VESA Standard

Super VGA

Video Electronics Standards Association

1330 South Bascom Avenue, Suite D
San Jose, CA 95128-4502

Phone: (408) 971-7525
Fax: (408) 286-8988

SUPER VGA BIOS EXTENSION

Standard # VS891001

October 1, 1989

PROPOSAL

To standardize a common software interface to Super VGA video adapters in order to provide simplified software application access to advanced VGA products.

SUMMARY

The standard provides a set of functions which an application program can use to a) obtain information about the capabilities and characteristics of a

specific Super VGA implementation and b) to control the operation of such hardware in terms of video mode initialization and video memory access. The functions are provided as an extension to the VGA BIOS video services, accessed through interrupt 10.

Contents

1. Introduction

This document contains a specification for a standardized interface to extended VGA video modes and functions. The specification consists of mechanisms for supporting standard extended video modes and functions that have been approved by the main VESA committee and non-standard video modes that an individual VGA supplier may choose to add, in a uniform manner that application software can utilize without having to understand the intricate details of the particular VGA hardware.

The primary topics of this specification are definitions of extended VGA video modes and the functions necessary for application software to understand the characteristics of the video mode and manipulate the extended memory associated with the video modes.

Readers of this document should already be familiar with programming VGAs at the hardware level and Intel iAPX real mode assembly language. Readers who are unfamiliar with programming the VGA should first read one of the many VGA programming tutorials before attempting to understand these extensions to the standard VGA.

2. Goals and Objectives

The IBM VGA[1] has become a de-facto standard in the PC graphics world. A multitude of different VGA offerings exist in the marketplace, each one providing BIOS or register compatibility with the IBM VGA. More and more of these VGA compatible products implements various supersets of the VGA standard. These extensions range from higher resolutions and more colors to improved performance and even some graphics processing capabilities. Intense competition has dramatically improved the price/performance ratio, to the benefit of the end user.

However, several serious problems face a software developer who intends to take advantage of these "Super VGA"[2] environments. Because there is no standard hardware implementation, the developer is faced with widely disparate Super VGA hardware architectures. Lacking a common software interface, designing applications for these environments is costly and technically difficult. Except for applications supported by OEM-specific display drivers, very few software packages can take advantage of the power and capabilities of Super VGA products.

The purpose of the *VESA VGA BIOS Extension* is to remedy this situation. Being a common software interface to Super VGA graphics products, the primary objective is to enable application and system software to adapt to and exploit the wide range of features available in these VGA extensions.

Specifically, the *VESA BIOS Extension* attempts to address the following two main issues: a) Return information about the video environment to the application and b) Assist the application in initializing and programming the hardware.

2.1 Video environment information

Today, an application has no standard mechanism to determine what Super VGA hardware it is running on. Only by knowing OEM-specific features can an application determine the presence of a particular video board. This often involves reading and testing registers located at I/O addresses unique to each OEM. By not knowing what hardware an application is running on, few, if any, of the extended features of the underlying hardware can be used.

The *VESA BIOS Extension* provides several functions to return information about the video environment. These functions return system level information as well as video mode specific details. ***Function 00h*** returns general system level information, including an OEM identification string. The function also returns a pointer to the supported video modes. ***Function 01h*** may be used by the application to obtain information about each supported video mode. ***Function 03h*** returns the current video mode.

2.2 Programming support

Due to the fact that different Super VGA products have different hardware implementations, application software has great difficulty in adapting to each environment. However, since each is based on the VGA hardware architecture, differences are most common in video mode initialization and memory mapping. The rest of the architecture is usually kept intact, including I/O mapped registers, video buffer location in the CPU address space, DAC location and function, etc.

The *VESA BIOS Extension* provides several functions to interface to the different Super VGA hardware implementations. The most important of these is ***Function 02h***, Set Super VGA video mode. This function isolates the application from the tedious and complicated task of setting up a video mode. ***Function 05h*** provides an interface to the underlying memory mapping hardware. ***Function 04h*** enables an application to save and restore a Super VGA state without knowing anything of the specific implementation.

