recover from a protection violation condition. The coprocessor operation causing this condition is terminated.

XGA Adapter Identification, Location and XGA Mode Setting

This section describes XGA subsystem identification and XGA mode setting. Information on VGA mode setting, and on switching from XGA mode to VGA mode, is described in “Switching the XGA subsystem from XGA to VGA Mode” on page 3-224.

An new mechanism for identifying XGA family subsystems has been introduced with the XGA-NI adapter, which is described in “XGA Display Mode Query and Set (DMQS).” DMQS will identify XGA family adapters, provide information for Extended Graphics Mode setting, and ensure migration for applications and drivers on future XGA hardware and displays.

As the original XGA subsystem did not support DMQS, it is necessary for device drivers written to run on both XGA and XGA-NI subsystems to incorporate both mechanisms, attempting first to identify and locate the XGA-NI subsystem using DMQS, and failing that to use the original XGA specific mechanism described in “Locating and Initialising the XGA Subsystem without DMQS” on page 3-208.

DMQS may be implemented on original XGA level subsystems, so software should not assume that the existence of DMQS capability is proof of XGA-NI level of function.

In a system with multiple XGA subsystems, if any one XGA subsystem has DMQS capability, it will provide DMQS services for all XGA subsystems recognised. Software should not use the original XGA identification procedure if DMQS BIOS Services are supported (see “DMQS BIOS Interface” on page 3-194).

XGA Display Mode Query and Set (DMQS)

DMQS Architecture Overview

DMQS will identify XGA family adapters, provide information for Extended Graphics Mode setting, and ensure migration for applications and drivers on future hardware and displays.

DMQS is comprised of two types of data: DMQS primary data and DMQS display information, contained in the display information files.

The primary data is returned to the software via an INT 10 Video BIOS code point.

The DMQS primary data contains the following information for each XGA instance:

- XGA implementation level identifier
- Location of XGA I/O registers or ports in I/O space
- Location of memory mapped XGA registers in system address space
- Location of 1 Meg memory mapped XGA aperture
- Location of 4 Meg memory mapped XGA aperture
- System address at which the XGA accesses video memory
- The composite ID of the attached display (see “Composite Display ID” on page 3-204)
- Amount of video memory available

The Adapter POST 'hooks' the INT 10 Video BIOS to point to two new code points. One code point returns the total size of the DMQS data array for all XGA instances. The other code point returns the DMQS data to the caller's buffer.

Software accesses the new BIOS code points to obtain the DMQS data stored by POST. Using the information from the composite ID field in the DMQS data, the device driver generates the DMQS display information file name. The DMQS display file is stored in a reserved directory named XGA$DMQS, or in the directory named in the DMQSPATH environment variable.

Note: In some operating systems, an alternative directory or path may be necessary.

Software should first look for the DMQSPATH environment variable to locate the directory containing the DMQS display information files. If the DMQSPATH environment does not exist, software

| should then look for a directory named XGA$DMQS on the boot
| disk.

| The DMQS display information file contains the following data:

- Display specific data
  - Physical display dimensions
  - Display type (color, mono, LCD, CRT etc.)
- The number of distinct Extended Graphics modes available on this display
- For each such mode available
  - The mode dimensions
  - The minimum level of XGA Adapter that supports this mode
  - XGA Standard Register settings to place the XGA Adapter in that particular mode.

| Using the data contained in both the XGA DMQS primary data and
| the DMQS display information file, the Device Driver/Application
| can determine:

- The capability and physical characteristics of the XGA family adapter and display
  - XGA Implementation level
  - Video memory size
  - Physical display dimensions
  - Color/Mono/LCD display information
- The location of all XGA registers and display buffers
- List of the modes available on the adapter/display combination
- Mode setting data for each mode

| With this information software can set the XGA and attached
| display into any available Extended Graphics mode without any
| hard-coded dependencies on displays or adapters.

| If the DMQS display information file cannot be located, software
| should revert to direct mode setting as described in “Locating and Initialising the XGA Subsystem without DMQS” on page 3-208.

| DMQS BIOS Interface

| The following two Video Int 10h code points are required to pass
| DMQS data to the software.

```
| Video BIOS Int 10h Software Interrupt function

| (AH) = 1FH - XGA Display Mode Query and Set (DMQS)


| (AL) = 00H - Read DMQS Data Length

| On Return:
|    (AL) = 1FH - function supported
|    (BX) = Number of bytes of DMQS data
```

```
| Video BIOS Int 10h Software Interrupt function

| (AL) = 01H - Read DMQS Data
|    (ES:DI) - User buffer pointer for return of information

| On Return:
|    User buffer contains DMQS data
|    (AL) = 1FH - function supported

|    As many as eight instances of XGA are possible.  One copy of
|    the following data structure exists for every instance:

|    (DI+00H) word  - Offset in bytes to DMQS data for next XGA instance
|    (DI+02H) byte  - Slot number

|    (DI+03H) byte  - XGA implementation function level identifier
|    (DI+04H) byte  - XGA implementation resolution level identifier

|    (DI+05H) word  - Vendor identifier - identifies card vendor
|    (DI+07H) word  - Vendor defined field

|    (DI+09H) word  - XGA Adapter I/O register base address
|    (DI+0BH) word  - XGA Coprocessor register base address - The
|                     location of memory mapped XGA coprocessor
|                     registers in system address space
|                     Multiply the value of this field by 10h to
|                     get the physical address
|    (DI+0DH) word  - 1 Megabyte System Video Memory Aperture - The
|                     location of 1 meg memory mapped XGA aperture
|                     in physical address space.  A value of 0
|                     indicates that the aperture is not allocated.
|                     Multiply the value of this field by 100000h
|                     to get the physical address
|    (DI+0FH) word  - 4 Megabyte System Video Memory Aperture - The
|                     location of 4 meg memory mapped XGA aperture
|                     in physical address space.  A value of 0
|                     indicates that the aperture is not allocated.
|                     Multiply the value of this field by 100000h
|                     to get the physical address
|    (DI+11H) word  - Video Memory Base Address - The location of video
|                     memory in XGA system address space.  Multiply the
|                     value of this field by 100000h to get the physical
|                     address.
|    (DI+13H) word  - Composite ID of the attached display
```

```
(DI+15H) byte  - Amount of video memory available,
                 in multiples of 256K bytes
(DI+16h) dword - Alternate XGA Coprocessor register base address -
                 The location of alternative memory mapped XGA
                 coprocessor registers in protect mode system
                 address space. A value of 0 indicates that the
                 alternative register location does not exist. A
                 non-zero value is the physical location in system
                 address space. If present, higher performance is
                 available using the registers at this location.

(DI+?? ) misc -  DMQS Data for further XGA Instances
```

Notes:

1. Although the bits per pixel information has been omitted from the BIOS interface, it can be inferred from the XGA level (current level has 16 bits per pixel maximum), the video memory size, and the number of pixels on the screen. Divide the video memory size in bits by the number of pixels on the screen in a particular mode (pixel height time pixel width) to get the maximum possible bits per pixel. Round off or down to the nearest supported bits per pixel value.
2. All fields will be coded in Intel format (low order byte first in word).
3. These calls return DMQS primary data for all XGA subsystems present in the system, both XGA and XGA-NI. XGA-NI subsystems recognise and provide DMQS services for non DMQS capable XGA subsystems.

DMQS Display Information Files

Software should expect to find the DMQS display information files in the XGA$DMQS directory on the boot disk, or alternatively in the directory specified in the DMQSPATH environment variable. This section on DMQS display information file installation is included for information only.

Adapter and System Diskettes: Configuration information for future video subsystems must include a composite DMQS display file. The composite file is made up of the individual DMQS display information files for all displays available at that time. It is made by merging the individual display information files into a composite file. The individual files can be merged in any order.

| The naming convention for the composite display file is the
| adapter ID followed by the letter M with an extension of DGS. For
| an adapter with a POS ID of hex 8FD9, the filename for the
| composite file is 8FD9M.DGS.

| Future systems with the XGA subsystem integrated on the system
| board will provide the composite file on the Reference Diskette.
| Adapters will provide the file on the Option Diskette.

| ***Display Diskette:*** Future displays which support new function, will
| ship with a display diskette to support the adapter in the extended
| graphics modes. The display diskette contains the DMQS display
| information file. The diskette will be a DOS format (FAT) diskette.

| The naming convention for the display information file is the letters
| MON followed a 4 character alpha-numeric string which would
| typically be an ASCII representation of the composite Display ID.
| These files use the file extension DGS. For a display with an ID of
| hex 001C, the filename for the display information file is
| MON001C.DGS.

| Information in the DMQS display information file helps identify
| levels of hardware support. The revision level for the DMQS
| display information file allows an update to the file to replace an
| earlier version. And within the individual mode table, a field
| identifies the minimum level of XGA hardware that must be
| present to use that mode.

| ***DMQS Display Information Files Installation:*** The installation of
| the display information files is operating system specific. The
| display files may be installed during device driver installation.
| Both the composite display file and any necessary individual
| display information files would be copied to a subdirectory named
| XGA$DMQS. The DMQSPATH environment variable may also be
| used to locate DMQS display information files in an alternative
| directory. The path to the XGA$DMQS directory and the means of
| finding the path is operating system specific.

| In a LAN Server environment using remote IPL, the boot disk for
| such systems is on the server. Display information files are
| included with the boot image for diagnostics. For other
| environments, the location of the display files is operating system
| specific.

DMQS Display Information File Structure

Overview: The DMQS display information files are stored in the XGA$DMQS directory of the boot drive unless the user chooses to store the XGA$DMQS subdirectory on another path. These files can be used by applications to determine the display characteristics, the available modes and the register values to use in setting modes.

The XGA$DMQS directory contains a number of individual files, one file for each display available.

As described in “DMQS Customisation File” on page 3-205, the composite ID returned in the DMQS primary Data Area may be over-ridden under user control.

If the DMQS display information file cannot be located, software should revert to direct mode setting as described in “Locating and Initialising the XGA Subsystem without DMQS” on page 3-208.

The following figures show the structure of the directory and the individual display information files.

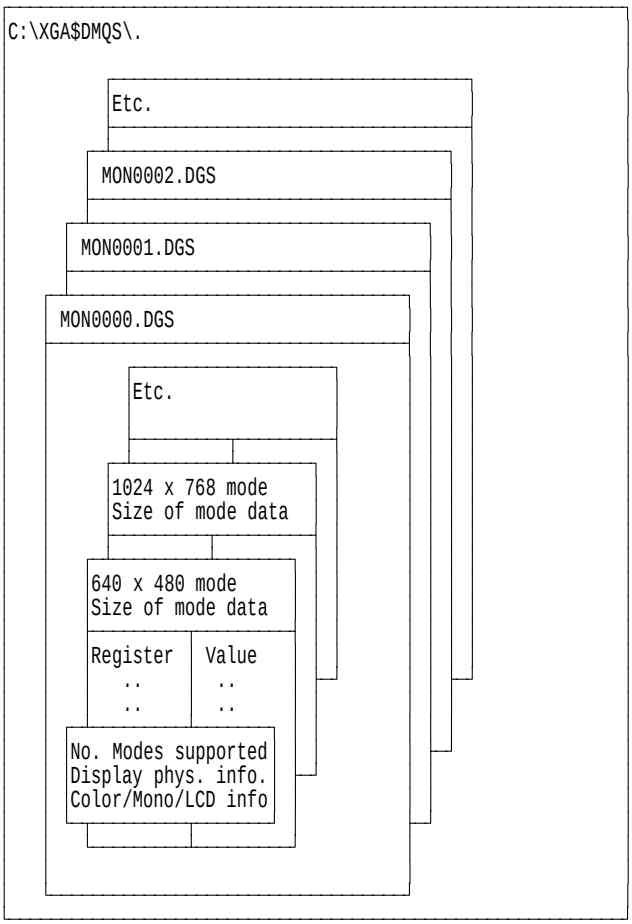


Figure 3-187. DMQS display information file structure

Details: The following table (Figure 3-188) shows the detailed layout of the DMQS display information file. All fields are in hexadecimal, Intel format (low order byte first in word).

Figure 3-188. DMQS display information file layout

Offset	Data Type	Description
00h	Bytes	DGS header data
		The header must be hardcoded in hex to the following value: EB1B 901D 0045 4453 5900 0000 0000 2020 2000 0022 00EB 0690 FE01 0101 00EB 0190 F8CB
		This should be ignored by software.
22h	Word	Total length of data in the file (bytes)
		This field is used to unpack multiple display information files from a composite file
24h	Word	Composite ID - used for unpacking a composite display file
26h	Byte	Revision Level - used to control file update
27h	Byte	Number of modes supported by this display
28h	Word	Display Type
		00h Mono CRT **01h** Color CRT **02h** Mono LCD **03h** Color LCD **04h** Mono Borderless capable **05h** Color Borderless capable
2Ah.	Word	Width of Screen in millimeters
2Ch.	Word	Height of Screen in millimeters
2Eh	Bytes	Null terminated ASCII string which describes the display in user friendly terms. The length of this field is 80 bytes.
		The string may be less than 80 bytes.
7Eh	Word	Offset of Individual Mode Data from the beginning of this file
80h	Word	Length of first optional extension, including identifier.
		A length value of 0 indicates no optional extensions exist.
		A non-zero value indicates that one or more optional extensions are present. Each optional extension consists of a length field, a 16 bit identifier, and the optional extension data. A zero length field terminates the chain of optional extensions.
82h	Word	First optional extension identifier (if present)
84h	Bytes	First optional extension data (if present)
xxh	Individual Mode Data	First of multiple tables of variable length mode specific data, see Figure 3-189 on page 3-201 for layout.

The following table (Figure 3-189) shows the DMQS display data for individual modes. Multiple instances of this data may exist within the display file, one for each mode available on the applicable display.

Figure 3-189 (Page 1 of 2). DMQS display information file mode data

Offset	Data Type	Description
00h	Word	Length of Individual Mode Data (bytes in this table)
02h	Word	Screen pixel Width of Mode
04h	Word	Screen pixel Height of Mode
06h	Word	Minimum XGA implementation levels on which this mode is supported. A display may be capable of more modes than an earlier adapter implementation may be capable of supporting. See "XGA Level Identifier" on page 3-205.
08h	Word	Vendor ID For modes which are unique to a specific vendor, this field must be set. Currently, this field is reserved. It will be set to zero.
0Ah	Word	Reserved This field is intended to be used as a vendor defined field. This field will be activated when the Vendor ID (above) field is set. This field may be defined by the vendor to specify unique modes of operation.
0Ch	Word	Mode function type flags This field identifies special capability the mode may have. **Bit 0** 0 = Normal, 1 = borderless **Bit 1** 0 = Interlaced, 1 = Non-Interlaced **Bits 2-15** Reserved
0Eh	Word	N = Offset in bytes to mode set data from beginning of this table
10h	Word	Mode Pixel Rate This 16 bit value is 4 times the mode pixel rate in megaHertz. For example, a value of 360 indicates a mode pixel rate of 90 MHz.
12h	Word	Mode Line Rate This 16 bit value is 10 times the mode line rate in KiloHertz. For example, a value of 315 indicates a mode line rate of 31.5 kHz.
14h	Word	Mode Frame Rate This 16 bit value is 10 times the mode frame rate in Hertz. For example, a value of 750 indicates a frame refresh rate of 75 Hz.
16h	Word	Length of first optional extension, including identifier A length value of 0 indicates no optional extensions exist A non-zero value indicates that one or more optional extensions are present. Each optional extension consists of a length field, a 16 bit identifier, and the optional extension data. A zero length field terminates the chain of optional extensions.
18h	Word	First optional extension identifier (if present)
1Ah	Bytes	First optional extension data (if present)

Note: Multiple Repetitions of Mode Setting Register Value "triplets," as follows:

N+00h, N+03h, etc.	Byte	Register Type **00h** Write to XGA Direct Access I/O Register **01h** Write to XGA Indexed Access I/O Register **02h** OR with XGA Indexed Access I/O Register **03h** OR with XGA Direct Access I/O Register **04h** AND with XGA Indexed Access I/O Register **05h** AND with XGA Direct Access I/O Register
N+01h, N+04h, etc.	Byte	Register Offset from Base I/O **Direct** 0-Fh **Indexed** 0-FFh
N+02h, N+05h, etc.	Byte	Register Value

Notes:

1. The 'Mode set data' section is a sequence of register settings required to place the hardware in the desired mode.

Mode setting from the DMQS Display Information File

The “mode set data” section of the DMQS display information file Individual Mode Data includes only the section of the mode setting code that is Display specific, such as CRT Controller settings.

