**Video Information System (VIStm) BIOS  
API Specification**

**Tandy Corporation  
November 1992**

**Video Information System (VIStm) and the VIS logo are trademarks of Tandy Corporation.**

**Table of contents**

1. Introduction 1

2. AT BIOS 1

2.1. Floppy and hard disk services 1

2.2. Serial port services 1

2.3. System services 2

2.4. Keyboard services 3

2.5. Parallel port services 3

2.6. System timer and real-time clock services 4

2.7. Bootstrap loader 4

2.8. Miscellaneous services 4

3. Video BIOS 4

3.1. VGA-compatible BIOS functions 5

3.2. VESA super-VGA BIOS extensions 6

3.3. VIS video BIOS extensions 7

3.4. Notes 8

4. VIS/Multimedia BIOS Extensions 10

4.1. Introduction 10

4.2. System services 10

4.3. Hand controller services 12

4.4. Memory card services 13

4.5. Waveform output (D/A) services 15

4.6. Waveform input (A/D) services 17

4.7. Waveform input/output data structures 19

4.8. Synthesizer services 26

4.9. Mixer services 28

4.10. Timer Services 38

5. Notes 39

Appendix A - VIS Information 39

A.1. SINF/OEM chunk 40

A.2. SINF/AUDI chunk 40

A.3. Example 41

# 1. Introduction

Following is a specification of the VIS BIOS API. This BIOS is made up of several components. These are the AT BIOS, Video BIOS, and VIS/Multimedia BIOS extensions.

The AT BIOS is a subset of an AT-compatible BIOS with some PS/2-compatible additions.

The video BIOS is a subset of a VGA-compatible BIOS with VESA extensions and VIS-unique extensions.

The VIS/Multimedia BIOS extensions consist of newly defined functions to support new devices. These devices are waveform input/output, synthesizer, audio mixer, hand controller, and timer.

In this document 'user' refers to higher-level software that is using the BIOS support.

Unless specified otherwise, all functions assume the processor is operating in real mode.

# 2. AT BIOS

## 2.1. Floppy and hard disk services

Although the VIS does not include a floppy or hard drive the BIOS will provide an AT-compatible interface like that of any other diskless AT-compatible system.

Because there are no disk drives all INT 13h functions, except those listed below, will return CF=1 and AH=01h. The following functions are the exceptions.

AH=00h - Reset diskette system

This function will return CF=0 and AH=00h.

AH=01h - Read status of last operation

This function will operate normally and return the status of the last INT 13h operation.

AH=08h - Read drive parameters

This function will return information indicating that no drives are installed. When called with 00<=DL<=7Fh it will return AX=BX=CX=DX=ES=DI=0000h. When called with 80h<=DL<=0FFh it will return CF=1 and AH=01h.

AH=15h - Read DASD type

Return CF=0 and AH=00h.

## 2.2. Serial port services

The VIS does not include a serial port. However, a serial port is an option (for a modem). The BIOS will support the AT-compatible INT 14h functions. The supported functions are summarized below.

AH=00h - Initialize serial communications port

AH=01h - Send character

AH=02h - Receive character

AH=03h - Read serial port status

## 2.3. System services

The BIOS will support most AT-compatible INT 15h functions and some PS/2-compatible INT 15h functions. The supported functions are summarized below. All unsupported functions will return CF=1 and AH=86h.

AH=4Fh - Keyboard intercept

The BIOS provides a default handler for this function. The keyboard interrupt service routine will issue an INT 15h with AH=4Fh for every scan code it receives. Applications can hook this function to filter scan codes. This is an AT-compatible feature.

AH=85h - System request key pressed

The BIOS provides a default handler for this function. The keyboard interrupt service routine will issue an INT 15h with AX=8500h for key make and AX=8501h for key break. Applications can hook this function to detect system request key make and break activity. This is an AT-compatible feature.

AH=87h - Move block

The BIOS provides AT-compatible support for this function. However, applications should not use this function directly. Instead, applications should use the XMS interface for extended memory.

AH=88h - Extended memory size determine

The BIOS provides AT-compatible support for this function.

AH=0C0h - Return system configuration parameters

The BIOS provides AT-compatible support for this function.

AH=0C1h - Return extended BIOS data area segment address

The BIOS provides PS/2-compatible support for this function.

AH=0C2h - Pointing device BIOS interface

The BIOS provides PS/2-compatible support for the following subfunctions.

AL=00h - Enable/disable pointing device

AL=01h - Reset pointing device

AL=02h - Set sample rate

AL=03h - Set resolution

AL=04h - Read device type

AL=05h - Pointing device interface initialization

AL=06h - Extended commands

BL=00h - Return status

BL=01h - Set scaling to 1:1

BL=02h - Set scaling to 2:1

AL=07h - Device driver far call initialization

## 2.4. Keyboard services

The BIOS will provide an AT-compatible INT 16h interface to the keyboard. In the VIS system a keyboard is an option, not standard equipment.

The BIOS's keyboard ISR will issue INT 1Bh when Ctrl-Break is pressed. The ISR will issue INT 05h for Print Screen. However the default BIOS INT 05h routine will perform no action or alter any registers. The keyboard ISR will issue INT 15h with AX=8500h for System Request key make and AX=8501h for System Request key break. The ISR will enter a pause loop when the Pause key is pressed. The ISR will reset the machine when Ctrl-Alt-Del is pressed.

The keyboard hardware interface is not 100% compatible and applications that hook INT 09h and read scan codes directly may not operate correctly. It is recommended that applications wishing to process scan codes directly intercept them using the 'Keyboard Intercept' function AH=4Fh INT 15h.

The supported INT 16h functions are summarized below. All unsupported functions will return after subtracting 12h from AH.

AH=00h - Keyboard read

AH=01h - Keystroke status

AH=02h- Shift status

AH=03h - Set typematic rate

AH=05h - Keyboard write

AH=10h - Extended keyboard read

AH=11h - Extended keystroke status

AH=12h - Extended shift status

## 2.5. Parallel port services

The VIS does not include a parallel port. The equipment flag returned by INT 11h-Equipment Determination will reflect the lack of parallel ports.

If INT 17h is issued with AH<=2 then the service will return without performing any action or altering any registers. If INT 17h is issued with AH>2 then the service will return after subtracting 2 from AH.

BIOS support for parallel ports will be added if required by a future device.

## 2.6. System timer and real-time clock services

The system 8254-compatible timer will generate INT 08h as in AT-compatible systems. The BIOS INT 08h service routine will issue INT 1Ch every timer tick.

The BIOS will provide an AT-compatible INT 1Ah interface to the system timer. The supported INT 1Ah functions are summarized below. All unsupported functions will return after subtracting 1 from AH.

AH=00h - Read system timer time counter

AH=01h - Set system timer time counter

The VIS does not include a real-time clock. Real-time clock services are therefore not supported.

## 2.7. Bootstrap loader

At the end of the system power-on initialization routine, the BIOS will issue INT 19h to boot the OS.

