary)	1MB Aperture Location (hex)
0000	Disabled
0001	00100000
0010	00200000
0011	00300000
0100	00400000
0101	00500000
0110	00600000
0111	00700000
1000	00800000
1001	00900000
1010	00A00000
1011	00B00000
1100	00C00000
1101	00D00000
1110	00E00000
1111	00F00000

Figure 3-177. 1MB Aperture Base Address Value Assignments

Virtual Memory Description

The XGA coprocessor can address either real or virtual memory. When addressing real memory, the linear address calculated by the coprocessor is passed directly to the system microprocessor or local video memory. When addressing virtual memory, the linear address from the coprocessor is translated by on-chip virtual memory translation logic before the translated address is passed to the system microprocessor or local video memory. Virtual address translation is enabled or disabled by a control bit in the XGA.

The coprocessor uses two levels of tables to translate the linear address from the coprocessor to a physical address. Addresses are translated through a page directory and page table to generate a physical address to memory pages that are 4KB in size. The page directory and page tables are of the same form as those used by the 80386 Processor Paging Unit.

Address Translation

The linear address from the coprocessor is divided into three fields that are used to look up the corresponding physical address. The fields, called directory index, table index, and offset, are illustrated in the following figure.

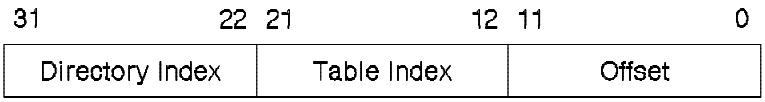


Figure 3-178. Linear Address Fields

The location of the page directory is at a fixed physical address in memory that must be on a page (4KB) address boundary. The coprocessor has a Page Directory Base Address register that must be loaded with the address of the page directory base.

The translation process is illustrated in Figure 3-179 on page 3-178.

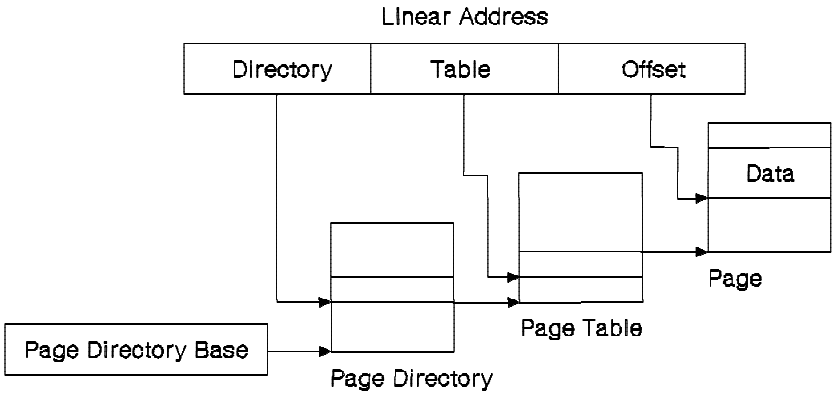


Figure 3-179. Linear to Physical Address Translation

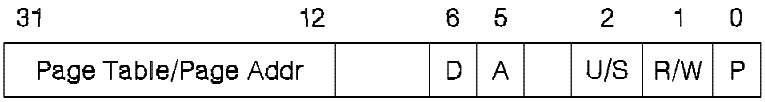
The Directory Index field of the linear address is used to index into the page directory. The entry read from the page directory contains a 20-bit page table address and some statistical information in the low order bits.

The 20-bit page table address points to the base of a page table in memory. The Table Index field in the linear address is used to index into the page table. The entry read from the page table contains a 20-bit page address and some statistical information in the low order bits.

The 20-bit page address points to the base of a 4KB page in memory. The Offset field in the linear address is used to index into the page. The entry read from the page contains the data required by the memory access.

Page Directory and Page Table Entries

The entries of the page directory and page table are very similar. The format of an entry is shown in the following figure.



D - Dirty Bit
A - Accessed Bit
U/S - User/Supervisor Bit
R/W - Read/Write Bit
P - Present Bit

Figure 3-180. Page Directory and Page Table Entry

The top 20 bits of the entry are either the page table address or the page address. The low order bits are as follows:

Dirty Bit (bit 6): This bit is set before a write to an address covered by that page table entry occurs. The dirty bit is undefined for page directory entries.

Accessed Bit (bit 5): This bit is set for both types of entry before a read or write access occurs to an address covered by the entry.

User/Supervisor and Read/Write Bits (bits 2,3): These bits prevent unauthorized use of page directory and page table entries. Accesses by the coprocessor can be defined as a supervisor or user access, depending on the status of the application using the coprocessor. The access type is defined by a bit in the virtual memory (VM) Control register. If the access is defined as supervisor, no protection is provided and all accesses to the page directory and page tables are permitted.