The complete XGA subsystem DMQS mode set sequence consists of

1. Initial XGA subsystem display-independent initialisation
2. Display-dependent mode specific initialisation, using the “mode set data” from the display information file
3. Final XGA subsystem display-independent initialisation

The complete XGA subsystem mode set sequence is as shown in Figure 3-190 on page 3-204.

XGA Register Name	XGA Reg. ID	Value	Comments
Interrupt Enable	21x4	00	Initial Value
Interrupt Status	21x5	FF	
Operating Mode	21x0	04	Set Extended Graphics Mode
Palette Mask	64	00	Blank Display
Video Mem Aperture Ctl	21x1	00	Initial Value
Video Mem Aperture Index	21x8	00	Initial Value
Virt Mem Ctl	21x6	00	Initial Value
Memory Access Mode	21x9	As reqd.	Mode depth (no. colors)
Note: 1. At this point XGA subsystem mode setting becomes display and mode specific, and the “mode set” register settings read from the Display Configuration file should be written to the appropriate XGA registers. 2. The initial palette should then be loaded, by writing to the appropriate XGA subsystem palette/sprite registers. 3. The video memory should also be initialized at this point, to avoid random data appearing when the palette mask is set to make the current display PEL map contents visible.			
Sprite Control	36	00	Initial Value
Start Addr Low	40	00	Initial Value
Start Addr Me	41	00	Initial Value
Start Addr High	42	00	Initial Value
Display Pel Map Width Low	43	A0	As required
Display Pel Map Width High	44	00	As required
Display Mode 2	51	04	As required
Border Color	55	00	Initial Value
Palette Mask	64	FF	Make visible

Figure 3-190. DMQS Extended Graphics Mode Register Settings

Composite Display ID

The composite ID of the attached display is derived during POST, and is made available to the software in the DMQS primary data Area, as described in “DMQS BIOS Interface” on page 3-194.

Each display with unique function or characteristics and therefore a unique display information file has a unique display ID. The display presents the display ID through pins on the display connector. The details of its derivation are shown in “Display Type Detection” on page 3-213.

XGA Level Identifier

The XGA level identifier is returned as part of the BIOS Interface as described in “DMQS BIOS Interface” on page 3-194. The XGA level identifier field consists of two bytes. One is the functional level identifier, which identifies the level of the Display Controller chip. The next is the resolution level identifier, which identifies the level of the Serializer Palette DAC chip.

Functional Level Identifier Identifies the level of the Display Controller chip

- **'03'x** Base XGA implementation (XGA Display Adapter/A)
- **'05'x** XGA-NI implementation level of function

Resolution Level Identifier Identifies the level of the Serializer Palette DAC chip

- **'00'x** Base XGA implementation (XGA Display Adapter/A) (Maximum 45 MHz Pel rate)
- **'03'x** XGA-NI Serializer Palette DAC. (Maximum 90 MHz Pel rate)

DMQS Customisation File

An additional file in the DMQS file directory should be consulted to ascertain additional XGA customisation parameters, prior to reading the DMQS display configuration file. This file, the name of which is “XGASETUP.PRO,” (if present) contains system customisation information, as follows:

- Specifies a display ID alias for a particular slot. This overrides the display ID physically presented by the display, and specifies an alternate DMQS Display Configuration File name to be used instead.
- Identifies the slot to be used as the primary graphics display. In a multiple XGA system, the software normally chooses which XGA subsystem to use based on factors such as screen size, XGA subsystem functionality, etc.. This entry overrides the software default, and forces the nominated slot to be the primary XGA subsystem, rather than that chosen arbitrarily by the software.

DMQS Customisation File Example: Below is an example DMQS customisation file:

```
/*********************************************************************
/*
/* FILENAME: XGASETUP.PRO
/*
/* DESCRIPTION: Profile to set XGA DMQS preferences on current system.
/*
/* LAST MODIFIED: 04/09/92 at 16:13:38
/*
/*********************************************************************

/*********************************************************************
<SLOT, NUMBER=5, MONITOR_ID=F0FF>IBM 8515, 8516

/*********************************************************************
<SLOT, NUMBER=8, MONITOR_ID=EXMP> Special DMQS DIF file

/*********************************************************************
<STARTUP, NUMBER=5>
```

The profile above does the following:

1. Informs the software that the display in slot 5 is a display of type F0FF and the display information file MONF0FF.DGS should be used to obtain information about the installed display.
2. Informs the software that the display in slot 8 is a display of type “EXMP” and the display information file MONEXMP.DGS should be used to obtain information about the installed display.
3. Identifies the slot (5 in this example) which holds the XGA subsystem to be used as the primary graphics display.

Note: If any of the slot numbers and/or display IDs are invalid when read then the tag will be ignored and it shall not have any affect on how the XGA subsystem hardware is initialized.

XGA Subsystem DMQS Customisation Tags: The syntax for the generalised DMQS Customisation tag is as follows:

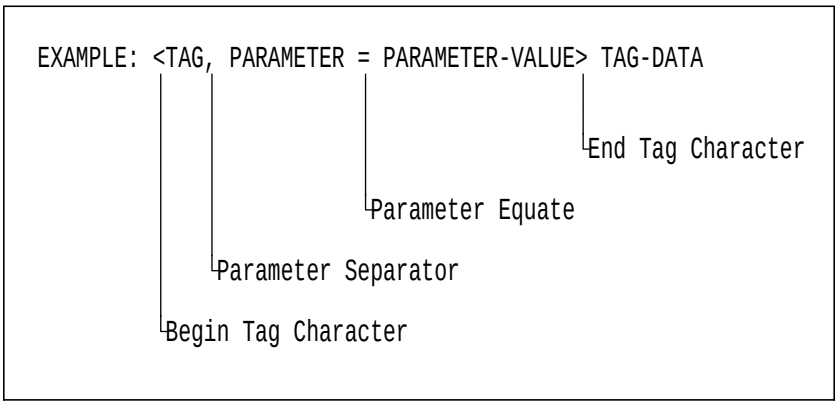


Figure 3-191. DMQS Customisation Tag Syntax

- Any number of blanks and/or new-lines can be used to separate the begin tag character, the end tag character, tags, parameters, and parameter values. So, you can flow a single tag with many parameter values over multiple lines and space the elements within the tag out for readability.
- The tag is the first text item following the begin tag character "<" and ending with the first comma "," or end tag character ">".
- If any parameters exist, then they are always separated by a comma ",". Otherwise, if no parameters exist for the tag, then the tag is immediately followed by an end tag character ">".
- Parameters are always assigned a value via the parameter equate character "=".

Individual tags are defined as follows:

SLOT Syntax Diagram:

```
<SLOT, NUMBER=[slot number],MONITOR_ID=[display ID]>Text
```

Parameters:

NUMBER=[**slot number**] **(REQUIRED)** An integer value that indicates the system slot number that the XGA subsystem occupies. If this is an invalid number or there is no XGA subsystem in the specified slot, then this tag will be ignored.

MONITOR_ID=[**display ID**] **(REQUIRED)** A four character alpha-numeric string that will be used to construct the name of the DMQS Display Configuration file to be loaded in place of the default file. This value will be used to generate a file name of the form MONXXXX.DGS where XXXX will be the display ID value specified in the tag.

Text (OPTIONAL) A comment field, that should be ignored by software.

Description This tag allows the user to specify a display ID override for display(s) attached to XGA subsystems with DMQS support. The display ID value in this customisation file will override the physical value read from the display by the XGA subsystem.

STARTUP Syntax Diagram:

```
<STARTUP, NUMBER=[slot number]>Text
```

Parameters:

NUMBER=[**slot number**] **(REQUIRED)** An integer value that indicates the system slot number that the XGA subsystem occupies. If this is an invalid number or there is no XGA subsystem in the specified slot, then this tag will be ignored.

Text (OPTIONAL) A comment field, that should be ignored by software.

Description This tag allows the user to specify which particular XGA subsystem is to be used by an XGA mode application, where the default chosen by the application is inconvenient.

Locating and Initialising the XGA Subsystem without DMQS

This section describes the original method for XGA subsystem identification and initialisation. Software should initially attempt to identify the XGA subsystem using DMQS, as described in “XGA Display Mode Query and Set (DMQS)” on page 3-192. Only when DMQS has been found to be not supported in the system, or if a

| DMQS display information file cannot be found, should software
| resort to the method of XGA subsystem identification and mode
| setting described in this section.

| Software should not attempt to use this method in addition to
| DMQS, as DMQS (if found) will provide support for both XGA and
| XGA-NI subsystems.

| The procedure is as outlined here, and more detail is given later in
| this section.

| 1. Identify if XGA subsystem is present by examining the system
| adapter's POS IDs one by one.

| 2. Locate the various I/O spaces of the XGA subsystem(s) spaces
| by decoding the XGA subsystem POS data.

| 3. Read the display ID to determine the attached Display type.

| 4. Determine the amount of VRAM installed on the XGA
| subsystem..

| 5. Determine the modes available on the attached display.

| 6. Set the XGA subsystem into the required XGA mode, either
| 640x480 or 1024x768 resolution.

| 7. Handle any VGA primary adapter considerations, as described
| in “VGA Primary Adapter Considerations” on page 3-219.

| This procedure should be repeated, if necessary, until sufficient
| XGA subsystems in the system have been identified.

| XGA Subsystem Identification

| To identify all XGA subsystems, put every adapter in the system,
| including the system board video subsystem, into setup mode in
turn, and examine their POS IDs to locate any XGA adapters in
the system.

For option cards, the procedure is described in System Services BIOS call INT 15h, AH=C4h Programmable Option Select in the *IBM Personal System/2 and Personal Computer BIOS Interface Technical Reference*.

For the system board video subsystem, a different procedure is necessary. To place the system board video subsystem in setup mode, write hex 0DF to port 94H; to enable it, write hex 0FF to port 94H.

Interrupts must be disabled for the entire period of time that each adapter is in setup mode.

The POS IDs for all adapters in the system must be read and examined to locate all the XGA adapters in the system.

The following POS IDs have been preallocated to the XGA subsystem and follow-on XGA register compatible subsystems:

- 8FD8h to 8FDBh inclusive
- 8FD0h to 8FD3h inclusive
- VESA reserved IDs, as follows:
  - 0240h to 027Fh inclusive
  - 0830h to 0A7Fh inclusive
  - 0A90h to 0BFFh inclusive

Check for these POS IDs when determining the existence of the XGA subsystem in the system.

After successfully matching POS IDs, read the remainder of the POS data bytes for that subsystem. This data is used to calculate the location of the XGA subsystem registers and display buffers in I/O and physical system memory address space. Descriptions of the POS data bit assignments are in “XGA POS Registers” on page 3-170. For future compatibility, mask out all reserved and unused POS data bits before using the data for these calculations.

Location of XGA subsystem I/O Spaces

See “XGA POS Registers” on page 3-170 for the technical background to the following register and address space calculations.

ROM Address: Calculate the ROM address from POS data as follows:

```
ROM Address = (ROM Address field × hex 2000) + hex 0C0000
```

The ROM Address field is read from POS Register 2, bits 4 to 7.

XGA Coprocessor Registers: The XGA coprocessor registers are referenced from a base address. This address depends on the Instance (0– 7) of the XGA subsystem and the ROM address calculated as shown in “ROM Address.” The Coprocessor register base address is calculated as follows:

```
(((128 × Instance) + hex 1C00) + ROM address)
```

The Instance is read from POS Register 2, bits 1 to 3.

For example:

Assuming Instance = 6 and ROM address = hex 0C0000, the Coprocessor Base Address is hex 0C1F00.

I/O Registers: The XGA I/O registers are referenced from a base I/O address. The I/O address is calculated as follows:

Hex 21x0 (where x is the Instance)

The Instance is read from POS Register 2, bits 1 to 3.

The Video Memory Base Address: The Video Memory Base Address is calculated from the Video Memory Base Address field in POS Register 4, and the Instance.

Figure 3-192 and Figure 3-193 on page 3-212 show how these two values combine to give the video memory base address.

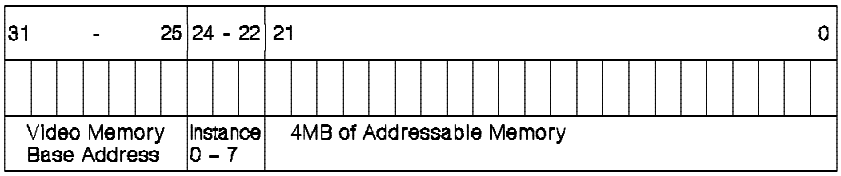


Figure 3-192. XGA Video Memory Base Address

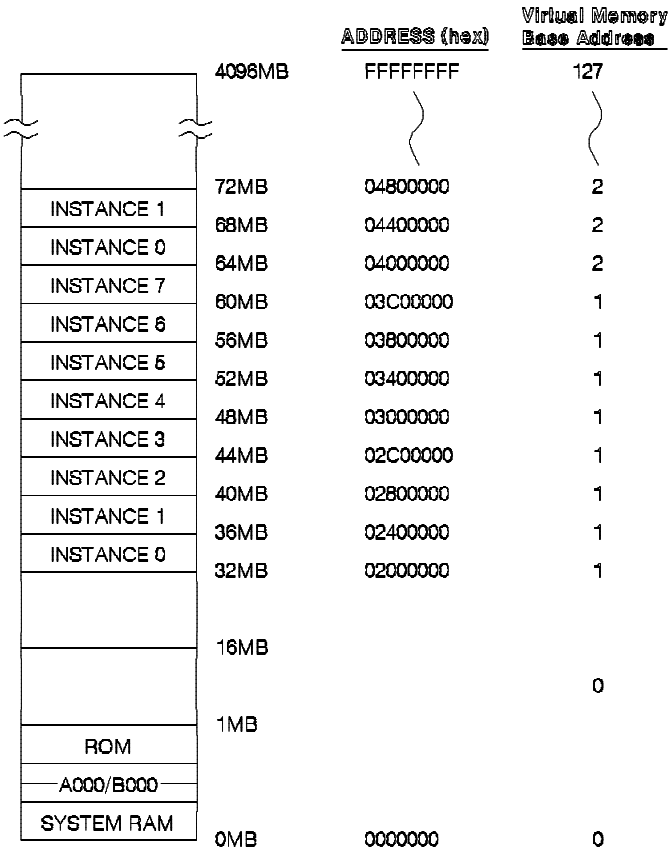


Figure 3-193. The XGA Video Memory Base Address Diagram

The Video Memory Base Address field defines a 32MB address range and the Instance defines a 4MB address range within the 32MB range.