The BIOS INT 19h and INT 18h services will perform no action and return without altering any registers. It is expected that the ROM-based OS will have hooked INT 19h during the ROM scan.

## 2.8. Miscellaneous services

The BIOS will provide AT-compatible support for these functions.

INT 11h - Equipment determination

INT 12h - Memory size determination

The value returned by this function may indicate less than 640 Kb of conventional RAM due to RAM used by the extended BIOS data area.

# 3. Video BIOS

The video support will have three components:

VGA-compatible BIOS functions.

VESA super-VGA BIOS extensions.

VIS video BIOS extensions.

The supported video modes are:

|  |  |  |
| --- | --- | --- |
| **Mode number** | **Resolution** | **Description** |
| 0D | 320x200 | 4-plane (16 color) |
| 0E | 640x200 | 4-plane (16 color) |
| 12 | 640x480 | 4-plane (16 color) |
| 13 | 320x200 | 8-bit packed pixel (256 color) |
| 59 | 320x200 | 4-plane color-repeat |
| 5A | 320x400 | 4-plane color-repeat |
| 5B | 640x200 | 4-plane color-repeat |
| 5C | 640x400 | 4-plane color-repeat |
| 5D | 640x200 | 8-bit packed pixel (256 color) |
| 5E | 640x400 | 8-bit packed pixel (256 color) |
| 5F | 320x400 | Direct color 5:6:5 |
| 60 | 384x256 | YUV-16 |
| 61 | 320x200 | YUV-8 |
| 62 | 320x200 | Direct color 1:5:5:5 |
| 63 | 320x200 | YUV-16 |
| 65 | 320x400 | 4-plane (16 color) |
| 66 | 320x400 | YUV-8 |
| 67 | 320x400 | Direct color 1:5:5:5 |
| 68 | 320x400 | YUV-16 |
| 6B | 640x400 | 4-plane (16 colors) |
| 6D | 320x200 | Overlay (256 color background, 15 color foreground) |
| 6F | 320x200 | Direct color 5:6:5 |
| 70 | 640x400 | YUV-8 |

## 3.1. VGA-compatible BIOS functions

The video support will include BIOS functions that are compatible with a standard VGA BIOS. Not all standard VGA BIOS functions will be implemented. Calls that deal only with alphanumeric modes are not supported. Calls that write to the EGA palette registers are not supported. Character and pixel functions are not supported (including character read/write, pixel read/write, scrolling, write TTY). The supported functions are listed here.

AH=00h - Set mode

AH=05h - Select active display page

AH=0Fh - Read current video state

AH=10h - Set palette registers

See the notes on the EGA palette on page 9.

AL=00h - Set individual palette register

AL=01h - Set overscan register

AL=02h - Set all palette registers and overscan

AL=03h - Toggle intensify/blinking bit

AL=07h - Read individual palette register

AL=08h - Read overscan register

AL=09h - Read all palette registers and overscan registers

AL=10h - Set individual color register

AL=12h - Set block of color registers

AL=13h - Set color page

AL=15h - Read individual color register

AL=17h - Read block of color registers

AL=1Ah - Read color page state

AL=1Bh - Sum color values to gray shades

AH=12h - Alternate select

BL=10h - Return EGA information

BL=31h - Default palette loading during mode set

BL=33h - Summing to gray shades

BL=36h - Video screen on/off

AH=1Ah - Read/Write display combination code

AH=1Bh - Return functionality/state information

AH=1Ch - Save/Restore video state

## 3.2. VESA super-VGA BIOS extensions

The video support will include an implementation of VESA Super VGA BIOS Extensions Standard 1.2. For completeness, the supported VESA functions are listed here. The reader should refer to the VESA Super VGA Standard version 1.2 for details on using these functions.

Function 00h - Return super VGA information

Function 01h - Return super VGA mode information

Function 02h - Set super VGA mode

Function 03h - Return current video mode

Function 04h - Save/Restore super VGA video state

Function 05h - CPU video memory window control

Function 06h - Set/Get logical scan line length

Function 07h - Set/Get display start

Function 08h - Set/Get DAC palette Control

The BIOS defines these OEM-defined super VGA memory models.

10h - Overlay

11h - YUV-16

12h - YUV-8

13h - 4-plane color repeat

These models are returned by VESA function 01h - 'Get super VGA mode information'.

## 3.3. VIS video BIOS extensions

The video support includes new functions defined to support special features of the VIS video system. These video services are accessed by executing INT 10h with AH=50h. Individual functions are selected by placing a function code in AL.

The functions will return with AL=50h and an error code in AH. Unless otherwise specified all other registers not used for return parameters are preserved through the function call. If an error occurs then registers that normally return parameters will have undefined values. AH=0 for no error. AH=1 for general failure.

Here is a summary of the functions provided:

AL=1 - Set interlace operation on/off

AL=2 - Hardware assisted color fill

AL=3 - Set vertical retrace interrupt interval

AL=6 - Set overlay mode transparent pixel value

AL=8 - Video switch

AL=1 - Set interlace operation on/off

In 200 line modes this function can be used to turn interlacing on or off.

In: DL = 0 - Interlace off

= 1 - Interlace on

Out:

AL=2 - Hardware assisted color fill

In 4-plane modes the hardware can fill a portion of video memory with a given color value.

BH=0 - Fill video memory with color

In: BL=0 - Return immediately after starting fill operation

=1 - Return after fill operation is complete

SI = start address in video memory

DI = end address in video memory (must be > SI)

DL = color value with which to fill

Out:

Both SI and DI must be even addresses.

This function can either return immediately after color fill is initiated or return after color fill is completed (depending on the value passed in BL). If the function returns immediately after color fill is initiated then the user can use the following function to determine when the operation is complete.

BH=1 - Test for fill complete

In:

Out: DL=0 - Color fill operation in progress

DL=1 - Color fill operation compete

When this function is called to perform a hardware color fill and return immediately, the application must not access the video system or make any other video BIOS calls except for 'Test for fill complete' until 'Test for fill complete' returns with DL=1 for 'Color fill operation complete'.

AL=3 - Set vertical retrace interrupt interval

The video hardware can interrupt the CPU at the start of a vertical retrace interval. This function can be used to set the number of vertical retraces that should occur between interrupts.

In: DL = Number of vertical retraces per interrupt (1-8)

Out:

AL=6 - Set overlay mode transparent pixel value

In overlay mode this function can be used to set the transparent pixel value. When overlay mode is set the transparent pixel value is 0.

In: DL = 4-bit transparent pixel value

Out:

AL=8 - Video switch

This function switches the video output between the VIS video and the external video input.

In: DL=0 - VIS video

=1 - External video

## 3.4. Notes

### 3.4.1. Overlay mode

Overlay mode contains three planes of image data, one background and two foreground (foreground 0 and foreground 1).