For a user access, the U/S and R/W bits are checked to ensure that access to that entry is permitted. The meaning of these bits is shown in the following figure.

U/S	R/W	Access Rights of User
0	0	Access not permitted
0	1	Access not permitted
1	0	Reads permitted, writes not permitted
1	1	Reads and writes permitted

Figure 3-181. Page Directory and Page Table Access Rights in User Mode

Present Bit (bit 0): The present bit indicates whether a page directory or page table entry can be used in translation. If the bit is set, it indicates that the page table or page that the entry refers to is present in memory.

The XGA Implementation of Virtual Memory

The XGA coprocessor operates with a page directory and page tables in the format described. The coprocessor has its own internal cache of translated addresses to avoid it having to perform the two-stage translation process on every coprocessor access. This cache is referred to as a translate look-aside buffer.

The Translate Look-aside Buffer

The translate look-aside buffer (TLB) has two entries, one entry for the source, pattern, and mask PEL maps, and another for the destination PEL map, as shown in Figure 3-182. Each entry is reserved specifically for use by one of these maps. Each entry in the TLB contains the top 20 bits of a linear address (the address tag), an entry valid flag bit, and the top 20 bits of the physical address (the real page address) corresponding to that linear address. When a linear address is passed from the coprocessor to the virtual address hardware, the top 20 bits of the linear address are first compared against the appropriate TLB entry address tag. If they match and the TLB entry flag bit is valid, the real page address in that TLB is used as the top 20 bits of the physical address for that access. The bottom 12 bits of the physical address are provided from the bottom 12 bits of the linear address (the offset).

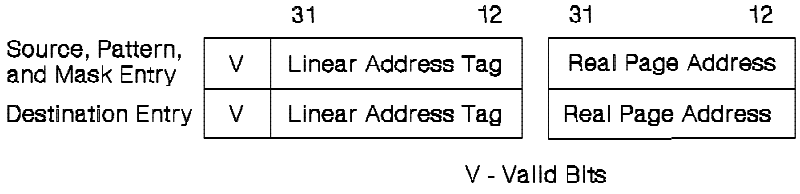


Figure 3-182. Translate Look-Aside Buffer

If the linear address from the coprocessor matches the address tag in the TLB for the particular map in use, the access is said to have caused a TLB hit. If the tag does not match, a TLB miss occurs.

The TLB contents are cleared by the hardware under the following circumstances:

- Whenever the Page Directory Base Address register is written
- Whenever a coprocessor operation is suspended (by setting the suspend operation bit in the Coprocessor Control register). See “Coprocessor Control Register (Offset 11)” on page 3-136.

TLB Misses

If a TLB miss occurs, the coprocessor automatically performs the two-level translation required to form the required page address. The contents of the XGA Page Directory Base Address register are used to access the appropriate page directory entry that is used to access the appropriate page table entry. The real page address, resulting from the translation process, is stored in the TLB for use by subsequent accesses that address the same page.

Memory access performed by the coprocessor can be categorized as follows:

Read accesses Performed on the source, pattern, mask, and destination maps.

Write accesses Performed only on the destination map.

When the virtual memory hardware accesses the page directory and page tables for a TLB miss, it examines and updates the flags in the low order bits of the entries, as follows:

Accessed Bit (bit 5): Any access (read or write) sets this bit in both the page directory and page table entries.

Dirty Bit (bit 6): Write accesses set the dirty bit in the page table entry. The dirty bit is undefined in page directory entries.

User/Supervisor and Read/Write Bits (bits 2, 3): These bits are examined by the coprocessor. The coprocessor has a bit, programmed by the host operating system, to indicate whether it is being used by a supervisor or user. In user mode the coprocessor determines whether access is permitted, depending on the state of the user/supervisor and the read/write bits in the page directory and page table entries. If access is not permitted, the coprocessor sends a VM-protection-violation interrupt to the system microprocessor and terminates the access cycle. The host

operating system must take the appropriate action to recover from the protection-violation interrupt.

Present Bit (bit 0): The coprocessor examines the present bit of the page directory and page table entries. If this bit is not set, it indicates that the page table or page corresponding to that entry may not be resident in memory, and the XGA sends a VM page-not-present interrupt to the system microprocessor. The host operating system fetches the page table or page and places it in memory. The access can then be completed.

Remaining Page Directory or Page Table Entry Bits: The coprocessor ignores all the other bits in the page directory or page table entry. The coprocessor does not modify these bits; they can be used by the operating system. It is advisable to keep entries in the same format as the 80386 page directory and page table entry formats; therefore the Intel rules on the use of the remaining bits must be followed.