2.3 Compatibility

A primary design objective of the *VESA BIOS Extension* is to preserve ***maximum compatibility to the standard VGA environment***. In no way should the BIOS extensions compromise compatibility or performance. Another but related concern is to minimize the changes necessary to an existing VGA BIOS. RAM- as well as ROM-based implementations of the BIOS extension should be possible.

2.4 Scope of standard

The purpose of the *VESA BIOS Extension* is to provide support for extended ***VGA*** environments. Thus, the underlying hardware architecture is assumed to be a VGA. Graphics software that drives a Super VGA board, will perform its graphics output in generally the same way it drives a standard VGA, ie. ***writing directly*** to a VGA style frame buffer, manipulating graphics controller registers, directly programming the palette etc. No significant graphics processing will be done in hardware. For this reason, the *VESA BIOS*

Extension does not provide any graphics output functions, such as BitBlt, line or circle drawing, etc.

An important constraint of the functionalities that can be placed into ***the*** *VESA BIOS Extension****,*** is that ***ROM-space is severely limited in certain existing BIOS implementations***.

Outside the scope of this *VESA BIOS Extension* is handling of different monitors and monitor timings. Such items are dealt with in other VESA fora. The purpose of the *VESA BIOS Extension* is to provide a standardized software interface to Super VGA graphics modes, independent of monitor and monitor timing issues.

3. Standard VGA BIOS

A primary design goal with the *VESA BIOS Extension* is to minimize the effects on the standard VGA BIOS. Standard VGA BIOS functions should need to be modified as little as possible. This is important since ROM- as well as RAM-based versions of the extension may be implemented.

However, two standard VGA BIOS functions are affected by the VESA extension. These are ***Function 00h*** (Set video mode) and ***Function 0Fh*** (Read current video state). VESA-aware applications will not set the video mode using VGA BIOS function 00h. Nor will such applications use VGA BIOS function 0Fh. ***VESA BIOS functions 02h*** (Set Super VGA mode) and ***03h*** (Get Super VGA mode) will be used instead.

However, VESA-unaware applications (such as old Pop-Up programs and other TSRs, or the CLS command of MS-DOS), might use VGA BIOS function 0Fh to get the present video mode. Later it may call VGA BIOS function 00h to restore/reinitialize the old video mode.

To make such applications work, VESA recommends that whatever value returned by VGA BIOS function 0Fh (it is up to the OEM to define this number), it can be used to ***reinitialize*** the video mode through VGA BIOS function 00h. Thus, the BIOS should keep track of the last Super VGA mode in effect.

It is recommended, but not mandatory, to support ***output functions*** (such as TTY-output, scroll, set pixel, etc.) in Super VGA modes. If the BIOS extension doesn't support such output functions, bit D2 (Output functions supported) of the ***ModeAttributes*** field (returned by VESA BIOS function 01h) should be cleared.

4. Super VGA mode numbers

Standard VGA mode numbers are 7 bits wide and presently ranges from 00h to 13h. OEMs have defined extended video modes in the range 14h to 7Fh. Values in the range 80h to FFh cannot be used, since ***VGA BIOS function 00h*** (Set video mode) interprets bit 7 as a flag to clear/not clear video memory.

Due to the limitations of 7 bit mode numbers, VESA video mode numbers are 15 bits wide. To initialize a Super VGA mode, its number is passed in the BX register to ***VESA BIOS function 02h*** (Set Super VGA mode).