The displayed image is 320x200 pixels. The background is 8 bits per pixels (1 pixel per byte/256 colors) and therefore one complete background image occupies approximately 64K bytes of memory. The foreground is 4 bits per pixel (two pixels per byte/16 colors including transparent) and therefore one complete foreground image occupies approximately 32K bytes of memory. The background plane and the two foreground planes each have enough memory for two complete images (128Kb for the background, 64Kb for each foreground).

The background memory is accessed at CPU addresses A0000 to AFFFF. Foreground 0 is accessed at CPU addresses B0000 to B7FFF. Foreground 1 is accessed at CPU addresses B8000 to BFFFF. In other words, A0000 to AFFFF is the CPU window to the background memory. B0000 to B7FFF is the CPU window to the foreground 0 memory. B8000 to BFFFF is the CPU window to the foreground 1 video memory.

The VESA BIOS can be used to access various parts of the video memory. The VESA BIOS can be used to move the CPU window to other parts of the background memory. The CPU windows to the foreground memories will move correspondingly. For example, if the VESA BIOS is used to access the second 64K of the 128K background memory, then the second 32K of each foreground memory will be accesable through their CPU windows.

The VESA BIOS can also be used to change the logical dimensions of the screen. For example, the logical scan line width can be set to 500 pixels (only 320 are displayed). This allows 262 scan lines vertically (only 200 are displayed). Then horizontal or vertical panning can be accomplished easily by using the VESA BIOS.

Only one foreground plane is displayed at any given time. To select which foreground plane to display, use the standard VGA BIOS function AH=05h 'Select active display page' to set either page 0 or 1 to select foreground 0 or 1 respectively.

### 3.4.2. EGA palette

The standard EGA RAM palette is replaced by a ROM in the VIS system. Three different EGA ROM palettes are supported. These are:

|  |  |  |  |
| --- | --- | --- | --- |
| EGA palette registers | Modes D,E | Mode 12 and all 4-plane extended modes | Mode 13 and all non-4-plane extended modes |
| 0 | 00h | 00h | 00h |
| 1 | 01h | 01h | 01h |
| 2 | 02h | 02h | 02h |
| 3 | 03h | 03h | 03h |
| 4 | 04h | 04h | 04h |
| 5 | 05h | 05h | 05h |
| 6 | 06h | 14h | 06h |
| 7 | 07h | 07h | 07h |
| 8 | 10h | 38h | 08h |
| 9 | 11h | 39h | 09h |
| 10 | 12h | 3Ah | 0Ah |
| 11 | 13h | 3Bh | 0Bh |
| 12 | 14h | 3Ch | 0Ch |
| 13 | 15h | 3Dh | 0Dh |
| 14 | 16h | 3Eh | 0Eh |
| 15 | 17h | 3Fh | 0Fh |

These palette values are the same as those used as default settings for modes 0Dh, 0Eh, 12h, and 13h.

If necessary, users should alter the colors in the VGA palette instead of the EGA palette.

If the user calls any BIOS function that normally alters an EGA palette register, then the BIOS will see if one of the EGA ROM palette values matches the requested value. If one does then the BIOS will switch to that EGA ROM palette. Note that this may alter the value of other EGA palette registers. BIOS functions that alter the EGA palette registers are AH=10h/AL=00h - 'Set individual palette register' and AH=10h/AL=02h - 'Set all palette registers and overscan'.

# 4. VIS/Multimedia BIOS Extensions

## 4.1. Introduction

The VIS and multimedia BIOS extensions are all accessed by executing INT 15h with AH=71h, AL=device code, and BH=function code.

The functions will return with AL=71h and an error code in AH. Unless otherwise specified all registers not used for return parameters are preserved through the function call. If an error occurs then registers that normally return parameters will have undefined values. In general, AH=0 for no error, AH=1 for general failure, and AH=0FFh if the function is not supported.

The device codes (placed in AL) are

0 - System services

1 - Hand controller services

3 - Waveform output services

4 - Waveform input services

5 - Synthesizer services

6 - Mixer services

7 - Timer services

## 4.2. System services

These functions provide various miscellaneous services for the VIS system. The system services are accessed by executing INT 15h with AH=71h and AL=0. Individual functions are selected by placing a function code in BH.

Here is a summary of the functions provided by the system services.

0 - Get version information

1 - Get equipment list

2 - Get system information buffer size

3 - Get system information

BH=0 - Get version information

This function returns the version of the VIS BIOS specification supported by the BIOS (currently 1.0). It also returns the version of the BIOS.

In:

Out: CX = specification version

DX = code version

Major version number is returned in the more significant byte. Minor version number is returned in the less significant byte.

BH=1 - Get equipment list

This function returns information about various system components. The information is returned in a 256-byte user-supplied buffer.

In: DX:SI = pointer to buffer

Out:

The caller should set all bytes within the buffer to 0 before calling this function.

The buffer information has the following format.

dwEquipmentFlags dd ? ; Equipment flags

The **dwEquipmentFlags** is a set of flags used to indicate the presence of devices, support of particular functions, or the format of additional buffer information. Currently these bits are defined.

<0> - Keyboard attached

0 - Keyboard not attached

1 - Keyboard attached

<1> - Mouse attached

0 - Mouse not attached

1 - Mouse attached

<31:2> - Reserved

All reserved bits should be ignored until defined. All other bytes returned in the buffer are reserved and should be ignored until defined.

BH=2 - Get system information buffer size

This function is used to determine the size of the buffer required by function BH=3 - 'Get system information'.

In:

Out: CX = Size of buffer (in bytes)

BH=3 - Get system information

This function fills a user supplied buffer with information describing the system. Basically this function serves the same purpose as function 1 but is more flexible and extendable. The format of the system information is described in appendix A on page 38. The user should first call function 2 to determine the size of the buffer required by this function.

In: DX:SI = Pointer to buffer

Out:

The caller should set all bytes within the buffer to 0 before calling this function.

## 4.3. Hand controller services

The hand controller is a device that has a nine position joystick (center, North, Northeast, East, Southeast, South, Southwest, West, Northwest) and 6 additional function switches.

The BIOS provides an interrupt driven API such that applications can be notified when there are changes in the switch states or joystick position. This relieves the application of polling for switch state changes.

These functions provide various services for the hand controller. The hand controller services are accessed by executing INT 15h with AH=71h and AL=1. Individual functions are selected by placing a function code in BH.

Here is a summary of the functions provided by the hand controller services.

0 - Get hand controller information

1 - Read hand controller state

2 - Get switch change handler callback address

3 - Set switch change handler callback address

BH=0 - Get hand controller information

This function returns information about the hand controller. The information is returned in a 256-byte user-supplied buffer.

In: DX:SI = pointer to buffer

Out:

The caller should set all bytes within the buffer to 0 before calling this function.

The buffer information has the following format.

dwHandInfoFlags dd ? ; Hand controller information flags

The **dwHandInfoFlags** is a set of flags used to indicate support of particular functions or the format of additional buffer information. Currently all bits are reserved and should be ignored until defined.