System Coherency

In any virtual memory system where more than one device is accessing virtual memory contents, problems can arise over coherency. It is essential that one device does not corrupt the tables or pages of the other device, and the tables and TLBs are kept coherent (in step) with the physical allocation of storage. The hardware mechanism provided by the coprocessor is sufficient to implement coherent virtual memory systems, but care should be taken to avoid coherency problems.

In particular, it is recommended that the 80386 and the coprocessor do not share page directories or page tables. Pages must not be marked as present unless they are locked in place in memory. This maintains coherency between TLB entries in the coprocessor and the true current allocation of real memory. It prevents the operating system from moving these pages out of memory while the coprocessor is accessing them.

VM Page-Not-Present Interrupts

When the coprocessor detects that a page table or page is not present, it sends a page-not-present interrupt to the system microprocessor. The system microprocessor operating system then fetches the required page table or page (usually from disk) and places it in memory. The system microprocessor can determine the faulting address by reading the Current Virtual Address register. After the required page table or page has been fetched, the operating system restarts the faulting memory access by clearing the page-not-present interrupt bit in the XGA. This causes the hardware to retry the access to the faulted entry.

The system microprocessor operating system will probably switch tasks when receiving a page-not-present interrupt. In this case it suspends the coprocessor operation in the normal way (see “Suspending Coprocessor Operations” on page 3-130) before clearing the page-not-present interrupt. The coprocessor state is then saved. When the task where the page-not-present interrupt occurred is restarted, the coprocessor state is restored, the required page table or page is placed in memory, the interrupt is cleared, and the coprocessor operation is restarted. The interrupt must be cleared before the coprocessor operation is restarted, otherwise further interrupts can be lost.

VM Protection-Violation Interrupts

If the coprocessor is directed to access tables or pages that are not permitted (as defined by the user/supervisor and read/write entry bits), the coprocessor generates a protection-violation interrupt. This indicates that something is wrong with the virtual memory system (as set up by operating system software), or that the coprocessor has been programmed incorrectly.

The operating system will probably terminate the coprocessor operation and possibly terminate the faulting task. The coprocessor operation can be terminated by writing to a control bit in the Coprocessor Control register. The coprocessor responds in a similar manner for protection-violation interrupts as it does for page not present interrupts; clearing the interrupt causes the hardware to retry the memory access. To avoid a repeated interrupt, the coprocessor operation must be terminated before the protection-violation interrupt is cleared.

The XGA in Segmented Systems

In a segmented system design, all memory is allocated in blocks called segments. Memory within a segment is guaranteed to be contiguous, and can therefore be addressed directly by the coprocessor using physical addresses (VM is turned off). The segment must be locked in place before any coprocessor operation to ensure that the operating system does not reuse the memory during the operation.

When using 16-bit addressing in the 80386 (for example, under the OS/2 version 1.3 operating system), it is not possible to define a segment of more than 64KB. If the coprocessor data in system memory is restricted to no more than 64KB in length, a single segment can be used and the coprocessor can directly address the data using physical addresses.

When using the OS/2 version 1.3 operating system, larger areas of memory can be requested, but are given in blocks of 64KB (maximum) that are unlikely to be contiguous in real memory. If larger areas in system memory are required, the driving software can turn on the coprocessor VM address translation and perform its own memory management using memory allocated to it by the operating system.

Virtual Memory Registers

The following registers provide virtual memory support for the PEL interface.

Page Directory Base Address Register (Coprocessor Registers, Offset 0)

This write-only register has coprocessor registers offset of hex 0.

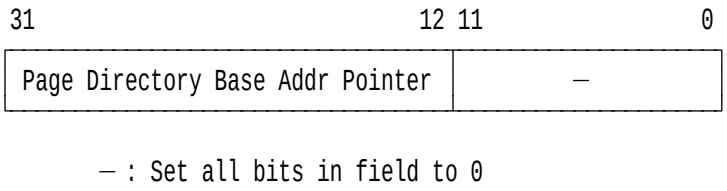


Figure 3-183. Page Directory Base Address Register, Offset Hex 0

The field in this register is defined as follows:

Page Directory Base Addr Pointer
This field (bits 31– 12) is a 20-bit pointer to the page in physical memory containing the current page directory for the current task.

Loading this register clears the translate look-aside buffer.

Note: This register can be loaded only after the XGA is put in supervisor mode.

Current Virtual Address Register (Coprocessor Registers, Offset 4)

This read-only register has coprocessor registers offset of hex 4.
Do not write to this register.