The format of VESA mode numbers is as follows:

D0-D8=	**Mode number**
	If D8 == 0, this is not a VESA defined mode
	If D8 == 1, this is a VESA defined mode
D9-D14=	**Reserved by VESA for future expansion (= 0)**
D15=	**Reserved (= 0)**

Thus, VESA mode numbers begin at 100h. This mode numbering scheme implements standard VGA mode numbers as well as OEM-defined mode numbers as ***subsets*** of the VESA mode number. That means that regular VGA modes may be initialized through VESA BIOS function 02h (Set Super VGA mode), simply by placing the mode number in BL and clearing the upper byte (BH). OEM-defined video modes may be initialized in the same way.

To date, VESA has defined ***a 7-bit video mode number, 6Ah,*** for the 800x600, 16-color, 4-plane graphics mode. The corresponding 15-bit mode number for this mode is 102h.

The following VESA mode numbers have been defined:

15-bit mode number	**7-bit mode number**	**Resolution**	**Colors**

100h	-	640x400	256
101h	-	640x480	256
102h	6Ah	800x600	16
103h	-	800x600	256
104h	-	1024x768	16
105h	-	1024x768	256
106h	-	1280x1024	16
107h	-	1280x1024	256

5. CPU Video Memory Windows

A standard VGA sub-system provides 256k bytes of memory and a corresponding mechanism to address this memory. Super VGAs and their extended modes require more than the standard 256k bytes of memory but also require that the address space for this memory be restricted to the standard address space for compatibility reasons. CPU video memory windows provide a means of accessing this extended VGA memory within the standard CPU address space.

This chapter describes how several hardware implementations of CPU video memory windows operate, their impact on application software design, and relates them to the software model presented by the VESA VGA BIOS extensions.

The VESA CPU video memory windows functions have been designed to put the performance insensitive, non-standard hardware functions into the BIOS while putting the performance sensitive, standard hardware functions into the application. This provides portability among VGA systems together with the performance that comes from accessing the hardware directly. In particular, the VESA BIOS is responsible for mapping video memory into the CPU address space while the application is responsible for performing the actual memory read and write operations.

This combination software and hardware interface is accomplished by informing the application of the parameters that control the hardware mechanism of mapping the video memory into the CPU address space and then letting the application control the mapping within those parameters.

5.1 Hardware design considerations

5.1.1 Limited to 64k/128k of CPU address space

The first consideration in implementing extended video memory is to give access to the memory to application software.

The standard VGA CPU address space for 16 color graphics modes is typically at segment A000h for 64k. This gives access to the 256k bytes of a standard VGA, i.e. 64k per plane. Access to the extended video memory is accomplished by mapping portions of the video memory into the standard VGA CPU address space.

Every super VGA hardware implementation provides a mechanism for software to specify the offset from the start of video memory which is to be mapped to the start of the CPU address space. Providing both read and write access to the mapped memory provides a necessary level of hardware support for an application to manipulate the extended video memory.

5.1.2 Crossing CPU video memory window boundaries

The organization of most software algorithms which perform video operations consists of a pair of nested loops: an outer loop over rows or scan lines and an inner loop across the row or scan line. The latter is the proverbial inner loop, which is the bottle neck to high performance software.

If a target rectangle is large enough, or poorly located, part of the required memory may be within the video memory mapped into the CPU address space and part of it may not be addressable by the CPU without changing the mapping. It is desirable that the test for re-mapping the video memory is located outside of the inner loop.

This is typically accomplished by selecting the mapping offset of the start of video memory to the start of the CPU address space so that at least one entire row or scan line can be processed without changing the video memory mapping. There are currently no super VGAs that allow this offset to be specified on a byte boundary and there is a wide range among super VGAs in the ability to position a desired video memory location at the start of the CPU address space.

The number of bytes between the closest two bytes in video memory that can be placed on any single CPU address is defined as the granularity of the window mapping function. Some super VGA systems allow any 4k video memory boundary to be mapped to the start of the CPU address space, while other super VGA systems allow any 64k video memory boundary to be mapped to the start of the CPU address space. These two example systems would have granularities of 4k and 64k, respectively This concept is very similar to the bytes that can be accessed with a 16 bit pointer in an Intel CPU before a segment register must be changed (the granularity of the segment register or mapping, here is 16 bytes).