For example:

Assuming Instance = 6 and the Video Memory Base Address field = 1, the Video Memory Base Address is hex 03800000.

The video memory base address, when calculated, serves two separate purposes:

4MB System Video Memory Aperture: If enabled (read from bit 0 in POS Register 4 to determine if the aperture is enabled), the 4MB system video memory aperture is located at this address in physical system address space. If virtual addressability to this range of physical address space can be achieved, the entire video memory can be accessed through this aperture at this address.

Video Memory Location in XGA Address Space: This address is used to identify video memory to the XGA coprocessor. It significance and use is described in "Video Memory Address Range" on page 3-235.

1MB Aperture Base Address: The 1MB aperture base address is calculated from the 1MB Aperture Base Address field in POS Register 5, bits 0 to 3.

```
If (1MB Base field ≠ 0)
     1MB Aperture Base Address = 1MB Base field × hex 100000

If (1MB Base field = 0) 1MB Aperture is disabled.
```

Display Type Detection

In order to determine what type of display is attached to the Video Subsystem, it is necessary to read the display's identification number, or ID. This ID is used to obtain information about the display such as: the resolutions supported, whether it is monochrome or color, and possibly the size of the screen.

The ID for each display is a 16 bit number, and usually uniquely identifies the display type. Some displays that have similar characteristics but are not the same model, have the same ID.

The recommended method of obtaining the display ID is by use of a BIOS call, Int 10h, (AH) = 1Fh - XGA Display Mode Query and Set (DMQS). See "XGA Display Mode Query and Set (DMQS)" on page 3-192. If it is necessary to read the display ID explicitly then the following procedure must be followed.

| The display ID is read from the “Display ID and Comparator” register, which returns four ID bits at a time. Four reads must be performed in order to obtain all sixteen bits. The components of the ID are selected by manipulating the values of Horizontal Sync and Vertical Sync that are output to the display. Therefore, the ID may only be read when disruption of these signals may be tolerated, such as power-on time, or when changing display modes.

| After setting the required “Sync Polarity” (SP field in Display Control 1 Register) to any of the various combinations of Horizontal and Vertical Sync listed below, it is necessary to wait for 15 micro-seconds for this change to take effect before display ID may be read. This is best achieved by doing five consecutive reads or writes to any byte wide XGA I/O port.

| The display ID Reading sequence is as follows:

| 1. Prepare the CRTC for reset (Display Control 1 Register - Index 50 - DB field = '01'b)
| 2. Reset the CRTC (Display Control 1 Register DB field = '00'b)
| 3. Set “Sync Polarity” (SP field in Display Control 1 Register) to '01'b This sets VSYNC to '0'b and HSYNC to '1'b.
| 4. After a 15 microsec. wait, read the display ID bits from Display ID and Comparator Register (Index 52). Place them in a hex variable A.
| 5. Set SP to '10'b This sets VSYNC to '1'b and HSYNC to '0'b.
| 6. After a 15 microsec. wait, read the display ID bits again. Place them in a hex variable B.
| 7. Set SP to '00'b This sets VSYNC to '0'b and HSYNC to '0'b.
| 8. After a 15 microsec. wait, read the display ID bits again. Place them in a hex variable C.
| 9. Set SP to '11'b This sets VSYNC to '1'b and HSYNC to '1'b.
| 10. After a 15 microsec. wait, read the display ID bits again. Place them in a hex variable D.

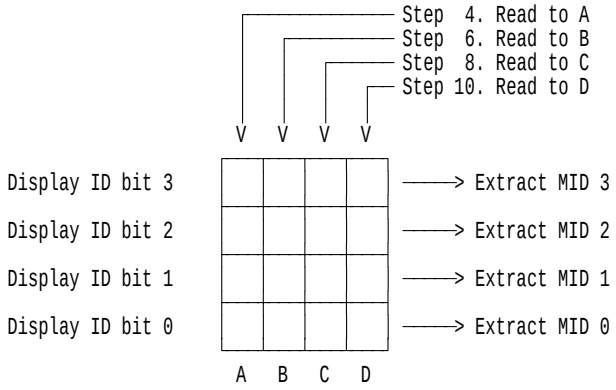
| Assemble the 16 bits into four nibbles, one for each MID pin, from MSB to LSB, as shown in Figure 3-194 on page 3-215. The resulting four-hex-digit number (from MID bit 3 to bit 0) is the display ID.

After reading four nibbles:

A				B				C				D			
ID3 (3)	ID2 (3)	ID1 (3)	ID0 (3)	ID3 (2)	ID2 (2)	ID1 (2)	ID0 (2)	ID3 (1)	ID2 (1)	ID1 (1)	ID0 (1)	ID3 (0)	ID2 (0)	ID1 (0)	ID0 (0)

where ID3 = Display ID 3, bit 1 etc.
(1)

Step:	3	5	7	9
H:	1	0	0	1
V:	0	1	0	1



After rearrangement:

Display ID 3				Display ID 2				Display ID 1				Display ID 0			
3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0
ID3 (3)	ID3 (2)	ID3 (1)	ID3 (0)	ID2 (3)	ID2 (2)	ID2 (1)	ID2 (0)	ID1 (3)	ID1 (2)	ID1 (1)	ID1 (0)	ID0 (3)	ID0 (2)	ID0 (1)	ID0 (0)

This is the 16 bit display ID

Figure 3-194. Reading the Display ID.

Figure 3-196 on page 3-217 shows a list of displays and their associated IDs.

Video Memory Size Determination

There are two ways to determine the size of video memory installed. Both rely on a write-readback-check, in which a particular value is written to a key location. This value is then read to determine whether the written value has persisted.

- Use the system processor to write a value through an aperture to the word at offset 768KB into video memory. This technique assumes that the system video memory real mode aperture is available. See the sample code in the following figure.

```
;* Assume GS points to start of A0000 Real mode aperture
;* and VGA adapter is in text mode so A0000 Real mode
;* aperture is available for this operation.
;* Where registers are shown as (for instance 21x0h), this should
;* be filled in with the appropriate IO port address after determining
;* the location of the XGA subsystem in IO space
;*
;* First put the adapter PARTIALLY in extended graphics mode
;* to allow use of the system video memory Aperture
      mov   al,0
      mov   dx,21x4h      ; disable XGA interrupts
      out   dx,al
;
      mov   ax,0064h
      mov   dx,21xAh      ; Blank palette
      out   dx,ax         ; indexed XGA register 64h
;
      mov   ax,04h
      mov   dx,21x0h      ; Set adapter in Extended Graphics Mode
      out   dx,al
;
      mov   al,01h
      mov   dx,21x1h      ; Locate video memory Aperture at A0000
      out   dx,al
;
      mov   dx,21x8h      ; System video memory indx reg.
      mov   al,0ch        ; Offset 768K
      out   dx,al         ;
;
      mov   byte ptr gs:[0],0A5h ; Set byte to A5h
      mov   byte ptr gs:[1],0h   ; Avoid shadows on data lines
;
      cmp   byte ptr gs:[0],0A5h  ; Test against value written
      jne   vram_512k     ; 512K video memory only
;
      mov   byte ptr gs:[0],5Ah ; Set byte to 5Ah
      mov   byte ptr gs:[1],0h   ; Avoid shadows on data lines
;
      cmp   byte ptr gs:[0],0A5h  ; Test against value written
      je    vram_1Meg      ; 1 Meg if still matches
      jmp   vram_512k      ; Otherwise 1/2 meg found
```

Figure 3-195. Video Memory Size Determination

- Use the XGA subsystem PxBlt capability to perform a test similar to the previous example. Transfer a constant color to the location in video memory, then transfer that value back from video memory to system memory using busmastership.

  This technique works regardless of the availability of a system video memory aperture. However, it requires physical addressability to a location in system memory for the busmastership operation.

Extended Graphics Modes Available

The following figure shows the list of modes available according to the display type and size of video memory configured on the XGA subsystem.

Composite Display ID	Display ID (Decimal)	Display Size (inch)	Color	Max Address	512KB Memory	1MB Memory
FFFF	None				None	None
FF0F	8503	12	Mono	640x480	640x480x64 Grays	640x480x64 Grays
FFF0	8513	12	Color	640x480	640x480x256 Colors	640x480x256 Colors
	8512	14				640x480x65536 Colors
	8518					
F0FF	8515	14	Color	1024x768	640x480x256 Colors	640x480x256 Colors
	8516				1024x768x16 Colors	640x480x65536 Colors
						1024x768x16 Colors
						1024x768x256 Colors
F00F	8604	15	Mono	1024x768	640x480x64 Grays	640x480x64 Grays
	8507	19			1024x768x16 Grays	1024x768x16 Grays
						1024x768x64 Grays
F0F0	8514	16	Color	1024x768	640x480x256 Colors	640x480x256 Colors
					1024x768x16 Colors	640x480x65536 Colors
						1024x768x16 Colors
						1024x768x256 Colors
90F0	8517	17	Color	1024x768	640x480x256 Colors	640x480x256 Colors
					1024x768x16 Colors	640x480x65536 Colors
						1024x768x16 Colors
						1024x768x256 Colors

Figure 3-196. Availability of Extended Graphics Modes

Extended Graphics Mode Setting Procedure

To set the XGA subsystem into Extended Graphics mode, the configuration must be capable of supporting the required mode as listed in “Extended Graphics Modes Available” on page 3-217.

XGA Register Name	XGA Reg. ID	Oper	Color Mode Value (hex) 1024x 768x 256	1024x 768x 16	640x 480x 256	640x 480x 65536	Comments
Interrupt Enable	21x4	=	00	00	00	00	Initial Value
Interrupt Status	21x5	=	FF	FF	FF	FF	
Operating Mode	21x0	=	04	04	04	04	Set Extended Graphics Mode
Palette Mask	64	=	00	00	00	00	Blank Display
Video Mem Aperture Ctl	21x1	=	00	00	00	00	Initial Value
Video Mem Aperture Index	21x8	=	00	00	00	00	Initial Value
Virt Mem Ctl	21x6	=	00	00	00	00	Initial Value
Memory Access Mode	21x9	=	03	02	03	04	Initial Value
Disp Mode 1	50	=	01	01	01	01	Prepare for reset
Disp Mode 1	50	=	00	00	00	00	Reset CRT Ctrl
Horiz Total Low	10	=	9D	9D	63	63	)
Horiz Total High	11	=	00	00	00	00	)
Horiz Display End Low	12	=	7F	7F	4F	4F	)
Horiz Display End High	13	=	00	00	00	00	)
Horiz Blank Start Low	14	=	7F	7F	4F	4F	)
Horiz Blank Start High	15	=	00	00	00	00	)
Horiz Blank End Low	16	=	9D	9D	63	63	)
Horiz Blank End High	17	=	00	00	00	00	)
Horiz Sync Start Low	18	=	87	87	55	55	)
Horiz Sync Start High	19	=	00	00	00	00	)
Horiz Sync End Low	1A	=	9C	9C	61	61	)
Horiz Sync End High	1B	=	00	00	00	00	)
Horiz Sync Posn	1C	=	40	40	00	00	)
Horiz Sync Posn	1E	=	04	04	00	00	)
Vert Total Low	20	=	30	30	0C	0C	)
Vert Total High	21	=	03	03	02	02	) XGA CRT
Vert Disp End Low	22	=	FF	FF	DF	DF	) Controller
Vert Disp End High	23	=	02	02	01	01	) param
Vert Blank Start Low	24	=	FF	FF	DF	DF	)
Vert Blank Start High	25	=	02	02	01	01	)
Vert Blank End Low	26	=	30	30	0C	0C	)
Vert Blank End High	27	=	03	03	02	02	)
Vert Sync Start Low	28	=	00	00	EA	EA	)
Vert Sync Start High	29	=	03	03	01	01	)
Vert Sync End	2A	=	08	08	EC	EC	)
Vert Line Comp Low	2C	=	FF	FF	FF	FF	)
Vert Line Comp High	2D	=	FF	FF	FF	FF	)
Sprite Control	36	=	00	00	00	00	Initial Value
Start Addr Low	40	=	00	00	00	00	Initial Value
Start Addr Me	41	=	00	00	00	00	Initial Value
Start Addr High	42	=	00	00	00	00	Initial Value
Buffer Pitch Low	43	=	80	40	50	A0	
Buffer Pitch High	44	=	00	00	00	00	
Clock Sel	54	=	0d	0d	00	00	
Display Mode 2	51	=	03	02	03	04	
Ext Clock Sel	70	=	00	00	00	00	
Display Mode 1	50	=	0F	0F	C7	C7	
Note: Initial Palette loading must be done at this point, by writing to the appropriate XGA subsystem palette/sprite registers. The video memory must also be initialized at this point, to avoid random data appearing when the palette mask is set to make the current display PEL map contents visible.							
Border Color	55	=	00	00	00	00	Initial Value
Palette Mask	64	=	FF	FF	FF	FF	Make visible

Figure 3-197. Extended Graphics Mode Register Settings

VGA Primary Adapter Considerations

Where a single XGA subsystem is providing both VGA and Extended Graphics function, particularly on a system with single display subsystem or display, an application using the subsystem in Extended Graphics mode takes on a number of additional systems responsibilities, particularly in the DOS environment.

Before switching the subsystem into Extended Graphics mode, examine the Operating Mode register, bits 0 and 2, to determine whether the XGA subsystem is enabled in VGA mode or 132-column text mode.

If the XGA subsystem is not enabled in VGA mode, the subsystem is operating as an auxiliary video subsystem and systems messages can be left to the primary VGA source. In this case, the XGA subsystem must not be put into VGA mode unless the current VGA is disabled.

If the XGA subsystem is enabled in VGA mode, the subsystem is the system primary video subsystem, and a number of special considerations apply.

Chaining the INT 10h Video BIOS Handler: The application must chain the INT 10h Video interrupt handler and display calls to the INT 10h handler while the application is using the XGA subsystem in Extended Graphics mode.

There are a number of hot-key and error handlers that may attempt to communicate with the VGA while the XGA subsystem is in Extended Graphics mode, so code must be written to handle such calls.

The majority of calls to the INT 10h handler can be ignored (simply return to the caller) while the XGA subsystem is in Extended Graphics mode, but some calls require correct handling.

(Ah)=00h Set Mode

Set mode calls can come from a critical error or nonmaskable interrupt (NMI) handler. Because failure to restore VGA mode can result in the loss of critical error data or dialogue, applications must allow the mode set operation.

For normal VGA mode setting procedure to occur, the INT 10h handler must restore the subsystem as

necessary to VGA mode before chaining on to the next INT 10h interrupt handler.

Note: The NMI handler traditionally issues a *Return Current Video State* to determine the current mode, followed by a *Video Set Mode* to the current mode.

If the INT 10h Video interrupt handler of the application detects a video set mode with AL = 7Fh, mode 03h should be substituted after restoring the subsystem to VGA mode.

(Ah)=0Fh Return current video state

In Extended Graphics mode, the application's INT 10h interrupt handler should return a current mode of 7Fh in AL, to indicate that the subsystem is in a non-VGA mode.

This is a special mode number assigned for this purpose.

INT 24h, Critical Error Handler: The application should trap and revector the DOS critical error handler interrupt vector (INT 24h), as described in the *DOS Technical Reference Manual*. The application is then notified on DOS Critical errors.

The critical error handler of the application should save the video state of the subsystem (as far as necessary), and put the XGA subsystem into VGA mode before chaining on, using the saved vector to the original critical error handler. This lets the dialogue between the critical error handler and the user proceed normally.

After returning from the chained critical error handler, the critical error handler of the application must examine the return code in AL to determine the appropriate action.