All other bytes returned in the buffer are reserved and should be ignored until defined.

Calling this function also causes the BIOS to reset the hand controller system. This function should be called at least once before all other hand controller functions so that the BIOS can reset the hand controller system.

BH=1 - Read hand controller state

In:

Out: CX = wHand1Switches

DX = wHand2Switches

**wHandxSwitches** indicates the state of the hand controller. The bits of wHandxSwitches are defined as follows. A bit set to 1 indicates that a switch is pressed.

<0> - Down

<1> - F1

<2> - Primary

<3> - F2

<4> - F4

<5> - Secondary

<6> - F3

<7> - Left

<8> - Up

<9> - Right

<15:10> - reserved

BH=2/3 - Get/Set switch change handler address

These functions are used to install a user-supplied switch change handler. The BIOS will service the hand controller hardware and read the hand controller state information. The BIOS will pass the state information to the user's switch change handler.

BH=2 - Get switch change handler address

In:

Out: DX:SI - pointer (segment:offset) to switch change handler installed with last call to 'Set switch change handler address'

BH=3 - Set switch change handler address

In: DX:SI = pointer (segment:offset) to switch change handler

Out:

The switch change handler is entered by a far call from the BIOS to the switch change handler address. It should return by a far return. Before calling the handler the BIOS places the hand controller state information in register DX and CX as described in the 'Read hand controller state' function. The switch change handler may be called from within a hardware interrupt service routine and therefore should be as quick as possible. The switch change handler must preserve all registers except for AX.

In the get/set switch change handler address functions the handler address can be null (segment:offset=0000:0000). When the handler address is null the BIOS will not call the handler.

Applications should first get and save the current handler address before installing their own. Upon exiting, the application should restore the original address.

## 4.4. Waveform output (D/A) services

A waveform output device provides a means to playback digital audio data.

Several of these functions use the waveform header structure. This structure is defined in detail in the waveform header section on page 18. Also, a status word within the header is maintained by the BIOS during playback. The user can monitor this status as described in the discussion on the waveform header.

Although unlikely, it is possible for a system to include several waveform output devices. If there are N devices in the system then they are numbered from 0 to N-1.

The waveform output services are accessed by executing INT 15h with AH=71h and AL=3. Individual functions are selected by placing a function code in BH. Here is a summary of the functions provided by the waveform output services.

0 - Get number of waveform output devices

1 - Get waveform output device information

2 - Play waveform

3 - Pause waveform playback

4 - Resume waveform playback

5 - Stop waveform playback

6 - Break waveform playback loop

BH=0 - Get number of waveform output devices

This function returns the number of waveform output devices.

In:

Out: BL = Number of devices

BH=1 - Get waveform output device information

This function returns information about a given waveform output device. The information is returned in a 256-byte user-supplied buffer.

In: BL = Device number

DX:SI = Pointer to buffer

Out:

The caller should set all bytes within the buffer to 0 before calling this function.

The buffer information has the following format.

dwWaveOutInfoFlags dd ? ; Waveform output information flags

lpFormatList dd ? ; Pointer to list of supported formats

The **dwWaveOutInfoFlags** is a set of flags used to indicate support of particular functions or the format of additional buffer information. Currently all bits are reserved and should be ignored until defined.

The pointer **lpFormatList** may point into ROM or RAM (possibly into another part of the buffer itself). It points to a list of the supported audio data formats. The list is made of words terminated by a word -1. The interpretation of the values in the list is the same as the wFormat field of the waveform header described on page 18.

All other bytes returned in the buffer are reserved and should be ignored until defined.

BH=2 - Play waveform

This function queues a request to play a block of waveform data.

In: BL = Device number

DX:SI = Pointer to waveform header structure

Out:

Possible error codes are:

2 - Unsupported format

3 - Device busy

Because some of the hardware resources are often shared among waveform playback and recording services it may be impossible to satisfy a playback request due to current recording activity. In this case this function will return a error code of 3 for 'Device busy'.

If there are no playback requests currently playing and there are no other requests queued and the waveform output is not paused then this request will immediately go into service. Otherwise the request is queued and will not be serviced until all previous requests are serviced. If the user wants to guarantee that this request be serviced immediately then a call can first be made to 'Stop waveform playback' before 'Play waveform'.

The waveform header must not be altered by the user until playback is completed or stopped. To determine when it is safe to destroy the structure the user can monitor the playback status updated by the BIOS in the waveform header.

BH=3 - Pause waveform playback

This function pauses playback of the current request. (The current request is the request in service.)

In: BL = Device number

Out:

The status of the current request is updated. The status of queued requests is not updated.

Calling this function when there is no request in service will cause waveform output to remain paused until 'Resume waveform playback' is called even if there is an intervening call to 'Play waveform'.

BH=4 - Resume waveform playback

This function resumes playback after pause has been called.

In: BL = Device number

Out:

The status of the current request is updated. The status of queued requests is not updated.

Calling this function when playback is already in progress has no effect and returns no error.

BH=5 - Stop waveform playback

This function stops playback and flushes the request queue.

In: BL = Device number

Out:

The status of all requests is updated.

Calling this function when already stopped has no effect and returns no error.

BH=6 - Break waveform playback loop

This function breaks playback of a loop within the current request.

In: BL - Device number.

Out:

The status of the current request is updated.

Calling this function when playback is not in a loop has no effect and returns no error.

## 4.5. Waveform input (A/D) services

A waveform input device provides a means to record digital audio data.

Several of these functions use the waveform header structure. This structure is defined in detail in the waveform header section on page 18. Also, a status word within the header is maintained by the BIOS during recording. The user can monitor this status as described in the discussion on the waveform header.

Although unlikely, it is possible for a system to include several waveform input devices. If there are N devices in the system then they are numbered from 0 to N-1.

The waveform output services are accessed by executing INT 15h with AH=71h and AL=4. Individual functions are selected by placing a function code in BH. Here is a summary of the functions provided by the waveform input services.

0 - Get number of waveform input devices

1 - Get waveform input device information

2 - Record waveform

3 - Pause waveform recording

4 - Resume waveform recording

5 - Stop waveform recording

BH=0 - Get number of waveform input devices

This function returns the number of waveform input devices.

In:

Out: BL - Number of devices

BH=1 - Get waveform input device information

This function returns information about a given waveform input device. The information is returned in a 256-byte user-supplied buffer.

In: BL = Device number

DX:SI = Pointer to buffer

Out:

The caller should set all bytes within the buffer to 0 before calling this function.

The buffer information has the following format.

dwWaveInInfoFlags dd ? ; Waveform input information flags

lpFormatList dd ? ; Pointer to list of supported formats

The **dwWaveInInfoFlags** is a set of flags used to indicate support of particular functions or the format of additional buffer information. Currently all bits are reserved and should be ignored until defined.