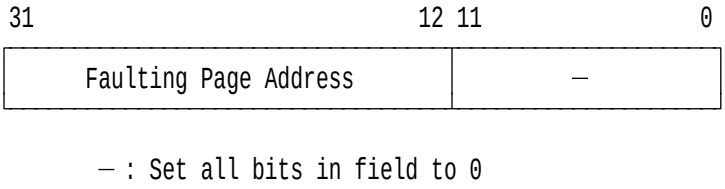


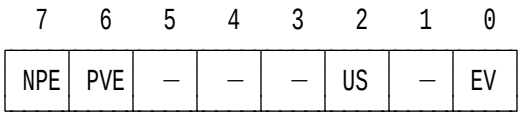
Figure 3-184. Current Virtual Address Register, Offset Hex 4

The field in this register is defined as follows:

Faulting Page Address
After a VM hardware-not-present interrupt or protection interrupt, read this field (bits 31– 12) to find the fault address page.

Virtual Memory Control Register (I/O Address 21x6)

This read/write register has an I/O address of hex 21x6. The Virtual Memory Control register is directly mapped to I/O address space.




```
  — : Set to 0, Undefined on Read
NPE : VM Page Not Present Interrupt Enable
PVE : VM Protection Violation Interrupt Enable
 US : User / Supervisor
 EV : Enable Virtual Address Lookup
```

Figure 3-185. Virtual Memory Control Register

The fields in this register are defined as follows:

NPE The VM Page Not Present Interrupt Enable field (bit 7) controls the sending of an interrupt when a VM page not present condition is detected. When set to 1, an interrupt is sent to the system microprocessor when the not present condition is detected. When set to 0, the not present condition does not send an interrupt. In both cases, the contents of the appropriate VM Interrupt Status register status bit are updated when the not present condition is detected.

PVE The VM Protection Violation Interrupt Enable field (bit 6) controls the sending of an interrupt when a VM protection violation condition is detected. When set to 1, an interrupt is sent to the system microprocessor when the protection violation condition is detected. When set to 0, the protection violation condition does not send an interrupt. In both cases the contents of the appropriate VM Interrupt Status register status bit are updated when the protection violation condition is detected.

US The User/Supervisor field (bit 2) indicates the privilege level of the currently-executing task. When set to 0, the executing task is at privilege levels 0, 1, or 2 (a supervisor task). When set to 1, the executing task is at privilege level 3 (a user task).

If set to supervisor (0), no protection checking is performed by the coprocessor on page directory and page table protection bits. If set to user (1), checking is performed, and a protection interrupt is sent if permitted access rights are violated.

EV The Enable Virtual Address Lookup field (bit 0) controls the virtual address translation. Subsequent addresses generated by the PEL interface hardware are looked up in page tables. If this bit is not set:

- Bit maps must be resident and contiguous.
- The PEL map base addresses are physical addresses.
- All addresses generated by the coprocessor are physical addresses.
- Nonpaged operating systems are supported.

Virtual Memory Interrupt Status Register (I/O Address 21x7)

This read/write register has an I/O address of hex 21x7. The Virtual Memory Interrupt Register is directly mapped to I/O address space.

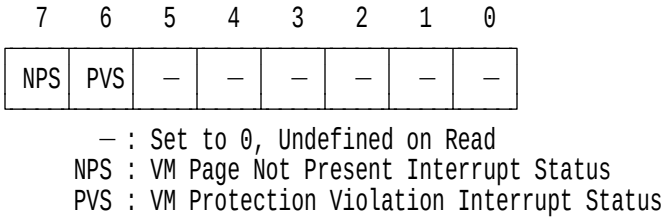


Figure 3-186. Virtual Memory Interrupt Status Register

The fields in this register are defined as follows:

NPS When a VM page not present condition occurs, the VM Page Not Present Interrupt Status field (bit 7) is automatically set to 1. This bit is reset to 0 by writing a 1 to it. This allows the value just read to be written back to clear the bits that were set. Writing a 0 to this bit has no effect.

Resetting this bit (writing a 1) causes the VM hardware to retry page translation. If this bit is to be reset before the not present condition has been repaired, the coprocessor operation must be suspended or terminated, otherwise another not-present interrupt is generated by the same not present condition.

PVS When a VM protection violation condition occurs, the VM Protection Violation Interrupt Status field (bit 6) is automatically set to 1. This bit is reset to 0 by writing a 1 to it. This lets the value just read to be written back to clear the bits that were set. Writing a 0 to this bit has no effect.

Resetting this bit (writing a 1) causes the VM hardware to retry page translation. If this bit is to be reset before the protection violation condition has been repaired, the coprocessor operation must be suspended or terminated, otherwise another protection-violation interrupt is generated by the same protection violation condition. Most operating systems do not attempt to