0, 1, 3 Control is returned to the application. Put the XGA subsystem back into Extended Graphics mode and restore the video state as necessary.

2 The program is aborted by the system. Leave the XGA subsystem in VGA mode and return.

Alternatively, the application can take over the entire critical error handling dialogue in Extended Graphics mode.

Note: The C language *signal* function can (in some implementations) be used to intercept the critical error handler for this purpose.

INT 23h, Ctrl-Break Exit Address: The application should trap and revector the DOS Ctrl-Break exit address interrupt vector (INT 23h), as described in the *DOS Technical Reference Manual*. The application is notified when the Ctrl-Break key combination is entered.

If the application is not otherwise intercepting Ctrl-Breaks, the XGA subsystem must be put back into VGA mode before chaining on, using the saved vector to the original Ctrl-Break handler. This lets the normal Ctrl-Break handler proceed.

Alternatively, the application can take over the entire Ctrl-Break handling in Extended Graphics mode.

Note: The C language *signal* function can be used to intercept the Ctrl-Break handler for this purpose.

INT 21h, Function 4Ch, Program Terminate Function: The subsystem must be in VGA mode on program termination, regardless of the how the program terminates or is terminated.

To ensure that this is done, the application must trap and revector the normal DOS program terminate function, DOS INT 21h function 4Ch, as described in the *DOS Technical Reference Manual*. On receiving notice of program termination, the application must put the subsystem back into VGA mode and unhook all other hooked interrupt vectors before chaining on for the remainder of program termination handling.

DOS INT 21h function 4Ch is the conventional method used by all programs to terminate. By trapping the DOS function interrupt (INT 21h) and monitoring calls to the program terminate function (4Ch), all routes for a program to terminate normally must be covered.

Note: There are other program terminate functions, including:

- INT 20h
- INT 27h
- INT 21h function 00h
- INT 21h function 31h.

For complete coverage, these calls can be revectored and trapped, but they are not used as commonly as the INT 21h function 4Ch.

All other function calls must be passed to the previous DOS function handler using the saved interrupt vector.

On detecting a call to function 4Ch, put the XGA subsystem into VGA mode before chaining on using the saved vector to the original DOS function handler. This lets the DOS program terminate function proceed normally.

Note: The C language *atexit* function can be used for this purpose.

Multiple XGA Subsystems
Up to eight XGA subsystems can be installed in a system.

Multiple XGA subsystems can coexist in Extended Graphics mode. Each instance occupies its own separate ranges of I/O and memory space. An application written to exploit multiple XGA subsystems in this mode can access each Instance of the subsystem without enabling and disabling the subsystem between accesses.

To comply with the restriction on VGA coexistence described in "XGA Adapter Coexistence with VGA," a multiple display subsystem application must record, on initialization, the XGA subsystem (if any) originally in VGA mode. On application termination, only *that* subsystem should be returned to VGA mode.

VGA Modes

Where the XGA subsystem is being used as a standard VGA, then the XGA subsystem is VGA compatible, and mode setting should be performed using the normal Int 10h Video BIOS services.

Where the XGA subsystem is being used in XGA mode, or coexisting with a VGA, or other XGA subsystems in VGA mode, then this section includes information on mode switching from XGA to VGA mode, and guidelines on VGA coexistence.

On switching between XGA and VGA modes, and also between some VGA modes, contents of video memory may be lost or re-ordered. Information on this in included in "Effects of VGA & XGA Mode Setting on Video Memory" on page 3-229.

XGA Adapter Coexistence with VGA

Because the VGA uses fixed I/O and memory mapped address spaces, only one VGA can be active at a time in a system. When the XGA subsystem is installed alongside a VGA or another XGA subsystem, only one of the VGA-capable subsystems can be enabled at one time. Software must not switch the XGA subsystem from XGA mode to VGA mode if another VGA adapter is already in VGA mode.

An application can use multiple coexisting VGA or XGA subsystems in VGA mode only by alternately disabling and enabling the various VGAs.

Do not enable more than one VGA concurrently. A disabled (or inactive) VGA retains its visible displayed data, and the overall effect is that of a multiple VGA application.

To successively enable and disable multiple coexisting XGA subsystems in VGA mode, use the Operating Mode register (21x0).

Switching the XGA subsystem from XGA to VGA Mode

To put either the XGA subsystem or the XGA-NI subsystem back into VGA mode (subject to the rules discussed in "VGA Primary Adapter Considerations" on page 3-219), perform the following operations:

1. Clear the first 256KB of video memory contents. This avoids screen flash caused by random data being present on switching into VGA mode.
2. Write data to the registers in the following sequence:

Value (hex)	Oper	XGA Reg	VGA Reg	Comments
00	=	21x1		Aperture Control register
00	=	21x4		Interrupt disable
FF	=	21x5		Clear interrupts
FF	=	64		Palette Mask register
15	=	50		Enable VFB, prepare for reset
14	=	50		Enable VFB, reset CRT controller
00	=	51		Normal scale factors
04	=	54		Select VGA oscillator
00	=	70		External Clock (VGA)
20	=	2A		Ensure no VSync interrupts
01	=	21x0		Switch to VGA mode
01	=		3C3	Enable VGA address decode

Figure 3-198. VGA Mode Write Sequence

3. Set the number of lines in VGA mode (if required) using Video BIOS INT 10h AH=12h.
4. Set the required VGA mode using Video BIOS INT 10h AH=00h, set mode.

The XGA subsystem is now in VGA mode.

Smooth Scrolling of VGA and 132 Column Text Modes

Smooth vertical scrolling of both VGA modes (text and graphics) and the XGA subsystem 132 column text modes is possible by manipulation of the following VGA registers.

Horizontal Pel Panning Register See “Horizontal PEL Panning Register” on page 2-95 for a detailed description.

This register is used in horizontal smooth scrolling to move the visible data within a byte or character. This register is incremented or decremented until its limit is reached. This register is then reset to its start value, and the Start Address High and Low Registers are incremented or decremented by 1 unit.

Preset Row Scan Register See “Preset Row Scan Register” on page 2-63 for a detailed description.

This register is used in vertical scrolling in text modes only. The Starting Row Scan Count field is used to offset the visible display buffer start vertically within a row of text mode characters.

This register is incremented or decremented until its limit (defined by the text mode character box height) is reached. This register is then reset to its start value, and the Start Address High and Low Registers are incremented or decremented to the start of the next row of characters.

Start Address High and Low Registers See “Start Address High Register” on page 2-67 for a detailed description.

For horizontal scrolling this register is incremented or decremented by one byte or 1 character at a time, whenever the Horizontal Pel Panning register limit is reached.

For vertical scrolling this register is incremented by one scan line of pels or characters whenever the Preset Row Scan Register limit is reached.

Input Status Register 1 See “Input Status Register 1” on page 2-45 for a detailed description.

The Vertical Retrace status bit is polled to determine when the XGA subsystem is in the vertical retrace interval. Scrolling must be synchronised with this bit to avoid screen flash and other visible screen effects.

| The exact sequence of operations is different in VGA modes from
| the XGA subsystem 132 column text mode.

| When smooth scrolling in VGA modes, the following sequence is
| used:

| 1. Wait for Vertical Retrace start ('1'b')
| 2. Wait for Vertical Retrace end ('0'b')
| 3. Update Start Address and Horizontal Pel Panning Registers as
| required
| 4. Wait for Vertical Retrace start ('1'b')
| 5. Update Preset Row Scan Register as required

| For smooth scrolling in XGA subsystem 132 column text mode, the
| following sequence is used:

| 1. Wait for Vertical Retrace start ('1'b')
| 2. Wait for Vertical Retrace end ('0'b')
| 3. Update Pel Panning Register as required
| 4. Wait for Vertical Retrace start ('1'b')
| 5. Update Start Address and Preset Row Scan Registers as
| required

| 132-Column Text Mode

| The XGA-NI subsystem supports 132 column text mode as Int 10h
| Video BIOS Mode 14h. The original XGA subsystem did not
| generally have BIOS support for 132 column text mode.

| Software should query Int 10h Video BIOS for the existence of the
| mode by issuing a Video BIOS INT 10h Return Functionality State
| Information call, and examining the list of BIOS supported modes
| for the existence of mode 14h.

| If mode 14h is supported in BIOS, issue the appropriate Video
| BIOS Set Mode call to put the subsystem into 132-column text
| mode. Where BIOS support is provided, it must be used to exploit
| the higher refresh rates available on displays attached to the
| XGA-NI subsystem, or to access the nine pel wide character sets
| available only on the XGA-NI subsystem. See Video BIOS in *IBM
| Personal System/2 and Personal Computer BIOS Interface
| Technical Reference* for details.

If Video BIOS Mode 14h is not supported in BIOS, the following sequence of operations puts the subsystem into 132-column text mode using 8 pel wide characters:

1. If necessary, put the XGA subsystem into VGA mode.
2. Write data to registers in the following sequence:

Value (hex)	Oper	XGA Reg	VGA 3D4/5	VGA 3C4/5	Other VGA	Comments
15	=	50				Prepare CRT controller for reset
14	=	50				Reset CRT controller
04	=	54				Select VGA clock

Figure 3-199. 132-Column Text Mode First Write Sequence

3. Set the number of lines in VGA mode using INT 10h AH=12h (200, 350, or 400).
4. Set VGA mode 3 using INT 10h AX=0003h. The 132-column text mode is a variation of the VGA text mode. The following table gives the variations from the standard mode.

5. Write data to registers in the following sequence:

Value (hex)	Oper	XGA Reg	VGA 3D4/5	VGA 3C4/5	Other VGA	Comments
01	|=	50				) Prepare CRT controller for reset
FD	& =	50				)
FC	& =	50				Reset CRT controller
03	=	21x0				132-column text mode
01	=	54				132-column clock frequency select
80	=	70				Select internal 132-column clock
EF	& =	50				Disable video extension
7F	& =		11			Enable VGA CRT controller reg update
A4	=		0			)
83	=		1			)
84	=		2			)
83	=		3			)
90	=		4			) Variations on VGA CRT
80	=		5			) controller syncs
A3	=	1A				)
00	=	1B				)
9F	& =	1C				)
F9	& =	1E				)
42	=		13			)
80	|=		11			Disable VGA CRT controller reg update
03	|=	50				Remove CRT controller reset
01	|=			01		8 bit characters
**	INP				3DA	Read sets attribute controller flip flop
13	=				3C0	) Sets attribute controller
00	=				3C0	) Registers hex 13 to 00
20	=				3C0	Restore palette

|= Logical OR to modify register contents

& = Logical AND to modify register contents

= Write value to register

INP Read value from register

Figure 3-200. 132-Column Text Mode Second Write Sequence

6. Write hex 84 to 40:4A in BIOS data area to force Video BIOS recognition of 132-column text mode.

The coded text buffer is now 132 columns wide. The mode can now be programmed like any other VGA text mode, with a coded text buffer located at hex B8000 in system address space.

If it is necessary to invoke a mode change using Video BIOS (INT 10h) while in 132-column text mode (for example, to vary the number of lines), follow step 1 on page 3-227 through step 6.

Effects of VGA & XGA Mode Setting on Video Memory

Software that relies on Int 10h Video BIOS mode setting to switch between VGA modes will be protected from the effects described in this section. For *all* software that switches between XGA modes and VGA modes, or sets VGA modes without using Int 10h Video BIOS, the following points should be observed when switching between modes, (see “Hardware Considerations” on page 2-11):

- All data in video memory is preserved during a mode switch, provided that the CRT controller is halted at the time using the Display Control 1 register (if switching out of Extended Graphics mode), or the Reset register (if switching out of VGA mode). The CRT controller is described in “CRT Controller” on page 3-23 and “Display Control 1 Register (Index 50)” on page 3-72.
- When switching between VGA modes, the mapping of the VGA memory maps to the video memory is controlled by 2 fields in the following VGA CRT Controller registers:
  - Word/byte mode (CRT Mode Control register, WB field)
  - Doubleword mode (Underline Location register, DW field).

  VGA modes can be split into three groups: byte modes, word modes, and doubleword modes.

  All switches between modes in the same group are indistinguishable from the same mode switches on the VGA.

  Switches between modes in different groups produce different effects from those observed on the VGA. Because the bits controlling the mapping are used for display purposes, the picture is scrambled in both cases.

  Partial mode switches (for example, to load fonts in a text mode) are also possible. The bits used to control the mapping of the data in video memory are used to control the picture display. Therefore, all partial mode switches to update the video memory that do not destroy the picture (and many that do) work correctly.

Programming the XGA subsystem

General Systems Considerations

Coexisting with LIM Expanded Memory Managers

The XGA subsystem uses memory-mapped registers located in the hex C0000/D0000 region of physical address space, as described in “XGA POS Registers” on page 3-170.

This area is used extensively by expanded memory managers to provide expanded memory services to applications.

When the application determines the location of the memory-mapped register space, it must interrogate any expanded memory manager to ensure that there is no contention for this range of physical address space. Use function 25 (Ah)=58h, get physical address array, as described in the *Lotus/Intel/Microsoft Expanded Memory Specification Version 4.0*. If there is contention, a warning should be issued advising the user to resolve it by use of the expanded memory manager call parameters (usually on the DEVICE= statement in CONFIG.SYS).

INT 2Fh, Screen Switch Notification

For the application to work successfully in multiple virtual DOS machine (MVDM) environments, or in the DOS compatibility box of the OS/2 operating system, it must trap and revector the DOS multiplex vector (2Fh), looking for (Ah)=40h. Any other values must be passed immediately to the chained INT 2Fh handler. This multiplex interrupt is used with (Ah)=40h to notify DOS applications of screen switches.

(Al)=01h DOS mode application being switched to the background.

The application must save its video state and put the display back into VGA mode (if applicable).

(Al)=02h DOS mode application being switched to the foreground.

The application can switch the subsystem back into Extended Graphics mode, and restore the saved video state.

The range of operations permitted within INT 2Fh processing is limited. For example, it is not permissible to issue disk I/O operations, precluding an entire save and restore of video memory and state. The only way of using this call is for the INT 2Fh interrupt handler to notify the application that a redraw is required (if the application program structure permits).

PS/2 System Video Memory Apertures

The XGA subsystem provides three possible apertures or windows to video memory in the physical memory address space of the system. If present, any of them can be used by the system processor to access directly the packed PEL display buffer. However all 3 apertures have drawbacks, as follows:

64K Aperture Located at A0000 or B0000 in real mode address space.

- Possible contention with VGA
- Possible contention with other XGA subsystems
- Limited size of Aperture (64K : 1Meg Video Buffer) necessitates frequent movement of aperture
- Granularity of aperture movement (64K minimum)

1 Meg Aperture Located below 16 Meg (above 1 Meg) in protect mode system address space.

- May not be enabled where 16 Meg of Memory is installed in system
- Only accessible by protect mode drivers.

4 Meg Aperture Located above 16 Meg in protect mode 32 bit address space

- Not available in 16 bit systems, such as those based on the i386SX processor
- Not available if the XGA subsystem is plugged into a 16 bit slot in a 32 bit system
- Only accessible by 32 bit protect mode drivers

As the XGA subsystem coprocessor make the use of apertures unnecessary, their use is not recommended. If software does require the use of apertures, the following considerations apply:

- Check the availability of the aperture before using it.
- Build flexibility into the software, to be able to use whichever aperture is available, including the 64K aperture.
- Consider informing the application user that the XGA subsystem must be installed in a 32-bit slot on a 32-bit system if it is found not to be.
- The XGA-NI subsystem 1 Meg aperture may be disabled by default by the System Setup program. Software may need to instruct users to run System Setup again to enable this aperture.