The pointer **lpFormatList** may point into ROM or RAM (possibly into another part of the buffer itself). It points to a list of the supported audio data formats. The list is made of words terminated by a -1. The interpretation of the values in the list is the same as the wFormat field of the waveform header described on page 18.

All other bytes returned in the buffer are reserved and should be ignored until defined.

BH=2 - Record waveform

This function queues a request to record a block of waveform data.

In: BL = Device number

DX:SI = Pointer to waveform header structure

Out:

Possible error codes are:

2 - Unsupported format

3 - Device busy

Because some of the hardware resources are often shared among waveform playback and recording services it may be impossible to satisfy a record request due to current playback activity. In this case this function will return a error code of 3 for 'Device busy'.

If there are no record requests currently recording and there are no other requests queued and waveform input is not paused then this request will immediately go into service. Otherwise the request is queued and will not be serviced until all previous requests are serviced. If the user wants to guarantee that this request be serviced immediately then a call can first be made to 'Stop waveform recording' before 'Record waveform'.

The waveform header must not be altered by the user until recording is completed or stopped. To determine when it is safe to destroy the structure the user can monitor the recording status maintained by the BIOS in the waveform header.

BH=3 - Pause waveform recording

This function pauses recording of the current request. (The current request is the request in service.)

In: BL = Device number

Out:

The status of the current request is updated. The status of queued requests is not updated.

Calling this function when there is no request in service will cause waveform input to remain paused until 'Resume waveform recording' is called even if there is an intervening call to 'Record waveform'.

BH=4 - Resume waveform recording

This function resumes recording after pause has been called.

In: BL = Device number

Out:

The status of the current request is updated. The status of queued requests is not updated.

Calling this function when recording is already in progress has no effect and returns no error.

BH=5 - Stop waveform recording

This function stops recording and flushes the request queue.

In: BL = Device number

Out:

The status of all requests is updated.

Calling this function when already stopped has no effect and returns no error.

## 4.6. Waveform input/output data structures

Here is a description of the data structures and formats used by waveform input and output functions.

### 4.6.1. Waveform header

The waveform header is used by the waveform input and output functions and has the following structure.

dwFlags dd ? ; Various flags

wMarker dw ? ; Playback position marker

wStatus dw ? ; Status

lpCallback dd ? ; Pointer to callback function

wFormat dw ? ; Format of audio data

wStartThreshold dw ? ; Auto-start threshold

reserved db 112 dup(0) ; Reserved - pad to 128 bytes with 0

db ? ; Start of control block list (see description of

; control blocks below)

**dwFlags** is 32 bits of various flags. Currently all are reserved and must be set to 0.

The user initializes **wStatus** to 0. The BIOS will update this word as the playback or record request is processed. The interpretation of bits <14:0> is as follows.

0000 - Initial value set by user

0001 - Playback/record request is waiting in queue

0002 - Playback/record in progress

0003 - Playback/record paused

0004 - Playback is in continuous loop

0005 - Recording request is waiting for sound

4000 - Playback/record is complete

Bit 15 of wStatus will be set to 1 when the BIOS no longer requires the waveform header. The user can safely destroy the waveform header after verifying that bit 15 of wStatus is 1.

The user should initialize **wMarker** to 0. The driver will update wMarker during playback. The use of wMarker is described in the waveform playback/record control blocks section on page 19.

The user initializes **lpCallback** to point to a user-supplied routine. Unless lpCallback is null the BIOS will call this routine every time it updates wStatus or wMarker. The callback may be made from within an interrupt service routine. Therefore the user's callback routine should be as quick as possible. The user's callback routine may alter AX, BX, CX, DX, BP, SI, DI, DS, or ES. The last callback will result from the BIOS setting bit 15 of wStatus to indicate 'wave header done'. This means that the wave header is no longer required by the BIOS and the user can reuse the RAM allocated to the wave header. When the 'wave header done' callback is made interrupts will be disabled (interrupts are enabled for all other callbacks). The user callback routine is responsible for reenabling interrupts (STI) as soon as possible after getting control.

The **wFormat** field specifies the format of the audio data to be played or recorded. The possible formats are described in the discussion of audio data formats on page 21. The buffers for the audio data are specified in the playback/record control blocks discussed on page 19.

**wStartThreshold** is used to allow recording to begin automatically when a sound occurs. This helps to prevent a long period of silence at the beginning of a recording. Since it is rare to have absolute silence, this value is used to determine how loud a sound must be before recording will begin. This value specifies in sample units how far a sample must deviate from absolute silence before recording begins. For example, if the system is to record 8-bit PCM and wStartThreshold is set to 4 then recording will not begin until a sample occurs such that the sample is either less than 7Dh or greater than 83h. A wStartThreshold value of 0 defeats the auto-start function.

The reserved area in the waveform header fills the header to offset 128 and should be initialized to 0 by the user. This area is reserved for future enhancements and/or a scratch pad for the BIOS.

Following the reserved area is the start of a list of control blocks. These blocks control the playback or recording of audio data. The next section describes the control blocks.

### 4.6.2. Waveform playback/record control blocks

Starting at offset 128 of the waveform header is a list of playback/record control blocks. These blocks control the playback or recording of audio data and essentially form a script that the BIOS will follow while playing or recording audio data.

Each block starts with a one byte block type identifier followed by a signature and block length. The signature is used by the BIOS to verify that a control block is valid. The block length is the size of the control block excluding the block type identifier, signature, and block length fields. The data following the block length field is dependent upon the block type. The BIOS will process a block and then proceed to the next block type identifier using the block length field.

Control block type 0 - Terminator

This control block marks the end of the list of control blocks. When processing reaches this block the request has been satisfied and the next request in the queue can be serviced.

bBlkType db 0 ; Terminator block

bBlkSig db 'W' ; Signature

wBlkLen dw 0 ; No data follows

Control block type 1 - Audio data

This control block specifies playback or recording of audio data.

bBlkType db 1 ; Audio data block

bBlkSig db 'W' ; Signature

wBlkLen dw 8 ; Size of this control block

lpAudioData dd ? ; Pointer to buffer for audio data

dwAudioLen dd ? ; Length of the audio data (in bytes)

**lpAudioData** points to a buffer for actual audio data. The format of the data within the buffer is dependent on the value in the wFormat field of the waveform header. These formats are specified later in 'Audio data formats' on page 21.

**dwAudioLen** specifies the size in bytes of the audio data buffer pointed to by lpAudioData. The size of the audio data should be the equivalent of at least 10 milliseconds playback or record time. Smaller lengths may result in distortion.

For playback lpAudioData should point to audio data to be played by the BIOS. The BIOS will use the value in wFormat to determine how to play the audio data and the value in dwAudioLen to determine how much to play.

For recording lpAudioData points to an empty buffer to be filled with recorded data by the BIOS. The BIOS will use the value in wFormat to determine how to record the data and the value in dwAudioLen to determine how much data to record.