Note that if the granularity is equal to the length of the CPU address space, i.e. the least significant address bit of the hardware mapping function is more significant than the most significant bit of the CPU address, then the inner loop will have to contain the test for crossing the end or beginning of the CPU address space. This is because if the length of the CPU address space (which is the granularity in this case) is not evenly divisible by the length of a scan line, then the scan line at the end of the CPU address will be in two different video memory which cannot be mapped into the CPU address space simultaneously.

5.1.3 Operating on data from different areas

It is sometimes required or convenient to move or combine data from two different areas of video memory. One example of this is storing menus in the video memory beyond the displayed memory because there is hardware support in all VGAs for transferring 32 bits of video data with an 8 bit CPU read and write. Two separately mappable CPU video memory windows must be used if the distance between the source and destination is larger than the size of the CPU video memory window.

5.1.4 Combining data from two different windows

The above example of moving data from one CPU video memory window to another CPU video memory only required read access to one window and only required write access to the other window. Sometimes it is convenient to have read access to both windows and write access to one window. An example of this would be a raster operation where the resulting destination is the source data logically combined with the original destination data.

5.2 Different types of hardware windows

Different hardware implementations of CPU video memory windows can be supported by the *VESA BIOS extension*. The information necessary for an application to understand the type of hardware implementation is provided by the BIOS to the application. There are three basic types of hardware windowing implementations and they are described below.

The types of windowing schemes described below do not include differences in granularity.

Also note that is possible for a VGA to use a CPU address space of 128k starting at segment A000h

5.2.1 Single window systems

Some hardware implementations only provide a single window. This single window will be readable as well as writable. However, this causes a significant performance degradation when moving data in video memory a distance that is larger than the CPU address space.

5.2.2 Dual window systems

Many super VGAs provide two windows to facilitate moving data within video memory. There are two separate methods of providing two windows.

5.2.2.1 Overlapping windows

Some hardware implementations distinguish window A and window B by determining if the CPU is attempting to do a memory read or a memory write operation. When the two windows are distinguished by whether the CPU is trying to read or write they can, and usually do, share the same CPU address space. However, one window will be read only and the other will be write only.

5.2.2.2 Non-overlapping windows

Another mechanism used by two window systems to distinguish window A and window B is by looking at the CPU address within the total VGA CPU address space. When the two windows are distinguished by the CPU address within the VGA CPU address space the windows cannot share the same address space, but they can each be both read and written.

6. Extended VGA BIOS

Several new BIOS calls have been defined to support Super VGA modes. For maximum compatibility with the standard VGA BIOS, these calls are grouped under one function number. This number is passed in the AH register to the ***int 10h handler***.

The designated Super VGA extended function number is ***4Fh***. This function number is presently unused in most, if not all, VGA BIOS implementations. A standard VGA BIOS performs no action when function call 4F is made.

6.1 Status information

Every function returns status information in the AX register. The format of the status word is as follows:

AL ==	**4Fh:**	**Function is supported**
AL !=	**4Fh:**	**Function is not supported**
AH ==	**00h:**	**Function call successful**
AH ==	**01h:**	**Function call failed**

Software should treat a non-zero value in the AH register as a general failure condition. In later versions of the *VESA BIOS Extension* new error codes might be defined.

6.2 Function 00h - Return Super VGA information

The purpose of this function is to provide information to the calling program about the general capabilities of the Super VGA environment. The function fills an information block structure at the address specified by the caller. The information block size is 256 bytes.