- Inform the user that system memory may need to be removed to permit the 1 Meg aperture to be enabled, where system memory does not permit its enablement.
- If using the real mode 64K aperture, be aware of contention with any VGA or other XGA subsystems.

The precise location of each aperture, and whether it is enabled, is returned as part of the DMQS primary Data, as described in "XGA Display Mode Query and Set (DMQS)" on page 3-192. If DMQS is not available, this can be determined by decoding the XGA subsystem POS data, as described in "Locating and Initialising the XGA Subsystem without DMQS" on page 3-208.

64KB System Video Memory Aperture

This aperture is at hex A0000 or B0000 in physical address space. The 64KB aperture is insufficient to access the entire subsystem display buffer. Therefore, the aperture position over the display buffer is controlled by using the Aperture Index register.

This is the only aperture in i86 real mode address space.

Other video adapters, such as another adapter or subsystem in either VGA or Extended Graphics mode, may contend for the use of this aperture. Only one video subsystem can have this aperture enabled at a time. If there is no contention for the hex A0000 or B0000 address spaces, this aperture is the only one that can be directly enabled by the application.

1MB System Video Memory Aperture

The base address of this aperture may be located on any 1MB boundary from 1MB to 15MB, or it may be disabled. This aperture is located and enabled at System Setup. In the case of the XGA-NI subsystem, this aperture may be disabled by default, in which case the user should manually enable it using Setup. Software may need to publish instructions to this effect. The aperture address is determined by the system configuration. To determine its position, and whether it is enabled, decode the POS data as described in "Locating and Initialising the XGA Subsystem without DMQS" on page 3-208.

In systems with multiple XGA subsystems, each one may have its own aperture. Depending on the hardware configuration, it is

possible for some, but not all coexisting XGA subsystems to have their 1MB system video memory apertures enabled.

This aperture is large enough to access the entire video memory without using the Aperture Index register to move the aperture. The Aperture Index register must be set to 0 when using this aperture.

This aperture is easily accessible only in protect mode environments. The operating system must provide addressability to the address range occupied by the aperture. Some operating systems attempt to restrict such addressability to protect device drivers or kernel device drivers. A small kernel device driver may need to be written to provide addressability. For example, in a 16-bit segmented system such as the OS/2 version 1.3 operating system, the following steps may be necessary to build global descriptor table (GDT) addressability to an aperture:

1. Allocate a GDT selector.
2. Modify the GDT entry directly to alter the permission bits to allow user mode (ring 3) access.
3. Alter the GDT segment length to be a 1MB segment. The entire 1MB video memory display buffers can then be accessed as a single segment.

Check that the aperture is enabled before assuming its existence. In systems with a full 16Mbytes of memory, this aperture may not be enabled. If the aperture is disabled, it cannot be enabled by the application. The application should then try to use the 4MB aperture.

4MB System Video Memory Aperture

The base address of this aperture may be located on any 4 megabyte boundary at or above 16MB, or it may be disabled. The aperture address is determined by the system configuration. To determine its position, and whether it is enabled, decode the POS data as described in “Locating and Initialising the XGA Subsystem without DMQS” on page 3-208.

In systems with multiple XGA subsystems, each one may have its own aperture.

This aperture is not available in 16-bit systems based on the 80386SX. This aperture does not exist when the XGA subsystem adapter is plugged into a 16-bit (short) slot on a 32-bit system.

Check that the aperture is enabled before assuming its existence. Also, check the Auto-Configuration register, as described in “Auto-Configuration Register (Index 04)” on page 3-48, to determine the bus width.

While this aperture is present when the XGA subsystem is plugged into a 32-bit slot on a 32-bit system, it may not be easily accessible in real-mode DOS or 16-bit protect-mode operating systems.

Video Memory Address Range

The video memory base address is returned in the DMQS primary data area, or calculated from POS settings on XGA subsystems without DMQS, as described in “Location of XGA subsystem I/O Spaces” on page 3-210.

The video memory address range is defined as the range of addresses starting at the video memory base address, with length equal to the video memory size. The video memory address range has a special significance to the XGA coprocessor. It defines the location of the video memory, including the display PEL map, in the XGA coprocessor's view of system address space. Therefore, the XGA coprocessor recognizes addresses in this range to be addresses in local video memory, rather than general system memory. This is how the XGA coprocessor differentiates video memory from system memory. If an address passed to the XGA coprocessor is in this range, the XGA coprocessor knows that it is operating on a bit map in video memory. If the address is outside this range, the XGA coprocessor assumes it is operating on a bit map in normal system memory, and attempts to use busmastership to access it.

The XGA subsystem operates internally on a 32-bit bus. Therefore, this address is a 32-bit address regardless of whether the XGA subsystem is installed in a 16- or 32-bit slot or system, or whether the 4MB system video memory aperture is enabled or disabled. This applies even on systems where such addresses are not otherwise possible.

Programming the XGA Subsystem in Extended Graphics Mode

This section describes and gives examples of using Extended Graphics functions of the XGA coprocessor.

General Register Usage

To avoid conflicts with possible future changes in the use of registers or register fields, applications must comply with the Register Usage Guidelines at the start of the various register definition sections.

XGA Coprocessor PEL Interface Registers

Extended graphics functions are graphics update operations involving up to four PEL maps. A PEL map is defined by five registers:

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

PEL Map Index Register: This register has an offset of hex 12. The PEL Map Index register defines which of the four possible maps is to be defined. The encoding of this 4-bit register is as follows:

Mask map	hex 0
PEL map A	hex 1
PEL map B	hex 2
PEL map C	hex 3

For example, to use PEL map A:

```
WRITE 01h to copr_regs offset 12h.
```

PEL Map Base Address Register: This register has an offset of hex 14. The PEL Map Base Pointer register defines the byte address in memory of the start of the PEL map. It is a 32-bit address register and can therefore address up to 4096MB of memory. A PEL map can be defined to be in the XGA video memory or in system memory.

As described in "Video Memory Address Range" on page 3-235, to define a PEL map as being in XGA video memory, the address put in this register must be in the following range:

```
Video Memory Base Address ◄► (Video
Memory Base Address + Video Memory size)
```

If the PEL map is in system memory and the Micro Channel interface is a 16-bit interface (for example, if the XGA adapter is installed in a 16-bit slot), the address of the map must be below 16MB.

PEL Map Width Register: This register has an offset of hex 18. The PEL map width is measured in PELs and is defined as one less than the required width.

For example, to set the width of a PEL map to 640 PELs:

```
WRITE 027Fh to copr_regs offset 18h
```

To set the width of a PEL map to 1024 PELs:

```
WRITE 03FFh to copr_regs offset 18h
```

PEL Map Height Register: This register has an offset of hex 20. The PEL map height is measured in PELs and is defined as one less than the required height.

For example, to set the height of a PEL map to 480 PELs:

```
WRITE 01DFh to copr_regs offset 20h
```

To set the height of a PEL map to 768 PELs:

```
WRITE 02FFh to copr_regs offset 20h
```

PEL Map Format Register: This register has an offset of hex 1C. This register specifies the bits per PEL of the PEL map. The encoding of the register is as follows:

1 bit/PEL Intel format	hex 00
2 bits/PEL Intel format	hex 01
4 bits/PEL Intel format	hex 02
8 bits/PEL Intel format	hex 03
16 bits/PEL Intel format	hex 04
1 bit/PEL Motorola format	hex 08
2 bits/PEL Motorola format	hex 09
4 bits/PEL Motorola format	hex 0A
8 bits/PEL Motorola format	hex 0B
16 bits/PEL Motorola format	hex 0C

Note: Values hex 04 and hex 0C are only valid on the XGA-NI subsystem.

For example, for an 8-bit/PEL Motorola format PEL map:

```
WRITE 0Bh to copr_regs offset 1Ch
```

The relationship between Intel and Motorola format PEL maps is discussed in “Video Memory Format” on page 3-20 and “Motorola and Intel Formats” on page 3-258.

All four PEL maps (A, B, C, and mask) can be initialized in this manner to be ready for later use. Maps A, B, and C can be used interchangeably as the source, destination, or pattern in all subsequent PEL operations.

Other Registers: For simple operations, the PEL Interface Control register must be cleared.

For example:

```
WRITE 00h  to copr_regs offset 11h
```

For simple operations, the Destination Color Condition Compare register must be set so that it has no effect on the operation.

For example:

```
WRITE 04h  to copr_regs offset 4Ah
```

To allow all planes of a PEL map to be updated, the PEL bit mask must be turned on. That is, set all bits to 1 that are required for the PEL size selected.

For example, for 8-bits-per-PEL:

```
WRITE 00FFh to copr_regs offset 50h
```

For simple operations, the carry chain mask must be turned on. That is, set all bits to 1 that are required for the PEL size selected.

For example, for 8-bits-per-PEL:

```
WRITE FFh to copr_regs offset 54h
```

Using the Coprocessor to Perform a PEL Blit (PxBlt)

This section describes the actions necessary to use the XGA coprocessor to perform a simple PxBlt.

Various types of PxBlt can be performed. This example is for a PxBlt into video memory using the Foreground Color register as the source data. The result is a solid rectangle drawn into the display PEL map. The example PxBlt has the following characteristics:

- Foreground color of hex 05
- 100 PELs wide and 60 PELs deep
- Positioned at screen coordinates X=200 and Y=150.

The following table lists the values that must be written to the coprocessor registers. Each value is explained following the table, along with information on the other forms of PxBlt available.

Value (hex)	Coprocessor Registers Offset (hex)
03	48
05	58
0063	60
003B	62
00C8	78
0096	7A
08118000	7C

Figure 3-201. Coprocessor Register Write Values

Mixes and Colors: Before a coprocessor operation can be performed, the background and foreground mixes have to be set. Mixes are logical or arithmetic functions performed on the source and destination data when performing a coprocessor operation. The mix functions available are as follows:

Code	Function
0	Zeros
1	Source AND Destination
2	Source AND NOT Destination
3	Source
4	NOT Source AND Destination
5	Destination
6	Source XOR Destination
7	Source OR Destination
8	NOT Source AND NOT Destination
9	Source XOR NOT Destination
A	NOT Destination
B	Source OR NOT Destination
C	NOT Source
D	NOT Source OR Destination
E	NOT Source OR NOT Destination
F	Ones
10	Maximum
11	Minimum
12	Add with Saturate
13	Subtract (Destination – Source) with Saturate
14	Subtract (Source – Destination) with Saturate
15	Average

Figure 3-202. Background and Foreground Mixes and Colors

Foreground and Background Mix Registers: The mixes to be applied to foreground and background PELs are specified in these two registers. The contents of the pattern map determine the PELs for foreground and background. In this example, the PxBlt is solid and contains only foreground PELs. The Foreground Mix register must be set to Source to give an understandable result on the screen.

For the example:

```
WRITE 03h  to copr_regs offset 48h
```

Foreground and Background Color Registers: The colors to be used for foreground and background PELs are specified in these two registers. In this example, the PxBlt is solid and only the Foreground Color register needs to be set up.

For the example:

```
WRITE 05h to copr_regs offset 58h
```

Other forms of PxBlt (for example, video memory to video memory) from a source map into a destination map do not use these color registers.

PxBlt Dimensions: The Operation Dimension 1 register must be loaded with the Width of PxBlt to be performed. The value loaded into the register must be one PEL less than the required width (in PELs).

For example, for a 100-PEL wide Pxblt:

```
WRITE 0063h to copr_regs offset 60h
```

The Operation Dimension 2 register must be loaded with the Height of PxBlt to be performed. The value loaded into the register must be one PEL less than the required height (in PELs).

For example, for a 60-PEL high Pxblt:

```
WRITE 003Bh to copr_regs offset 62h
```

PEL Map, Source, and Destination Registers

Source Map X and Y Registers: The source map is initialized as detailed in “Mixes and Colors” on page 3-241. Within the source map, two registers exist that contain the X and Y offset positions of the start of the source data for a PxBlt. These registers are used when performing a PxBlt using a source map. In this example, these registers are unused.

Destination Map X and Y Registers: The destination map is initialized as detailed in “Mixes and Colors” on page 3-241. Within the destination map, two registers exist that contain the X and Y offset positions of the start of the PxBlt.

For the example, to position the PxBlt at X=200 and Y=150 in the destination map:

```
WRITE 00C8h to copr_regs offset 78h (Destination Map X position)

WRITE 0096h to copr_regs offset 7Ah (Destination Map Y position)
```

Pattern Map X and Y Registers: The pattern map is initialized as detailed in “Mixes and Colors” on page 3-241. Two registers exist that contain the X and Y offset positions, within the pattern map, of the start of the pattern data for a PxBlt. These registers are used when performing a PxBlt using a pattern map.

In this example these registers are unused.

Mask Map Origin X and Y Offset Registers: The mask map is initialized as detailed in the previous chapter. Two registers exist that contain the X and Y offset positions of the start of the mask map relative to the top left corner of the destination map. These registers are used when performing a PxBlt using a mask map.

In this example these registers are unused.

PEL Operations Register: This is a 32-bit register that defines the operation the coprocessor performs.

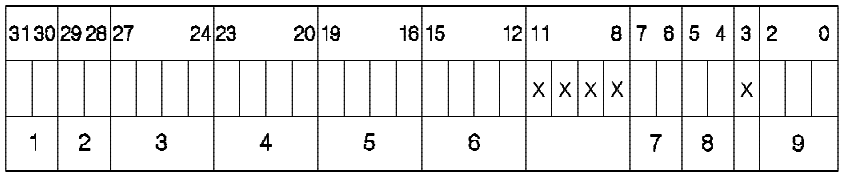


Figure 3-203. Bit Layout PEL Operations Register

The definition of the fields at the bottom of Figure 3-203 are:

1. Background Source
2. Foreground Source
3. Step Function
4. Source PEL Map
5. Destination PEL Map
6. Pattern PEL Map
7. Mask PEL Map
8. Drawing Mode
9. Direction Octant.

These fields, described in sequence, are required to assemble the PEL Operations register:

Background Source: These bits determine the origin of the background source PELs when an operation is performed.

The encoding for these bits is as follows:

Background Color	Binary 00 (for example, for a fixed register value to video memory PxBlt).
Source PEL Map	Binary 10 (for example, for a video memory to video memory PxBlt).

For this example, there is no background color, and the field is ignored.

```
Background Source = Binary 00
```

Foreground Source: These bits determine the origin of the foreground source PELs when an operation is performed.

The encoding for these bits is as follows:

Foreground Color	Binary 00 (for example, for a fixed register value to video memory PxBlt).
Source PEL Map	Binary 10 (for example, for a video memory to video memory PxBlt).

For this example there is a solid foreground color:

```
Foreground Source = Binary 00
```

Step Function: These bits define the type of operation that the coprocessor is required to do.

The encoding for these bits is as follows:

Draw and Step Read	Binary 0010
Line Draw Read	Binary 0011
Draw and Step Write	Binary 0100
Line Draw Write	Binary 0101
PxBlt	Binary 1000
Inverting PxBlt	Binary 1001
Area Fill PxBlt	Binary 1010

For this example:

```
Step Function = binary 1000
```

Source PEL Map: These bits define the PEL map used as the source map in the operation. This enables different maps to be setup in advance and defined for use as this register is loaded.

The encoding for these bits is as follows:

PEL Map A	Binary 0001
PEL Map B	Binary 0010
PEL Map C	Binary 0011

For this example, the contents of this field are ignored but they must not be a reserved value.

```
Source PEL Map = Binary 0001
```

Destination PEL Map: These bits define the PEL map used as the destination map in the operation. This enables different maps to be setup in advance and defined for use as this register is loaded.