Control block type 2 - Marker

This control block specifies a position marker. When the BIOS encounters a position marker during processing it will update the wMarker field in the waveform header with the marker value stored here. The user sets the value of the marker to any value he wishes. Marker blocks can be used to monitor playback or recording progress for synchronization.

bBlkType db 2 ; Marker block

bBlkSig db 'W' ; Signature

wBlkLen dw 2 ; Size of this control block

wMarker dw ? ; User-defined marker

Control block type 3 - Dummy

This control block can be used to store user-definable information. The BIOS ignores this block and moves on to the next block.

bBlkType db 3 ; Dummy block

bBlkSig db 'W' ; Signature

wBlkLen dw xx ; Length of user data

UserData db xx dup(?) ; User data

Control block type 4 - Loop start

This control block identifies the start of a repeat loop. All blocks between this block and the next 'Loop end' block are repeated a given number of times. This control block should be used only for playback.

bBlkType db 4 ; Loop start

bBlkSig db 'W' ; Signature

wBlkLen dw 4 ; Size of this control block

dwLoopCount dd ? ; Number of times to repeat loop

The **dwLoopCount** field can be set to (-1) to indicate continuous looping. If dwLoopCount is set to 0 then the BIOS will not process any of the control blocks within the loop. The loop, even if not continuous, can be broken by a call to 'Break playback loop'.

Control block type 5 - Loop end

This control block identifies the end of a repeat loop. All blocks between this block and previous 'Loop start' block are repeated. See block type 4. If there is no previous 'Loop start' block then this block is ignored. This control block should be used only for playback.

bBlkType db 5 ; Loop end

bBlkSig db 'W' ; Signature

wBlkLen dw 0 ; Always 0

Control block type 6 - Silence

This control block indicates an interval of silence. This control block should be used only for playback.

bBlkType db 6 ; Silence

bBlkSig db 'W' ; Signature

wBlkLen dw 4 ; Size of this control block

dwSilentLen dd ? ; Length of silence (measured in sample periods)

The length of the silence must be the equivalent of at least 10 milliseconds of playback time. Smaller lengths may result in distortion.

Control block type 7 - Pause

This control block causes playback or recording to enter the pause state. Playback or recording will remain paused until one of the 'Resume' or 'Stop' functions is called.

bBlkType db 7 ; Pause

bBlkSig db 'W' ; Signature

wBlkLen dw 2 ; Size of this control block

wMarker dw ? ; User-defined marker

The wMarker field is used the same as in 'Control block type 2 - Marker'.

Control block type 8 - Goto

This control block contains a pointer to the next control block. This allows a non-contiguous list of control blocks.

bBlkType db 8 ; Goto

bBlkSig db 'W' ; Signature

wBlkLen dw 4 ; Size of this control block

lpNextBlk dd ? ; Pointer to next control block

### 4.6.3. Audio data formats

There are several possible formats for the audio data. The audio data format is specified in the wFormat field of the waveform header. The possible formats follow. **It is not required that all VIS machines support all formats listed here.**

0 - 5.0125 KHz, mono, 8-bit PCM

1 - 5.0125 KHz, stereo, 8-bit PCM

2 - 5.0125 KHz, mono, 16-bit PCM

3 - 5.0125 KHz, stereo, 16-bit PCM

4 - 11.025 KHz, mono, 8-bit PCM

5 - 11.025 KHz stereo, 8-bit PCM

6 - 11.025 KHz, mono, 16-bit PCM

7 - 11.025 KHz, stereo, 16-bit PCM

8 - 22.05 KHz, mono, 8-bit PCM

9 - 22.05 KHz, stereo, 8-bit PCM

10 - 22.05 KHz, mono, 16-bit PCM

11 - 22.05 KHz, stereo, 16-bit PCM

12 - 44.1 KHz, mono, 8-bit PCM

13 - 44.1 KHz, stereo, 8-bit PCM

14 - 44.1 KHz, mono, 16-bit PCM

15 - 44.1 KHz, stereo, 16-bit PCM

The storage scheme of the audio data is independent of the sample rate. The sample rate only affects the playback or record rate. Therefore storage schemes are only defined for 8-bit mono PCM, 8-bit stereo PCM, 16-bit mono PCM, and 16-bit stereo PCM.

8-bit mono PCM

db ? ; Sample 0

db ? ; Sample 1

;... ; ...continuing for all samples...

Each 8-bit sample is treated as an unsigned integer. Sample 0 must start on a double word address.

8-bit stereo PCM

db ? ; Channel 0 (left) sample 0

db ? ; Channel 1 (right) sample 0

db ? ; Channel 0 (left) sample 1

db ? ; Channel 1 (right) sample 1

;... ; ...continuing for all samples...

Each 8-bit sample is treated as an unsigned integer. Channel 0 (left) sample 0 must start on a double word address.

16-bit mono PCM

dw ? ; Sample 0

dw ? ; Sample 1

;... ; ...continuing for all samples...

Each 16-bit sample is treated as a signed integer. Sample 0 must start on a double word address.

16-bit stereo PCM

dw ? ; Channel 0 (left) sample 0

dw ? ; Channel 1 (right) sample 0

dw ? ; Channel 0 (left) sample 1

dw ? ; Channel 1 (right) sample 1

;... ; ...continuing for all samples...

Each 16-bit sample is treated as a signed integer. Channel 0 (left) sample 0 must start on a double word address.

### 4.6.4 Examples

Here is an example of how to use the waveform data structures defined above. These examples are given in the context of audio playback.

Example 1

This example illustrates a simple use of the waveform header.

WaveHeader label byte

dd 0 ; dwFlags - Reserved - set to 0

dw 0 ; wMarker - initialize to 0

dw 0 ; wStatus - initialize to 0

dd 0 ; lpCallback - NULL for no callbacks

dw 4 ; wFormat - I want to play 11.025 KHz 8-bit mono PCM audio

dw 0 ; wStartThreshold - not used for playback

db 112 dup(0) ; reserved - initialize to 0

; Start list of control blocks. These blocks make up a script that the BIOS follows during

; playback or recording.

; Play 20 seconds of 11.025 KHz 8-bit mono PCM audio...

db 1 ; bBlkType - control block type 1 - audio

db 'W' ; bBlkSign - valid control block

dw 8 ; wBlkLen - control block length (less wBlkLen and bBlkType fields)

dd MyAudioData1 ; lpAudioData - pointer to my 11.025Khz 8-bit mono PCM samples

dd 20\*11025 ; dwAudioLen - length of MyAudioData1 (20 seconds of data)

; Terminate playback.

db 0 ; bBlkType - terminator

db 'W' ; bBlkSign - valid control block

dw 0 ; wBlkLen - no bytes follow

Example 2

This example illustrates a more complex use of the waveform header.

WaveHeader label byte

dd 0 ; dwFlags - Reserved - set to 0

dw 0 ; wMarker - initialize to 0

dw 0 ; wStatus - initialize to 0

dd MyCallback ; lpCallback - point to my callback routine

dw 11 ; wFormat - I want to play 22.05 KHz 16-bit stereo PCM audio

dw 0 ; wStartThreshold - not used for playback

db 112 dup(0) ; reserved - initialize to 0

; Start list of control blocks. These blocks make up a script that the BIOS follows during

; playback or recording.