Input:	AH= 4Fh	Super VGA support

AL= 00h **Return Super VGA information**
ES:DI= Pointer to buffer

Output: AX= Status
All other registers are preserved

The information block has the following structure:

```
VgaInfoBlock struc
        VESASignature           db      'VESA'                  ; 4 signature bytes
        VESAVersion             dw      ?               ; VESA version number
        OEMStringPtr            dd      ?               ; Pointer to OEM string
        Capabilities    db      4 dup (?)       ; capabilities of the video
environment
        VideoModePtr            dd      ?               ; pointer to supported Super
VGA modes
VgaInfoBlock ends
```

The ***VESASignature*** field contains the characters 'VESA' if this is a valid block.

The ***VESAVersion*** field specifies which VESA standard the Super VGA BIOS conforms to. The higher byte would specify the major version number. The lower byte would specify the minor version number. The initial VESA version number is 1.0. Applications written to use the features of a specific version of the *VESA BIOS Extension*, ***is guaranteed to work*** in later versions. The *VESA BIOS Extension* will be fully upwards compatible.

The ***OEMStringPtr*** is a far pointer to a null terminated OEM-defined string. The string may used to identify the video chip, video board, memory configuration etc., to hardware specific display drivers. There are no restrictions on the format of the string.

The ***Capabilities*** field describes what general features are supported in the video environment. The bits are defined as follows:

D0-31= Reserved

The ***VideoModePtr*** points to a list of supported Super VGA (VESA-defined as well as OEM-specific) mode numbers. Each mode number occupies one word (16 bits). The list of mode numbers is terminated by a -1 (0FFFFh). Please refer to chapter 2 for a description of VESA mode numbers. The pointer could point into either ROM or RAM, depending on the specific implementation. Either the list would be a static string stored in ROM, or the list would be generated at run-time in the information block (see above) in RAM.

6.3 Function 01h - Return Super VGA mode information

This function returns information about a specific Super VGA video mode. The function fills a mode information block structure at the address specified by the caller. The mode information block size is maximum 256 bytes.

Some information provided by this function is implicitly defined by the VESA mode number. However, some Super VGA implementations might support other video modes than those defined by VESA. To provide access to these modes, this function also returns various other information about the mode.

```
Input:      AH= 4Fh     Super VGA support
            AL= 01h     Return Super VGA mode information
            CX=         Super VGA video mode
            ES:DI=      Pointer to buffer

Output:     AX=         Status
            All other registers are preserved
```

The mode information block has the following structure:

```
ModeInfoBlock   struc

; mandatory information
        ModeAttributes[1]    dw    ?     ; mode attributes
        WinAAttributes[1]    db    ?     ; window A attributes
        WinBAttributes[1]    db    ?     ; window B attributes
        WinGranularity[1]    dw    ?     ; window granularity
        WinSize[1]           dw    ?     ; window size
        WinASegment[1]       dw    ?     ; window A start segment
        WinBSegment[1]       dw    ?     ; window B start segment
        WinFuncPtr[1]        dd    ?     ; pointer to window function
        BytesPerScanLine[1]        dw    ?     ; bytes per scan line
                                                 ; extended information
; optional information
        XResolution[2]             dw    ?     ; horizontal resolution
        YResolution[2]             dw    ?     ; vertical resolution
        XCharSize[2]         db    ?     ; character cell width
        YCharSize[2]         db    ?     ; character cell height
        NumberOfPlanes[2]    db    ?     ; number of memory planes
        BitsPerPixel[2]            db    ?     ; bits per pixel
        NumberOfBanks[2]     db    ?     ; number of banks
        MemoryModel[2]       db    ?     ; memory model type
        BankSize[2]          db    ?     ; bank size in kb

ModeInfoBlock   ends
```

The ***ModeAttributes*** field describes certain important characteristics of the video mode. ***Bit D0*** specifies whether this mode can be initialized in the present video configuration. This bit can be used to block access to a video mode if it requires a certain monitor type, and that this monitor is presently not connected. ***Bit D1*** specifies whether extended mode information is available. Video modes defined by VESA will have certain known characteristics, like resolution, number of planes, pixel format etc. Due to the severe space constraint for ROM based implementations of the *VESA BIOS Extension*, this information need not to be given for VESA-defined video modes. ***Bit D2*** indicates whether the BIOS have support for output functions like TTY output, scroll, pixel output etc. in this mode (it is recommended, but not mandatory, that the BIOS have support for all output functions).