The encoding for these bits is as follows:

PEL Map A	Binary 0001
PEL Map B	Binary 0010
PEL Map C	Binary 0011

For this example:

```
Destination PEL Map = Binary 0001
```

Pattern PEL Map: These bits define the PEL map used as the pattern map in the operation. This enables different maps to be setup in advance and defined for use as this register is loaded.

The encoding for these bits is as follows:

PEL Map A	Binary 0001
PEL Map B	Binary 0010
PEL Map C	Binary 0011
Foreground (fixed)	Binary 1000
Generated from Source	Binary 1001

For this example:

```
Pattern PEL Map = binary 1000
```

Mask PEL Map: These bits define whether the mask map is used in the operation.

The encoding for these bits is as follows:

Mask Map Disabled	Binary 00
Mask Map Boundary Enabled	Binary 01
Mask Map Enabled	Binary 10

For this example:

```
Mask PEL Map = Binary 00
```

Drawing Mode: These bits concern line drawing only and are discussed later. They are ignored during a PxBlt.

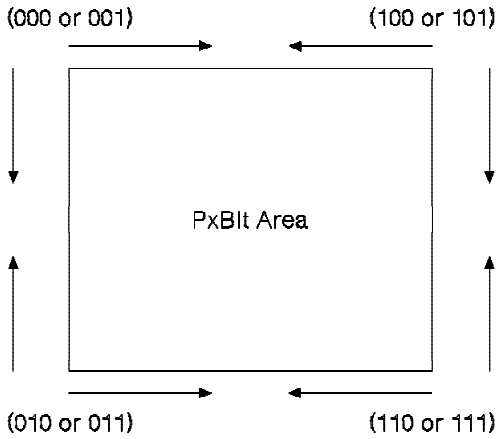
For this example:

```
Drawing Mode = Binary 00
```

Direction Octant: These bits, when concerned with PxBlts, determine the direction in which the PxBlt is drawn.

The encoding for these bits is as follows:

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



Note: Numbers are binary.

Figure 3-204. Operation Direction Diagram

These bits are normally set to binary 000, but other values are necessary to avoid PEL corruption when source and destination rectangles overlap.

For this example the PxBlt is in top left corner:

```
Direction Octant = Binary 000
```

Conclusion: Putting all these together for the PxBlt, the PEL Operations register must be set as:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	X	X	X	X	0	0	0	0	X	0	0	0

Figure 3-205. Definition for PEL Operations Register (Example)

For this example:

```
WRITE 08118000h to copr_regs offset 7Ch
```

Using the Coprocessor to Perform a Bresenham Line Draw

The steps required to draw a line of palette color hex 05 from (20,15) to (80,35), are summarized in the following table. The sections that follow explain each value and provide information on the other line drawing options available.

Value	Coprocessor Registers Offset (hex)
Hex 03	48
Hex 05	58
Decimal – 20	20
Decimal 40	24
Decimal – 80	28
Decimal 60	60
Decimal 20	78
Decimal 15	7A
Hex 05118000	7C

Figure 3-206. Palette Color Line Draw Steps

Mixes and Colors: Before a coprocessor operation is performed, the background and foreground mixes have to be set. Mixes are logical or arithmetic functions performed on the source and destination data when performing a coprocessor operation. The mix functions available are as follows:

Code	Function
0	Zeros
1	Source AND Destination
2	Source AND NOT Destination
3	Source
4	NOT Source AND Destination
5	Destination
6	Source XOR Destination
7	Source OR Destination
8	NOT Source AND NOT Destination
9	Source XOR NOT Destination
A	NOT Destination
B	Source OR NOT Destination
C	NOT Source
D	NOT Source OR Destination
E	NOT Source OR NOT Destination
F	Ones
10	Maximum
11	Minimum
12	Add with Saturate
13	Subtract (Destination - Source) with Saturate
14	Subtract (Source - Destination) with Saturate
15	Average

Figure 3-207. Background and Foreground Mixes and Colors

Foreground and Background Mix Registers: The Foreground Mix and Background Mix registers allow a mix (as detailed in the table) to be specified. These registers are discussed in the previous example, “Using the Coprocessor to Perform a PEL Blit (PxBlt)” on page 3-240.

For this example, the Foreground Mix register must be loaded with Source. The Background Mix register is not used in this example.

For the example with the Foreground Mix register:

```
WRITE 03h to copr_regs offset 48h
```

Foreground and Background Color Registers: The Foreground Color register must be set to the color required for the line.

For this example with the Foreground Color register:

```
WRITE 05h to copr_regs offset 58h
```

Bresenham Line Draw: The algorithm used to perform the line draw function on the XGA is the Bresenham Line Draw algorithm. This operates with all parameters normalized to the first octant (octant 0).

The first task is to calculate deltaX and deltaY (see the following figure).

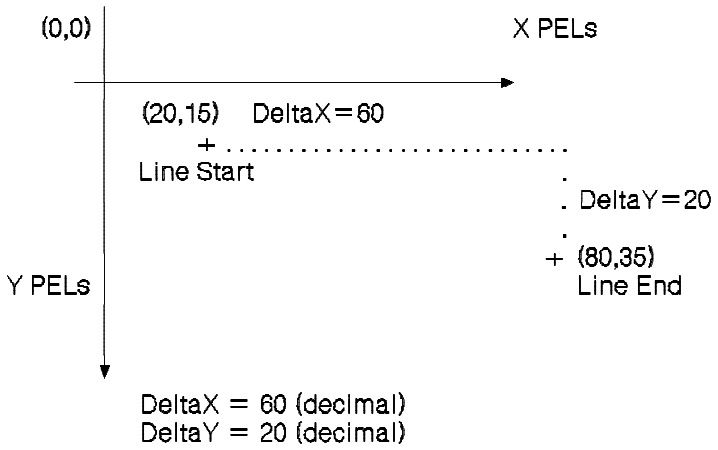


Figure 3-208. Line Draw Example in Octant 0

A line in the first octant has deltaX greater than deltaY, with both deltaX and deltaY positive, and deltaX greater than deltaY. If a line is to be drawn in another octant, the octant information is specified in the octant bits of the PEL Operation register. The line is drawn as if it were in the first octant.

To normalize a line to the first octant, follow these rules:

- If deltaX is – ve , set DX in octant bits of the PEL Operation register and make deltaX + ve.
- If deltaY is – ve , set DY in octant bits of the PEL Operation register and make deltaY + ve.
- If deltaY ≥ deltaX , set DZ in octant bits of the PEL Operation register and exchange deltaX and deltaY.

The terms deltaX and deltaY are the lengths of the line after it has been normalized to octant 0. The algorithm requires several parameters to be calculated. These are:

Bresenham Error Term Register: Bresenham Error Term
E = (2 × deltaY) − deltaX

For this example:

```
WRITE -20 decimal (FFECh) copr_regs offset 20h
```

Bresenham Constant K1 Register: Bresenham Constant
K1 = 2 × deltaY

For this example:

```
WRITE +40 decimal (0028h) copr_regs offset 24h
```

Bresenham Constant K2 Register: Bresenham Constant
K2 = 2 × (deltaY − deltaX)

For this example:

```
WRITE -80 decimal (FFB0h) copr_regs offset 28h
```

Operation Dimension Registers: The Operation Dimension 1 register should be loaded with deltaX after normalization. Because a value of 0 results in a line length of 1 PEL, deltaX (calculated in Figure 3-208 on page 3-250) equals the number to be drawn minus 1.

For this example:

```
WRITE +60 decimal (003Ch) to copr_regs offset 60h
```

The Operation Dimension 2 register is not used for line draw.

PEL Map Source and Destination

Source Map X and Y Registers: The source map is initialized as described in “XGA Coprocessor PEL Interface Registers” on page 3-236. Two registers exist that contain the X and Y offset positions within the source map of the start of the source data for a PxBlt. These registers are used for drawing a line using a source map. In this example, these registers are unused.

Destination Map X and Y Registers: The destination map is initialized as described in “XGA Coprocessor PEL Interface Registers” on page 3-236. Two registers exist that contain the X and Y offset positions within the destination map of the start of the line.

In this example with destination map X and Y positions:

```
WRITE 0014h to copr_regs
offset 78h

WRITE 000Fh to copr_regs
offset 7Ah
```

Pattern Map X and Y Registers: The pattern map is initialized as described in “XGA Coprocessor PEL Interface Registers” on page 3-236. Two registers exist that contain the X and Y offset positions within the pattern map of the start of the pattern data for a line. These registers are used when drawing a line using a pattern map. In this example, these registers are unused.

Mask Map Origin X and Y Offset Registers: The mask map is initialized as described in “XGA Coprocessor PEL Interface Registers” on page 3-236. Two registers exist that contain the X and Y offset positions of the start of the mask map relative to the top left corner of the destination map. These registers are used when drawing a line using a mask map. In this example, these registers are unused.

PEL Operations Register: This is a 32-bit register that defines the operation that the coprocessor performs.

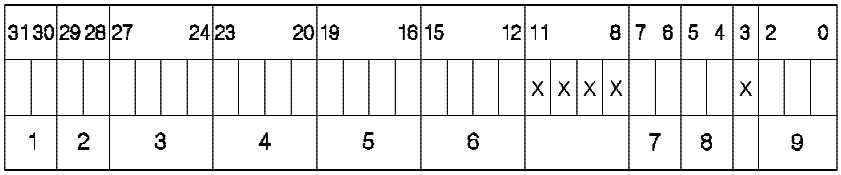


Figure 3-209. Bit Layout PEL Operations Register

The bits 0– 31 are shown on the top of Figure 3-209; Fields 1 through 9 are shown on the bottom. The definition of these fields is:

1. Background Source
2. Foreground Source
3. Step Function
4. Source PEL Map
5. Destination PEL Map
6. Pattern PEL Map
7. Mask PEL Map
8. Drawing Mode
9. Direction Octant.

These fields, described in sequence, are required to assemble the contents of the PEL Operations register:

Background Source: These bits determine the origin of the background source PELs when an operation is performed.

The encoding for these bits is as follows:

Background Color	Binary 00 (for example, for a fixed pattern line draw using a fixed register value)
Source PEL Map	Binary 10 (for example, for a variable color data pattern held in video memory to video memory draw).

In this example, the contents of this field are ignored because the line is solid and there are no background PELs:

```
Background Source = Binary 00
```

Foreground Source: These bits determine the origin of the foreground source PELs when an operation is performed.

The encoding for these bits is as follows:

Foreground Color	Binary 00 (for example, for a fixed pattern line draw using a fixed register value).
Source PEL Map	Binary 10 (for example, for a variable color data pattern held in video memory to video memory draw).

For this example:

```
Foreground Source = Binary 00  (Solid Foreground Color)
```

Step Function: These bits define the type of operation that the coprocessor is required to do.

Draw and Step Read	Binary 0010
Line Draw Read	Binary 0011
Draw and Step Write	Binary 0100
Line Draw Write	Binary 0101
PxBlt	Binary 1000
Inverting PxBlt	Binary 1001
Area Fill PxBlt	Binary 1010

For this example:

```
Step Function = binary 0101
```

Source PEL Map: These bits define the PEL map used as the source map in the operation. This enables different maps to be setup in advance, and defined for use as this register is loaded.

The encoding for these bits is as follows:

PEL Map A	Binary 0001
PEL Map B	Binary 0010
PEL Map C	Binary 0011

In this example the contents of this field are ignored, but they must not be a reserved value:

```
Source PEL Map = binary 0001
```

Destination PEL Map: These bits define the PEL map used as the destination map in the operation. This enables different maps to be setup in advance and defined for use as this register is loaded.

The encoding for these bits is as follows:

PEL Map A	Binary 0001
PEL Map B	Binary 0010
PEL Map C	Binary 0011

For this example with PEL map A:

```
Destination PEL Map = binary 0001
```

Pattern PEL Map: These bits define the PEL map used as the pattern map in the operation. This enables different maps to be setup in advance and defined for use as this register is loaded.

The encoding for these bits is as follows:

PEL Map A	Binary 0001
PEL Map B	Binary 0010
PEL Map C	Binary 0011
Foreground (fixed)	Binary 1000
Generated from Source	Binary 1001

For this example:

```
Pattern PEL Map = binary 1000
```

Mask PEL Map: These bits define whether mask map is used in the operation.

The encoding for these bits is as follows:

Mask Map Disabled	Binary 00
Mask Map Boundary Enabled	Binary 01
Mask Map Enabled	Binary 10

For this example:

```
Mask PEL Map = Binary 00
```

Drawing Mode: These bits determine the attributes of a line draw.

The encoding for these bits is as follows:

Draw All PELs	Binary 00
Draw First PEL Null	Binary 01
! Draw Last PEL Null	Binary 10
Mask Area Boundary	Binary 11

The first three options can be used when drawing a line. The fourth option is for use when drawing the outline of a shape to be filled using the area fill capability of the hardware.

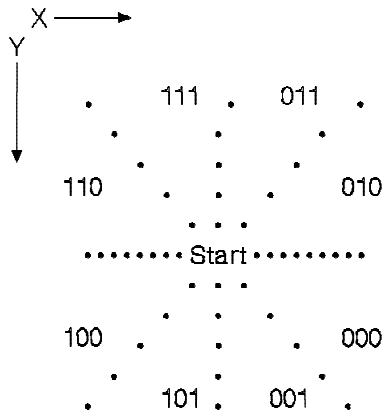
For this example for draw all PELs:

```
Drawing Mode = Binary 00
```

Direction Octant: These bits, when concerned with line draws, determine the direction in which the line is drawn.

The encoding for these bits is as follows:

Bit 2(DX)	1	if negative X direction
	0	if positive X direction
Bit 1(DY)	1	if negative Y direction
	0	if positive Y direction
Bit 0(DZ)	1	if $\|X\| \leq \|Y\|$
	0	if $\|X\| > \|Y\|$, (magnitude)



Note: Numbers are binary.

Figure 3-210. Direction Octant (Example)

For this example:

Direction Octant = binary 000 (X + ve, Y + ve, |X| > |Y|)

Conclusion: Putting all these together for the example line draw operation, the PEL Operations register must be set as:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	1	0	0	0	1	0	0	0	1	1	0	0	0	X	X	X	X	0	0	0	0	X	0	0	0

Figure 3-211. Definition for PEL Operations Register (Example)

For this example:

WRITE 05118000h to copr_regs offset 7Ch

Memory Access Modes

This register has an address of hex 21x9. The Memory Access Modes register is used to control the format of the data supplied by the system processor through a system video memory aperture. For conventional use, this register must be set to match the format of the data as seen by the system processor (Motorola or Intel), and the depth of the video memory bit map.

Through this register, the different formats available can be used to achieve useful and otherwise difficult conversions.

Motorola and Intel Formats

The internal organization of the video memory is Intel format. However, images and bit maps are traditionally stored in Motorola format. It is necessary to understand the format of the application's bit maps in system memory to get the correct results. The different formats are described in “Video Memory Format” on page 3-20.

The internal organization of video memory as Intel format can be hidden by appropriate use of the Memory Access Mode register (“Memory Access Modes”) and the various coprocessor PEL map format registers.

System Processor Access: When using the system processor to read or write data directly to or from video memory through a system video memory aperture, it is necessary to specify the format of the data using the Memory Access Mode register.

XGA Coprocessor Accesses: The format of all bit maps in system memory must be specified through the PEL Map Format register. This parameter is ignored for bit maps in video memory.