; Play 5 seconds of 22.05 KHz 16-bit stereo PCM audio...

db 1 ; bBlkType - control block type 1 - audio

db 'W' ; bBlkSign - valid control block

dw 8 ; wBlkLen - control block length (less wBlkLen and bBlkType fields)

dd MyAudioData1 ; lpAudioData - pointer to my 22.05Khz 16-bit stereo PCM samples

dd 5\*22050\*2\*2 ; dwAudioLen - length of MyAudioData1 (5 seconds of data)

; Update the wMarker field in the waveform header and call my callback routine if I've installed

; one.

db 2 ; bBlkType - marker block

db 'W' ; bBlkSign - valid control block

dw 2 ; wBlkLen - 2 bytes follow

dw 1234h ; wMarker - marker value that I made up

; Start a playback loop. Everything from here to the next 'Loop end' control block will be

; repeated.

db 4 ; bBlkType - loop start

db 'W' ; bBlkSign - valid control block

dw 4 ; wBlkLen - 4 bytes follow

dd 10 ; dwLoopCount - I want to loop 10 times

; Play 3 seconds of 22.05 KHz 16-bit stereo PCM audio. This will be repeated 10 times due to

; the surrounding loop start and loop end control blocks.

db 1 ; bBlkType - audio block

db 'W' ; bBlkSign - valid control block

dw 8 ; bBlkLen 8 bytes follow

dd MyAudioData2 ; lpAudioData - pointer to my 22.05 KHz 16-bit stereo PCM samples

dd 3\*22050\*2\*2 ; dwAudioLen - length of MyAudioData2 - 3 seconds of data

; Set the end of the loop. Everything from the last 'Loop start' control block to here will

; be repeated.

db 5 ; bBlkType - loop end

db 'W' ; bBlkSign - valid control block

dw 0 ; wBlkLen - no bytes follow

; Terminate playback.

db 0 ; bBlkType - terminator

db 'W' ; bBlkSign - valid control block

dw 0 ; bBlkLen - no bytes follow

## 4.7. Synthesizer services

The synthesizer services provide a means to playback MIDI music data by supporting functions that process MIDI messages and produce the corresponding sounds using system synthesizer hardware resources.

The synthesizer services are accessed by executing INT 15h with AH=71h and AL=5. Individual functions are selected by placing a function code in BH. Here is a summary of the functions provided by the synthesizer services.

0 - Get synthesizer information

1 - Reset synthesizer

2 - Send MIDI data

3 - reserved

4 - Open synthesizer device

5 - Close synthesizer device

Upon power-up and after calls to 'Reset synthesizer', the synthesizer will operate in a Multimedia PC (MPC) complatible mode. That is, it will operate as either a base-level or extended-level synthesizer as defined by the MPC standard. Base-level synthesizers respond on MIDI channels 13 to 16 with percussion on 16. Extended-level synthesizers respond on MIDI channels 1 to 10 with percussion on 10.

The General MIDI System Level 1 specification defines system exclusive messages to turn a General MIDI system on or off. These are TURN GENERAL MIDI SYSTEM ON and TURN GENERAL MIDI SYSTEM OFF. When the synthesizer receives the TURN GENERAL MIDI SYSTEM ON message then it will switch from MPC compatible mode to General MIDI (GM) compatible mode. When in GM mode, the synthesizer will respond on all MIDI channels 1 to 16 with percussion on 10. Note that even in GM mode the synthesizer may not satisfy the polyphony required by the General MIDI System Level 1 specification. When the synthesizer receives the TURN GENERAL MIDI SYSTEM OFF message it will return to MPC mode.

BH=0 - Get synthesizer information

This function returns information about the music synthesizer. The information is returned in a 256-byte user-supplied buffer.

In: DX:SI = Pointer to buffer

Out:

The caller should set all bytes within the buffer to 0 before calling this function.

The buffer information has the following format.

dwSynthInfoFlags dd ? ; Information flags.

bVoices db ? ; Maximum number of simultaneous voices.

bNotes db ? ; Maximum number of simultaneous notes.

dwSysexFormats dd ? ; Supported sysex messages.

The **dwSynthInfoFlags** is a set of flags used to indicate support of particular functions or the format of additional buffer information. Currently all bits are reserved and should be ignored until defined.

**bVoices** contains the maximum number of simultaneous voices. **bNotes** contains the maximum number of simultaneous notes. In most synthesizer implementations, bVoices and bNotes will be equal.

**dwSysexFormats** indicates which system exclusive messages are supported.

Bit 0 - General MIDI System On/Off (must be supported)

Bit 1 - OPL3 Patch Download

All implementations support the General MIDI System On/Off messages and therefore bit 0 will always be 1. Bit 1 is set if the implementation supports the OPL3 Patch Download message. Refer to the General MIDI System Level 1 specification for information on the General MIDI System On/Off messages. Information on the OPL3 Patch Download message will be provided separately.

BH=1 - Reset synthesizer

This function resets the synthesizer hardware to its initial state. The synthesizer returns to MPC mode and all downloaded patches are cleared.

BH=2 - Send MIDI data

This function is used to pass system exclusive MIDI messages and/or multiple short MIDI messages to the BIOS for processing. The data passed can be any number of bytes and does not need to be a complete MIDI message. This function can only be used while the synthesizer device is open. See the 'Open synthesizer device' and 'Close synthesizer device' functions.

In: DX:SI = Pointer to buffer containing MIDI message data

CX = Length of buffer (bytes)

Out:

BH=4 - Open synthesizer device

This function is used to inform the BIOS that the user wishes to use the synthesizer. This function, used in conjunction with the 'Close synthesizer device' function, helps the BIOS prepare the system to use the synthesizer.

In:

Out:

Possible error codes are:

2 - Device already open

BH=5 - Close synthesizer device

This function is used to inform the BIOS that the user no longer wishes to use the synthesizer. This function, used in conjunction with the 'Open synthesizer device' function, helps the BIOS prepare the system to use the synthesizer.

In:

Out:

Possible error codes are:

3 - Device already closed

Notes

It is unlikely that the internal synthesizer will support all MIDI message. Unsupported messages are ignored. The internal synthesizer will conform to the General MIDI Level 1 Specification to the largest extent reasonably possible. This includes conforming to the standard MIDI patch assignments.

## 4.8. Mixer services

### 4.8.1 Overview

The mixer API is intended to give application software information about the audio devices in the sytem and the interconnections between them, and provide a means to control the processing of the audio signals.

Audio devices

An audio device is a device that either produces an audio signal as output or receives an audio signal as an input. To distinguish between the two types, we use the terms audio 'source' and audio 'sink'. An audio source produces an audio signal while an audio sink receives, or listens to, an audio signal.