The field is defined as follows:

D0= Mode supported in hardware
- 0= Mode not supported in hardware
- 1= Mode supported in hardware

D1= Extended information available
- 0= Extended mode information not available
- 1= Extended mode information available

D2= Output functions supported by BIOS
- 0= Output functions not supported by BIOS
- 1= Output functions supported by BIOS

D3= Monochrome/color mode (see note below)
- 0= Monochrome mode
- 1= Color mode

D4= Mode type
- 0= Text mode
- 1= Graphics mode

D5-D15= Reserved

Note: Monochrome modes have their CRTC address at 3B4h. Color modes have their CRTC address at 3D4h. Monochrome modes have attributes in which only bit 3 (video) and bit 4 (intensity) of the attribute controller output are significant. Therefore, monochrome text modes have attributes of off, video, high intensity, blink, etc. Monochrome graphics modes are two plane graphics modes and have attributes of off, video, high intensity, and blink. Extended two color modes that have their CRTC address at 3D4h, are color modes with one bit per pixel and one plane. The standard VGA modes, 06h and 11h would be classified as color modes, while the standard VGA modes 07h and 0fh would be classified as monochrome modes.

The ***BytesPerScanline*** field specifies how many bytes each logical scanline consists of. The logical scanline could be equal to or larger than the displayed scanline.

The ***WinAAttributes*** and ***WinBAttributes*** describe the characteristics of the CPU windowing scheme such as whether the windows exist and are read/writeable, as follows:

```
D0= Window supported
        0= Window is not supported
        1= Window is supported
D1= Window readable
        0= Window is not readable
        1= Window is readable
D2= Window writable
        0= Window is not writeable
        1= Window is writeable
D3-D7= Reserved
```

WinGranularity specifies the smallest boundary, in KB, on which the window can be placed in the video memory.

WinSize specifies the size of the window in KB.

WinASegment and ***WinBSegment*** address specify the segment addresses where the windows are located in the CPU address space.

WinFuncAddr specifies the address of the CPU video memory windowing function. The windowing function can be invoked either through ***VESA BIOS function 05h***, or by calling the function directly. A direct call will provide faster access to the hardware paging registers than using Int 10h, and is intended to be used by high performance applications.

The ***XResolution*** and ***YResolution*** specify the width and height of the video mode. In graphics modes, this resolution is in units of pixels. In text modes this resolution is in units of characters. Note that text mode resolutions, in units of pixels, can be obtained by multiplying XResolution and YResolution by the cell width and height, if the extended information is present.

The ***XCharCellSize*** and ***YCharCellSize*** specify the size of the character cell in pixels.

The ***NumberOfPlanes*** field specifies the number of memory planes available to software in that mode. For standard 16-color VGA graphics, this would be set to 4. For standard packed pixel modes, the field would be set to 1.

The ***BitsPerPixel*** field specifies the number of bits that define the color of one pixel. 16-color and 256-color graphics modes would specify 4 and 8 respectively. Non-standard memory organizations can be specified using this field and the *NumberOfPlanes* field. For example, a 16-color packed pixel mode would be described as having 1 plane and 4 bits per pixel.

The ***MemoryModel*** field specifies the general type of memory organization used in this mode. The following models have been defined:

00h=	**Text mode**
01h=	**CGA graphics**
02h=	**Hercules graphics**
03h=	**4-plane planar**
04h=	**Packed pixel**
05h=	**Non-chain 4, 256 color**
06h-0fh=	**Reserved, to be defined by VESA**
10h-ffh=	**To be defined by OEM**

NumberOfBanks. This is the number of banks in which the scan lines are grouped. The remainder from dividing the scan line number by the number of banks is the bank that contains the scan line and the quotient is the scan line number within the bank. For example, CGA graphics modes have two banks and Hercules graphics mode has four banks. For modes that don't have scanline banks (such as VGA modes 0Dh-13h), this field should be set to 1.