Exploitation: Writing data in one format and reading it back in another is a technique that performs many useful and otherwise difficult or expensive bit map conversions.

Other Programming Considerations

Waiting for Hardware Not Busy

| ***Continuous polling:*** Once a XGA coprocessor operation has
| started, the XGA coprocessor operates asynchronously with the
| system processor. Software must wait for the previous operation
| to complete before initiating next operation using the XGA
| coprocessor. This is done by continuous polling of the
| coprocessor busy bit, as described in “Coprocessor Control
| Register (Offset 11)” on page 3-136.

| On XGA-NI subsystems only, the auxiliary coprocessor busy bit,
| described in “Auxiliary Coprocessor Status Register (Offset 09)” on
| page 3-136 is a higher performance alternative for use only on
| XGA-NI subsystems, as sampling does not delay the co-processor.

| Software using the coprocessor busy bit described in
| “Coprocessor Control Register (Offset 11)” on page 3-136 should
| be aware that polling this bit slows down the coprocessor because
| the current operation is paused to process the read of the
| Coprocessor Control register.

| To reduce this effect, software should use a double polling loop
| that checks the coprocessor busy bit, for example, once every 100
| times around the loop.

| An example of this is as follows:

```
|   if (XGA-NI) while (auxiliary_busy);

|   else for (;;) /* original XGA - double loop */
|   {

|       if ¬ (coprocessor_busy) break;
|       for (i=0;i<100;i++);

|   }
```

Operation Complete Interrupt: The coprocessor can be programmed to cause an interrupt to the system processor when an operation is completed.

This interrupt is a shared level. Interrupt response time therefore depends on other interrupt handlers chained on this shared level.

In protect mode operating systems in particular, the overheads and restrictions placed on interrupt handlers may make the performance of this technique prohibitive.

Advantages of using this method are:

- The XGA coprocessor is not slowed while waiting for completion.
- The system processor may be freed up for other tasks.

Disadvantages of using this method are:

- Program complexity.
- Interrupt response time gives a threshold in size of operation that is only exceeded by large PxBlt operations. The more complex the operating system, the higher the interrupt response time, and the larger the operation must be to benefit from using interrupts to notify the application of operation complete.

Overlapping PxBlts

PEL Block Transfer (PxBlt): The coprocessor PxBlt function is used to transfer a rectangular block of PELs from the source to the destination subject to a number of modifiers. It is important to predetermine whether the source and destination rectangles overlap. If the rectangles do not overlap, the order of processing PELs is immaterial. If the rectangles do overlap, program the PxBlt direction using the Direction Octant to ensure the expected result.

Inverting PxBlt

The inverting PxBlt is intended to convert images from the traditional application format of Y increasing upwards to the traditional display hardware format of Y increasing downwards. As such a PxBlt operates from both ends towards the middle, an Inverting PxBlt involving overlapping source and target rectangles inevitably overwrites PELs. Therefore, inverting PxBlts on overlapping rectangles should be avoided, unless for special effects.

Area Fill

Area fill is described in detail in “Area Fill” on page 3-120. This section describes its use from a programming perspective.

All area fill operations rely on a 1bpp area fill work plane, the same pel dimension as the destination bitmap. For a 1024x768 display, this amounts to 96K bytes, which must be allocated in either system memory or off-screen video memory. Higher performance is achieved by locating the area fill plane in off-screen video memory if available.

The area fill operation performed is commonly called “alternate fill.” After the outlines have been drawn into the area fill plane, the area fill bit blit operations works left to right across the fill plane, alternatively filling and not filling as it crosses drawn lines. The significance of this is that a doughnut shape will be correctly filled, leaving the hole in the doughnut unfilled.

Other common area fill algorithms, such as “flood fill” and “winding fill” are not available as XGA coprocessor functions, and must be performed in software.

Exclusive fill: The result of the alternate fill algorithm described above is an “exclusive fill.” Exclusive fill is so called because the filled area includes the top and left boundaries of the defined area, but ***excludes*** the bottom and right boundaries. This algorithm allows abutting filled areas to meet without overlapping.

Software that expects or requires inclusive area fill, in which the filled area includes all boundaries, must perform an additional step to merge the outline lines with the already filled area fill mask at the penultimate stage. Either an entire additional outline plane or the original line vectors must be retained by software for this purpose.

Restrictions

The following restrictions should be noted:

Destination Bit Map Width Restriction Incorrect results can occur if the XGA coprocessor is used to write over the edge of a destination bit map where the edge of the bit map is not 4-byte aligned. To avoid this, use one of the following methods:

- Ensure that all destination bit maps have a base address that is on a 4-byte boundary and are an exact multiple of 4 bytes wide.

  The visible display bit map naturally complies with this restriction.

- Where bit maps are not aligned, software clip all PxBlts in advance so that the destination bit map boundary is not crossed during the PxBlt.

Line Length Restriction The XGA coprocessor Destination X Address and Destination Y Address registers accept coordinates in the range (– 2048 to 6143). This gives a guardband effect, where it is possible to write coordinates anywhere in this range, and the operation is hardware scissored to the edge of the destination bit map. The limit on bit map size for coprocessor operations is 0 to 4095.

Because the Operation Dimension 1 register only accepts values in the range (0 to 4095), it is not possible to draw a line in a single operation across the entire guardband coordinate space.

A two-stage line draw can be performed easily, since the line parameters (for example, ET, K1, K2, Destination X and Y, Pattern X) are already set up in the hardware at the end of the previous line segment. It is only necessary to update the new line length in the Operation Dimension 1 register to draw the remainder of the line.

Common Problems

This section is a description of the most common problems experienced when programming the XGA coprocessor.

Source map depth must match destination The depth of the source and destination bitmaps must match. For example, if the source bitmap is 4 bpp, it cannot be used in any coprocessor operation with a destination bitmap of 8 bpp. If the source destination is 1bpp, it can be used as a pattern map, but not as a source map with a destination bitmap other than 1 bpp.

The result of failure to observe this rule can be obscure. The operation that fails is not the one that caused the problem, but is a subsequent XGA

coprocessor function of the same type some time in the future. As a result, this problem can be hard to detect.

Save Restore Sequence A number of problems regularly occur with save and restore. These are as follows:

- Failure to check operation suspended
- Failure to “terminate” before starting new operation
- Failure to wait for operation complete after terminating

Failure to wait for previous operation complete The XGA coprocessor operates asynchronously to the system processor, and may not finish an operation for some time. Software must be aware that writing to the XGA Pel Operations register only starts an operation, and software must then wait for the operation to complete before either:

- Writing to any XGA coprocessor registers
- Using the resulting bitmap from a XGA coprocessor operation
- Re-using, freeing or unlocking memory used to contain bitmaps

Operations outside bitmap guardband While there is a 2K guardband around bitmaps, this effectively means that the XGA coprocessor deals on 14 bit coordinates. Operations such as lines outside this coordinate range will still wrap, producing unexpected lines.

Bitmaps must be dword aligned The start address (as loaded into the XGA coprocessor pel map base pointer) of all bitmaps accessed by the XGA coprocessor must be located on a double word (32 bit dword aligned) boundary.

Subrectangles within a bitmap can be used to access data on other boundaries, as long as the bitmap start address is dword aligned.

Uninitialised X,Y pointers Source, Destination and Pattern map X & Y addresses and Mask map origin X & Y offset are uninitialised after a reset, or on initial power-up. The result of operations involving any of these addresses prior to initialisation is undefined. They must be initialised to a valid value before any operation involving the bitmap to which they refer.

Performance tips

This section describes how to achieve the optimum performance on the XGA subsystem.

Polling for completion: As described in "Waiting for Hardware Not Busy" on page 3-259, software should not continuously poll the coprocessor busy bit.

On XGA-NI subsystems use the Auxiliary busy bit. On XGA subsystems, use the double loop as described.

An improvement on this might be different delays according to the size of operation initiated, or a geometric delay, where the delay would start small and increase with repetition until the operation completes.

Text performance: The following have been found to improve text performance:

- Balance hardware and software. Software can leave all the work to the coprocessor, or alternatively software can do most of the work. The optimum solution varies by application, but some experimentation should be done to ensure that the coprocessor completes the previous character in a string at about the same time as the software has prepared the next character, so that hardware and software is approximately balanced.
- Software clip characters. Calculate in advance the minimum bounding rectangle of the displayed characters, and leave hardware clipping turned off. Do not use the coprocessor to PxBlt portions of characters that will be clipped out.
- Draw the entire background opaque rectangle as a single rectangle, not on a character by character basis.

Line drawing

- Do not use the line drawing operation for horizontal and vertical lines.

  It is faster to use the simple PxBlt operation for horizontal and vertical lines. Be aware that the Destination X and Y addresses will then need to be updated to the start of the next line segment.

- Destination X and Y addresses are automatically updated to the end of the drawn line. There is no need to set them prior to subsequent line segments.
- Coarse clip lines in software. Turn of XGA coprocessor hardware clip when clipping is not required. If both ends of the line are within the clip rectangle, turn clipping off. If both ends of the line are outside the same boundary, do not draw the line. Only use coprocessor hardware clip on lines that crosses the clip rectangle.

Software clip: Where possible, clip in software. It takes as long to hardware clip out lines and rectangles as it does to draw them. It is better to pre-calculate that the rectangle or line will not be drawn, than to use the hardware clip.

On rectangles (for instance characters or PxBlts), pre-calculate the subrectangle that will be unclipped.

On lines, only clip those lines that cross the clip boundary.

Video memory faster than system memory: Coprocessor operations on video memory is significantly faster than similar operations on system memory. Off-screen video memory is the optimum place to locate the font cache, patterns, brushes, the area fill plane, and the most frequently used bitmaps.

Sprite Handling

Technical details of the sprite are described in “Sprite” on page 3-26. This section concentrates only on programming aspects of sprite use.

Sprite Loading: The sprite is loaded as a 64 x 64 x 2 bits per PEL (bpp) Intel format image definition. Because the sprite definition in the application is invariably held in two separate 1-bpp Motorola format bit maps, it is necessary to merge and PEL swap the sprite definition into the 2-bpp Intel format before loading the sprite.

Sprite Positioning: The position of the sprite is then controlled by two separate controls:

Sprite Start Registers

The sprite is positioned on the display surface by specifying the position of the top left corner of the sprite definition relative to the top left corner of the visible bit

map, using the Sprite Horizontal Start and Sprite Vertical Start registers.

Sprite Preset Registers

The sprite start registers only accept positive values, and cannot be used to move the sprite partially off the display surface at the left and top edges. The Sprite Horizontal Preset and Sprite Vertical Preset registers are used to offset the start of the displayed sprite definition relative to the loaded definition.

For example, to display a 64 x 64 PEL sprite with the leftmost 32 PELs outside the left edge of the display surface, set the Sprite Horizontal Start register to 0, and the Sprite Horizontal Preset register to 32. The start position is now preset to the center of the loaded definition, giving the required effect.

The sprite preset can also be used to display sprites smaller than 64 x 64 PELs.

| ***Sprite update*** Update of the sprite contents can be synchronised with its display using the Sprite Display Complete Status bit described in “Interrupt Status Register (Address 21x5)” on page 3-41, to avoid partial sprites becoming visible during update. This bit can be polled to avoid using interrupts when required.

| **Palette formats**

| Technical details of the palette loading are described in “Palette” on page 3-29. This section concentrates only on programming aspects of palette use.

| Two palette formats are offered. While the R,G,B format of 3 bytes per entry is more space efficient, the alternative of R,B,G,X allows individual palette entries to be stored in a single Dword, and loaded into the appropriate palette location in a single Dword OUT to the XGA Data I/O Port 21xCh.

| **XGA Subsystem Save, Restore, Suspend & Resume**

| In a task or screen switching environment, it may be necessary to suspend the current operation, save the state of the display subsystem(s), and restore the subsystem to a known state, such as VGA text mode, so that another process or task may use the display subsystem. Subsequently the suspended process may be reactivated, at which time software must restore the entire display

| subsystem state, and resume the suspended operation at the point
| where it was suspended.

| This may be necessary even in DOS programs, if such programs
| are to be candidates to run in the Multiple Virtual DOS Machine
| environments of OS/2 and Windows.

| The complete XGA subsystem save and restore consists of five
| seperate parts, as follows:

- Video buffer contents
- Coprocessor state
- Sprite contents
- Palette contents
- Other I/O registers

| Each of these components is discussed below in more detail.

| ***Video Buffer Contents:*** Software may choose not to save and
| restore the entire video buffer contents, if there is a more memory
| efficient means of recreating the video buffer contents when
| requested.

| Software that chooses to save and or restore the entire video
| buffer contents may do so using either apertures, or the XGA
| coprocessor to PxBlt the video buffer to a screen save area, or
| logical video buffer.

| ***Coprocessor state:*** Capabilities:

- The coprocessor can be suspended at any time during an operation, and the state of the coprocessor can be saved.
- A new coprocessor operation can then be started.
- A suspended operation can be restored and resumed at any time.

| *Coprocessor Suspend and Save:* To suspend and save the
| coprocessor, software should perform the following precise
| sequence of operations:

1. Check for coprocessor busy, by testing the BSY bit the coprocessor control register, as described in “Coprocessor Control Register (Offset 11)” on page 3-136. If found to be busy, the following steps are necessary to suspend the coprocessor in the middle of the current operation.

   a. Suspend coprocessor, by setting the “SO” bit in the coprocessor control register, as described in “Coprocessor Control Register (Offset 11)” on page 3-136.
   b. Wait until Operation has been suspended, by polling the “OS” bit in the coprocessor control register, as described in “Coprocessor Control Register (Offset 11)” on page 3-136.
2. Select save mode, by setting “SR”='1'b in the coprocessor control register, as described in “Coprocessor Control Register (Offset 11)” on page 3-136.
3. Read lengths of data to be saved. These are available seperately as the lengths of save data A & B in Dwords (32 bit double words), as described in “State Length Registers (Offset C and D)” on page 3-139.
4. Read (using “REP I/O”) and save the exact number of Dwords for data port A (“Coprocessor Save/Restore Data Registers (Index 0C and 0D)” on page 3-49), as determined above.
5. Read (using “REP I/O”) and save the exact number of Dwords for data port B (“Coprocessor Save/Restore Data Registers (Index 0C and 0D)” on page 3-49), as determined above.
6. Terminate the current operation, by setting the “TO” bit in the coprocessor control register, as described in “Coprocessor Control Register (Offset 11)” on page 3-136.
7. Wait for processor not busy, either by polling the “BSY” bit in the coprocessor control register (“Coprocessor Control Register (Offset 11)” on page 3-136) or (on XGA-NI subsystems only) by polling the “ABSY” bit in the auxiliary coprocessor control register (“Auxiliary Coprocessor Status Register (Offset 09)” on page 3-136).

The coprocessor is now in an uninitialised reset state, ready to be used as required. Software must fully initialise the coprocessor before use, and should make no assumptions about its previous state.

Coprocessor Restore and Resume: To restore and resume any previously suspended operation and state, software should perform the following precise sequence of operations:

The coprocessor is assumed to be “Not busy,” as it would be following a complete suspend & save sequence, as described in “Coprocessor Suspend and Save” on page 3-267.

1. Select restore mode, by setting “SR”='0'b in the coprocessor control register, as described in “Coprocessor Control Register (Offset 11)” on page 3-136.

2. Write (using “REP I/O”) the exact number of Dwords previously saved to data port A (“Coprocessor Save/Restore Data Registers (Index 0C and 0D)” on page 3-49).
3. Write (using “REP I/O”) the exact number of Dwords previously saved to data port B (“Coprocessor Save/Restore Data Registers (Index 0C and 0D)” on page 3-49).
4. Resume any suspended operation, by resetting the “SO” bit in the coprocessor control register, as described in “Coprocessor Control Register (Offset 11)” on page 3-136.