The ***BankSize*** field specifies the size of a bank (group of scan lines) in units of 1 KB. For CGA and Hercules graphics modes this is 8, as each bank is 8192 bytes in length. For modes that don't have scanline banks (such as VGA modes 0Dh-13h), this field should be set to 0.

6.4 Function 02h - Set Super VGA video mode

This function initializes a Super VGA video mode. The BX register contains the Super VGA mode to set. The format of VESA mode numbers is described in chapter 2. If the mode cannot be set, the BIOS should leave the video environment unchanged and return a failure error code.

```
Input:     AH= 4Fh     Super VGA support
           AL= 02h     Set Super VGA video mode
           BX=  Video mode
                D0-D14= Video mode
                D15= Clear memory flag
                     0= Clear video memory
                     1= Don't clear video memory

Output:    AX=  Status
           All other registers are preserved
```

6.5 Function 03h - Return current video mode

This function returns the current video mode in BX. The format of VESA video mode numbers is described in chapter 2 of this document.

```
Input:     AH= 4Fh     Super VGA support
           AL= 03h     Return current video mode

Output:    AX=  Status
```

BX= Current video mode
All other registers are preserved

Notes:

In a standard VGA BIOS, function 0Fh (Read current video state) returns the current video mode in the AL register. In D7 of AL, it also returns the status of the memory clear bit (D7 of 40:87). This bit is set if the mode was set without clearing memory. In this Super VGA function, the memory clear bit will ***not be returned*** in BX since the purpose of the function is to return the video mode only. If an application wants to obtain the memory clear bit, it should call VGA BIOS function Fh.

6.6 Function 04h - Save/Restore Super VGA video state

These functions provide a mechanism to save and restore the Super VGA video state. The functions are a superset of the three subfunctions under standard VGA BIOS function 1Ch (Save/restore video state). The complete Super VGA video state (except video memory) should be saveable/restorable by setting the requested states mask (in the CX register) to 000Fh.

Input: AH= 4Fh Super VGA support
AL= 04h Save/Restore Super VGA video state
DL= 00h **Return save/restore state buffer size**
CX= Requested states
D0= Save/restore video hardware state
D1= Save/restore video BIOS data state
D2= Save/restore video DAC state
D3= Save/restore Super VGA state

Output: AX= Status
BX= Number of 64-byte blocks to hold the state buffer
All other registers are preserved

Input: AX= 4Fh Super VGA support
AL= 04h Save/Restore Super VGA video state
DL= 01h **Save Super VGA video state**
CX= Requested states (see above)
ES:BX= Pointer to buffer

Output: AX= Status
All other registers are preserved

Input: AH= 4Fh Super VGA support
AL= 04h Save/Restore Super VGA video state
DL= 02h **Restore Super VGA video state**
CX= Requested states (see above)
ES:BX= Pointer to buffer

Output: AX= Status
All other registers are preserved

Notes:

Due to the goal of complete compatibility with the VGA environment, the standard VGA BIOS function 1Ch (Save/Restore VGA state) has not been extended to save the Super VGA video state. VGA BIOS compatibility requires that function 1Ch returns a specific buffer size with specific contents, in which there is no room for the Super VGA state.

6.7 Function 05h - CPU Video Memory Window Control

This function sets or gets the position of the specified window in the video memory. The function allows direct access to the hardware paging registers. To use this function properly, the software should use ***VESA BIOS Function 01h*** (Return Super VGA mode information) to determine the size, location and granularity of the windows.