If an operation had been suspended prior to suspend, it has now been restarted from the point of suspension. Alternatively, if the processor was previously not busy, it is again in the same state.

Alternative XGA Coprocessor Register Set

An alternative XGA Coprocessor Register Location may be present on some systems, as can be determined by examining the DMQS primary data area as described in “DMQS BIOS Interface” on page 3-194.

If present, improved performance can be gained by accessing the XGA coprocessor registers at this alternative location. This physical location (if present) will be located in protect mode system address space. Only protect mode device drives are able to access the alternative XGA coprocessor registers at this location.

System Register Usage

When programming the XGA subsystem, it is often necessary to maintain addressability to:

- XGA coprocessor memory mapped address space
- XGA state data segment (application dependent) containing the I/O base address, in other words the location of the XGA registers in I/O space
- The normal function dependent application data, such as parameter blocks
- Global application dependent data.

Many of the XGA registers are 32-bit registers.

| To program the XGA subsystem efficiently, it is helpful to use the
| full i386 register set, specifically the FS and GS segment registers
| and the 32-bit extended data registers.

| Use of the extra segment registers allows concurrent
| addressability to all the separate data areas to be maintained
| without frequent segment register loading (a particularly expensive
| operation in protect modes).

Direct Color Mode
This section deals with matters unique to the direct color mode of the XGA subsystem.

Palette Loading: It is necessary to load the palette with a fixed set of values. These are described in “Direct Color Mode” on page 3-31.

| The original XGA subsystem does not support XGA coprocessor
| operations on 16 bit direct color bitmaps. The XGA-NI subsystem
| provides full 16 bit direct color support. The following section is
| applicable only to software written to support the original XGA
| subsystem.

| ***Coprocessor Support on XGA subsystems only:*** The XGA coprocessor does not support the 16 bits-per-PEL (bpp) mode. This mode is a display mode only, and must be programmed using the system processor to access the video memory display buffer directly using one of the system video memory apertures (see “PS/2 System Video Memory Apertures” on page 3-232 and “Memory Access Modes” on page 3-258).

The coprocessor is not disabled in this mode. However, the PEL map formats available for coprocessor operations are restricted to 1, 2, 4, or 8 bpp. The coprocessor can be used in this mode if the application manages the differences in bits per PEL. Some ingenuity is required to achieve useful results using the coprocessor in this way.

Bit Block Transfer Operations
By using the PxBlt operations on an 8-bpp bit map, doubling the dimension width of the bit maps involved, and avoiding arithmetic mixes, bit block transfer operations are possible. Use of the 1-bpp pattern and mask maps are possible if carefully considered and calculated.

Text Operations
Text operations using the coprocessor PxBlt function rely on 1-bpp patterns. By doubling the width of the individual character bit map patterns (interspersing the active bits with zero bits) and writing the high and low order bytes of the required color index separately, text operations are possible.

Use of DMA Busmastership

Physical Addressability to System Memory

The XGA subsystem coprocessor can operate as a Micro Channel busmaster. As a busmaster, the coprocessor is capable of bit map operations on bit maps up to 4KB by 4KB PELs anywhere in system address space, including video memory. A PxBlt operation can be defined as a function of four separate bit maps:

D' = f(S,D,P,M)

That is, the modified destination PEL (*D'*) is a function of the source (*S*), the current destination PEL (*D*), the pattern (*P*), and the mask (*M*). These bit maps can be anywhere in memory. The XGA coprocessor handles all bit maps alike. No special handling of a bit map in video memory is required.

This flexibility is very powerful, but requires support from the operating system to fully realize the benefits.

Busmastership is on i386 physical address space, while applications run on the system processor in virtual or linear address space. The system processor automatically converts such addresses to physical addresses internally through the page tables or segment descriptor tables. An adapter, such as the XGA coprocessor, has no physical access to the segment descriptors or the page tables. To use busmastership, the application (or its device drivers) must provide the XGA coprocessor with the physical address of all the bit maps on which it requires the XGA coprocessor to operate. Methods for providing the XGA coprocessor with physical addressability to all such resources, and the tasks necessary, vary according to the operating system and the mode of the system processor.

16 Meg Limitation on busmastership addressability

Where the XGA subsystem is installed in a 32 bit slot in a 32 bit system, DMA busmastership is possible over the full range of 32 bit (4 Gigabyte) address space.

However there are a number of situations where XGA subsystems will be installed on a 16 bit bus (16 Megabyte address space) where system memory is on a 32 bit bus (4 Gigabyte). If there is more than 16 Megabytes of system memory in such a system, DMA Busmastership will not be possible on memory located above 16 Megabytes.

| This situation will occur as follows:

| • Where the XGA subsystem is located in a 16 bit slot in a 32 bit system.

| • Where the XGA subsystem is located on an ISA AT bus in a i386DX or i486 based system with an internal local 32 bit memory bus.

| Software must ensure that all memory used in operations involving the XGA coprocessor is located in the lower 16 Megabytes of system address space. This principally applies to protect mode applications and device drivers, and may involve requesting memory in this area from operation system memory management services.

| Operation systems that support busmasters must provide the facility for software to specify that memory be allocated in this area of memory.

Real-Mode DOS Environments

The real-mode DOS environment is the simplest and easiest in terms of memory management. The application is limited to 640KB of real-mode DOS memory. Virtual-to-linear memory address conversion is done by means of a simple *shift left 4 and add* operation, and the nature of the real-mode DOS environment is that linear addresses are identical to physical addresses.

In the multiple virtual DOS machine (MVDM) environment, however, linear addresses are no longer identical to physical addresses, and a DOS application or device driver may not necessarily work correctly in an MVDM environment.

In most cases, the virtualization display driver of the MVDM hypervisor will cope with this, but applications must be tested in individual MVDM environments before full, real-mode DOS compatibility can be claimed.

Extended Memory: A DOS application can allocate large areas of extended memory as working bit maps for the application. It is unnecessary to have system processor addressability to such bit maps. The XGA coprocessor can do all the necessary accesses, and extended memory is ideal for this purpose.

The techniques required to allocate and use extended memory in a DOS application are not covered here.

LIM EMS Managers: The most common memory management technique that gives extra memory in the DOS environment is the Lotus-Intel-Microsoft Expanded Memory Services Manager. These memory managers implement the LIM 4.0 specification for a software interrupt driven memory management interface software interrupt 67h. On 80386 and above processors, memory is physically allocated as extended memory, and the LIM EMS manager maps this into expanded memory using the 80386 page tables.

The drawback to this technique is that a simple shift left 4 and add operation yields the linear, but not the physical address of the LIM frame. To determine the physical address, it is necessary to call the Operating System DMA services interface of the LIM EMS driver to convert linear addresses to physical addresses. This interface, based on Software Interrupt 4Bh, is described in the *IBM Personal System/2 and Personal Computer BIOS Interface Technical Reference*.

This interface is of recent origin, and early LIM drivers may not have implemented it. There are two choices for the application:

- Do not locate resources in LIM memory on which the XGA coprocessor is requested to operate.
- Specify a dependency in the application documentation on LIM EMS drivers that have implemented this interface.

32-Bit DOS Extended Environments: In this mode, exploitation of the power of the XGA coprocessor is easiest. The application can allocate large memory bit maps without accounting for the behavior of a memory manager that might change the location of the memory. Calculation of physical addresses is easily accomplished without the system overheads of full-blown protect mode operating systems. Access to the XGA system video memory aperture and coprocessor register address space can be accomplished easily.

Multiple Virtual DOS Machine Environments: In this mode, multiple DOS applications can run concurrently (even windowed on the same screen) in virtual DOS machines (VDMs) and each application appears internally to be running in the bottom 1MB of physical address space.

Full compatibility with real-mode DOS for a busmaster, such as the XGA coprocessor, is only provided if each DOS application using the XGA subsystem in Extended Graphics mode is locked in the bottom 1MB of physical address space. Because this is impractical, the virtualization display driver (VDD) of the MVDM hypervisor must include specific support functions to support VDMs running Extended Graphics mode applications.

One technique is described here, although there may be others that are equally effective.

When a VDM that is running an XGA Extended Graphics mode DOS application switches to the foreground, the VDD locks the entire 640KB of linear address space in the VDM without making the memory contiguous. The VDD then uses the page directory entry (PDE) of the foreground VDM to provide physical addressability to the noncontiguous linear address space. The virtual address capability of the XGA coprocessor can then be used by giving the XGA coprocessor direct DMA access to the page tables of the VDM. Because the entire 640KB DOS region is locked (except for LIM which will be discussed below), a DOS application will not normally supply linear addresses outside that range.

The Extended Graphics mode DOS application must not modify the XGA coprocessor Page Directory Base Address after it is set by the VDD when switching the VDM to the foreground. Application updates to this field can be prevented by placing the XGA coprocessor into user mode.

The DOS application may locate a resource, such as a font definition, in LIM memory and give the XGA coprocessor the linear address of the LIM frame, rather than the underlying address. This is normally handled in real-mode DOS by calling the *Operating System DMA Services* interface of the LIM EMS driver to convert linear addresses to physical. In the MVDM environment, the XGA coprocessor is in VM mode, and the linear address of the LIM frame is required, rather than the physical address. The VDD can monitor the LIM software interrupt (Int 67h), and ensure that any LIM *logical 16K pages* currently mapped into the LIM frames or windows of the VDM are locked. The page tables of the VDM will then naturally reflect the correct physical addresses for the LIM pages at the linear address of the LIM frame. Calls to the *Operating System DMA Services* interface must also be filtered out.

Protect Mode 16-Bit Segmented Environment: An application written for this environment has a range of limitations imposed by the operating system.

64KB Segment Limit: No memory object in this environment can be larger than 64KB, unless allocated by a kernel device driver on initialization.

The application cannot assume that two adjacent segments are located adjacently in physical address space.

Segment Motion: Segments may be moved in physical system memory at any time. Segments may even be swapped out to disk when memory is over committed.

All segments must be locked before the physical address is established.

Consideration must be given to the overall impact on system performance of locking large areas of memory. Locking increases the minimum physical memory configuration required to run the application.

System Overheads: Applications generally run at a low privilege level and video device drivers must be easily and frequently accessible by the application without large system overheads.

Applications using the XGA coprocessor need to make use of the memory management services of the operating system. These services (used for locking segments and determining the physical address of segments) are typically restricted to device drivers operating at high privilege levels.

The system overhead in reaching these services in such operating systems can be so high that it makes the writing of high performance applications difficult.

Access to XGA Registers and System Memory Apertures: Ingenuity is required to provide addressability to the I/O and memory space of the XGA subsystem. A technique for this is described in “PS/2 System Video Memory Apertures” on page 3-232.

Following is a suggested design for an application in this environment. This technique minimizes kernel or system overheads.

Use a kernel or ring 0 .SYS device driver to permanently allocate a range of physical memory (typically 128K) as kernel work space (KWS). The device driver can then generate a GDT selector to the KWS that is valid in user mode at ring 3. Both the virtual and physical addresses of the KWS are passed back to the application in user mode. The kernel device driver also provides user mode addressability to the register address space of the XGA coprocessor.

The device driver or application can then operate totally in user mode, passing resources (for example, bit maps or patterns) by system processor block moves into the KWS. The application can then use the busmastership capability of the XGA coprocessor to access the resources in the KWS without suffering the system overheads of switching into kernel mode again. Bit maps being transferred to or from the adapter can be double buffered through the KWS to overlap system processor and XGA coprocessor operations on large operations.

Paged Virtual Memory (virtual memory) Environments: This environment shares many constraints with the 16-bit segmented environment. The main difference is that the unit of granularity of memory objects has dropped from 64KB to 4KB; the virtual memory support in the XGA coprocessor is intended to support this environment.

4KB Discontiguous Pages: In this environment, memory is allocated to applications in 4KB pages. The system memory manager controls paging and can swap pages in and out of physical memory transparently to the application. The application can make no assumptions about the relationship between adjacent pages.

There are memory management calls available to the kernel or ring 0 device driver that let the device driver build a table containing the physical addresses of all the component pages of a large bit map. As with 16-bit segmented environments, described in “Protect Mode 16-Bit Segmented Environment” on page 3-276, the overhead of the transition to kernel mode makes such calls expensive. It is, however, possible to build such a table and to operate the XGA coprocessor in virtual memory mode. The

overall impact on system performance and minimum physical memory configurations should be considered. A bit map in this case could theoretically be 4Kx4Kx8 bits per PEL, which is a total of 16MB of locked physical memory.

It is possible to use the XGA coprocessor to interrupt (to indicate a page fault). However, this interrupt is a normal shared-adapter interrupt rather than a i386 page fault interrupt, and is handled at a lower priority. Most operating systems do not allow device drivers to call the memory management services to request the faulting pages.

Page Table Coherency: It may appear that the XGA coprocessor can operate off the system page tables because the XGA coprocessor uses i386-like page tables. Unfortunately, a typical virtual memory operating system uses one set of page tables per task. In a multitasking environment, only the currently executing task page tables remain coherent, while background task page tables become outdated or incoherent.

This implies that the XGA coprocessor can be operating on a set of page tables belonging to a background task. It cannot be assumed that the page table remains coherent, unless the component pages have been locked by a call to the system memory management interface by a kernel device driver.

System Overheads: The overheads associated in switching from the application privilege level to the kernel level have been described in *System Overheads* in “Protect Mode 16-Bit Segmented Environment” on page 3-276.

Access to XGA Registers and System Memory Apertures: It is necessary to provide addressability to these XGA subsystem I/O spaces. Call the operating system memory management services to map these ranges of physical system memory into the application task address space.

Suggested Design Model: The optimum design model is one that minimizes kernel overhead. A model similar to that suggested in “Protect Mode 16-Bit Segmented Environment” on page 3-276 is appropriate for this environment.

Video Memory Addressability in Virtual Memory Mode: "Video Memory Address Range" on page 3-235 has a description of how the XGA coprocessor differentiates video memory from system memory. When operating the XGA subsystem in VM mode, this differentiation is done after page table translation on physical address space. All addresses passed to the XGA coprocessor by the application or device driver are in linear address space, before page table translation. When the application or device driver is building VM addressability to system memory bit maps for the XGA subsystem, it must also map local video memory into the page table structure at the correct location in physical address space to allow the XGA coprocessor to differentiate video memory from system memory.

System Memory Access Limitation: The XGA subsystem can be plugged into any 16- or 32-bit slot in any i386SX, i386DX, or i486 system. In a 16-bit slot, the address range is limited because there are only 24 address lines on 16-bit slots. The range of physical addressability to system memory using busmastership is limited to 24-bit physical address space (or 16MB) when the subsystem occupies a 16-bit slot.

Systems based on the i386SX are 16-bit throughout. The limit of addressability of the system processor is 16MB.

There are constraints when:

- A 32-bit system is based on the i386DX or i486
- There is more than 16MB of physical memory installed
- The XGA subsystem is plugged into a 16-bit slot.

  The XGA coprocessor cannot access memory located above the 16MB line in physical address space. To determine if the XGA subsystem is in a 16-bit slot, examine the Auto-Configuration register, as described in "Auto-Configuration Register (Index 04)" on page 3-48. The application must ensure (with operating system assistance if necessary) that all memory bit maps on which the XGA processor is asked to operate are located below the 16MB line in physical address space.

  The alternative is for the application to specify that the XGA subsystem is always plugged into 32-bit slots on 32-bit systems.