Input: AH= 4Fh Super VGA support
AL= 05h **Super VGA video memory window control**
BH= 00h **Select super VGA video memory window**
BL= Window number
0= Window A
1= Window B
DX= Window position in video memory (in window granularity units)

Output: AX= Status
See notes below

Input: AH= 4Fh Super VGA support
AL= 05h **Super VGA video memory window control**
BH= 01h **Return super VGA video memory window**
BL= Window number
0= Window A
1= Window B

Output: AX= Status
DX= Window position in video memory (in window granularity units)
See notes below

Notes:

This function is also directly accessible through a far call from the application. The address of the BIOS function may be obtained by using VESA BIOS Function 01h, return Super VGA mode information. A field in the ModeInfoBlock contains the address of this function. Note that this function may be different among video modes in a particular BIOS implementation so the function pointer should be obtained after each set mode.

In the far call version, no status information is returned to the application. Also, in the far call version, the AX and DX registers will be destroyed.

Therefore if AX and/or DX must be preserved, the application must do so prior to making the far call.

The application must load the input arguments in BH, BL, and DX (for set window) but does not need to load either AH or AL in order to use the far call version of this function.

7. Application example

The following sequence illustrates how an application would interface to the *VESA BIOS Extension*. The hypothetical application is VESA-aware and calls the VESA BIOS functions. However, the application is not limited to supporting just VESA-defined video modes. Thus, it will inquire what video modes are available, before setting up the video mode.

1) The application would first allocate a 256 byte buffer. This buffer will be used by the VESA BIOS to return information about the video environment. Some applications will statically allocate this buffer, others will use system calls to temporarily obtain buffer space.

2) The application would then call ***VESA BIOS function 00h*** (Return Super VGA information). If the AX register does not contain 004Fh on return from the function call, the application can determine that the *VESA BIOS Extension* is not present and handle such situation.

   If no error code is passed in AX, the function call was successful. The buffer has been filled by the *VESA BIOS Extension* with various information. The application can verify that indeed this is a valid VESA block by identifying the characters 'VESA' in the beginning of the block. The application can inspect the ***VESAVersion*** field to determine whether the *VESA BIOS Extension* has sufficient functionality. The application may use the ***OEMStringPtr*** to locate OEM-specific information.

   Finally, the application can obtain a list of the supported Super VGA modes, by using the ***VideoModePtr***. This field points to a list of the video modes supported by the video environment.

3) The application would then create a new buffer and call the ***VESA BIOS function 01h*** (Return Super VGA mode information), to obtain information about the supported video modes. Using the ***VideoModePtr***, obtained in step 2 above, the application would call this function with a new mode number until a suitable video mode is found. If no appropriate video mode is found, its up to the application

to handle this situation.

The ***Return Super VGA mode information*** function fills a buffer specified by the application with information describing the features of the video mode. The data block contains all the information an application needs to take advantage of the video mode.

The application would examine the ***ModeAttributes*** field. To verify that the mode indeed is supported, the application would inspect bits D0. If D0 is cleared, then the mode is not supported by the hardware. This might happen if a specific mode requires a certain type of monitor, but that monitor is not present.

4) After the application has selected a video mode, the next step is to initialize the mode. However, the application might first want to save the present video mode. When the application exits, this mode would be restored. To obtain the present video mode, the ***VESA BIOS function 03h*** (Get Super VGA mode), would be used. If a non-VESA (standard VGA or OEM-specific) mode is in effect, only the lower byte in the mode number is filled. The upper byte is cleared.

5) To initialize the video mode, the application would use ***VESA BIOS function 02h*** (Set Super VGA mode). The application has from this point on full access to the VGA hardware and video memory.

6) When the application is about to terminate, it would restore the prior video mode. The prior video, obtained in step 4) above could be either a standard VGA mode, OEM-specific mode, or VESA-supported mode. It would reinitialize the video mode by calling ***VESA BIOS function 02h*** (Set Super VGA mode). The application would then exit.