We use the terms 'source' and 'sink' instead of 'input' and 'output' because these may be ambiguous depending on the point of view. For example, a microphone input jack is an audio source while a waveform input device (or waveform recorder) is an audio sink.

It is possible for a device to be both a source and a sink. For example, a mixer receives audio signals as input (audio sink) and produces audio signals as output (audio source).

The mixer model

A mixer is an audio device that allows several input audio signals to be combined into one or more output signals. It also allows some processing of the signals before and after combining them.

An input to the mixer is called an input line. An output from the mixer is called an output line. The mixer has one or more input lines and one or more output lines.

Zero or more of the input line signals are combined into each output line. For example, output line 0 might be a mixture of input lines 0, 2, and 3, while output line 1 might be a mixture of input lines 1 and 3.

Typically mixers can process the input and output signals. Usually this processing consists of, at least, a volume control. A control that processes an input line signal is called an input control. A control that processes an output line signal is called an output control.

A mixer input or output line is connected to other audio devices. A mixer has audio sources such as CD audio and waveform playback connected to its input lines. A mixer has audio sinks such as speakers or waveform recorders connected to its output lines. In most cases, the connection between a mixer line and another audio device is constant. In other cases, the device connected to a mixer line is software selectable. The connection between a mixer and another device is called a 'patch'. If the connected device is software selectable then the line is said to be 'patchable'.

The mixer API

The mixer API can be subdivided into two sets of functions. First, a set of functions is provided to provide information about the interconnection of the various audio devices in the system. The functions allow an application to construct a logical model of the connections between the audio devices. In these functions, a mixer is essentially just another audio device, albeit both a source and a sink. These functions are purely for obtaining information and perform no real action.

Second, a set of functions is provided for manipulating the mixer. Functions are provided for determining mixer capabilities, getting or setting various control levels, and selecting the devices on patchable input and output lines.

Under this API, the system can have up to 32 audio sources and 32 audio sinks.

The mixer services are accessed by executing INT 15h with AH=71h and AL=6. Individual functions are selected by placing a function code in BH. Here is a summary of the functions provided by the mixer services.

1 - Get audio source device information

2 - Get audio sink device information

3 - Get number of mixers

4 - Get mixer information

5 - Reset mixer

6 - Get line information

7 - Get line connections

8 - Set line connections

9 - Get line control

0Ah - Set line control

0Bh - Get line patch

0Ch - Set line patch

### 4.8.2 Device functions

These functions provide information about the interconnections between the various audio devices in the system.

BH=1 - Get audio source device information

This function returns information about the audio source devices.

This function uses a 256-byte user-supplied buffer.

In: DX:SI = pointer to buffer

Out:

The buffer information has the following format.

wReserved dw ? ; Reserved

wDeviceType dw ? ; Device type

dwConnectMap dd ? ; Device connections

This structure is repeated in the buffer for each source device - up to 32 source devices - to form a list of source devices. The first structure corresponds to source 0, the second to source 1, and so on.

The BIOS returns in **wDeviceType** the type of audio device. This word is broken into three fields. Bits<2:0> are used to distinguish between several devices of the same type. For example, if there are four CD players in the system, then these three bits would have the value 0 through 3 to identify which CD player. Bits<4:3> describe the mono/stereo characteristics of the device and are defined as follows:

00 - Stereo device

01 - Mono device - only right channel connected

10 - Mono device - only left channel connected

11 - Mono device - both left and right channels connected

Bits<15:5> identify the type of device. Device types are described in a later section. A device type of 0 indicates the end of the list and represents no device.

The BIOS returns in **dwConnectMap** a map indicating to which audio sinks the source is connected. For example, if the source is connected to sink devices 5 and 10 then bits 5 and 10 of dwConnectMap will be 1 and all other bits will be 0. A source can be connected to multiple sinks.

BH=2 - Get audio sink device information

This function returns information about the audio sink devices.

This function uses a 256-byte user-supplied buffer.

In: DX:SI = pointer to buffer

Out:

The buffer information has the following format.

wReserved dw ? ; Reserved

wDeviceType dw ? ; Device type

dwConnectMap dd ? ; Device connections

This structure is repeated in the buffer for each sink device - up to 32 sink devices - to form a list of sink devices. The first structure corresponds to sink 0, the second to sink 1, and so on.

The BIOS returns in **wDeviceType** the type of audio device. This word is broken into three fields. Bits<2:0> are used to distinguish between several devices of the same type. For example, if there are four waveform recorders in the system, then these three bits would have the value 0 through 3 to identify which waveform recorder. Bits<4:3> describe the mono/stereo characteristics of the device and are defined as follows:

00 - Stereo device

01 - Mono device - only right channel connected

10 - Mono device - only left channel connected

11 - Mono device - both left and right channels connected

Bits<15:5> identify the type of device. Device types are described in a later section. A device type of 0 indicates the end of the list and represents no device.

The BIOS returns in **dwConnectMap** a map indicating to which audio sources the sink is connected. For example, if the sink is connected to source device 1 then bit 1 of dwConnectMap will be 1 and all other bits will be 0.

### 4.8.3 Mixer functions

BH=3 - Get number of mixers

This function returns the number of mixers in the system. If there are N mixers in the system, then they are numbered 0 to N-1. This function would return N.

In:

Out: BL = number of mixers

BH=4 - Get mixer information

This function returns information about a mixer.

This function uses a 256-byte user-supplied buffer.

In: BL = mixer number

DX:SI = pointer to buffer

Out:

The buffer information has the following format.

dwFlags dd ? ; Various flags

bNumInputLines db ? ; Number of input lines

bNumOutputLines db ? ; Number of output lines

The user should set all bytes within the buffer to 0 before calling this function.

**dwFlags** is a set of flags used to indicate support of particular functions or the format of additional buffer information. Currently all bits are reserved and should be ignored until defined.

The BIOS returns in **bNumInputLines** the number of input lines in the mixer.

The BIOS returns in **bNumOutputLines** the number of output lines in the mixer.

BH=5 - Reset mixer

This functions resets a mixer to its power-up default state.

In: BL = mixer number

Out:

BH=6 - Get line information

This function returns information about a particular input or output line of a mixer.

This function uses a 256-byte user-supplied buffer.

In: BL=mixer number

DX:SI = pointer to buffer

Out:

The buffer information has the following format.

dwFlags dd ? ; Various flags

bLineNumber db ? ; Line number

dwLinePatches dd ? ; Possible patches

dwLineControls dd ? ; Supported controls

dwStaticDevices dd ? ; Static devices (always connected)

dwPatchableDevicesSet0 dd ? ; First set of patchable devices (one can be connected)

dwPatchableDevicesSet1 dd ? ; Second set of patchable devices (one can be connected)

dwPatchableDevicesSet2 dd ? ; Third set of patchable devices (one can be connected)

dwPatchableDevicesSet3 dd ? ; Fourth set of patchable devices (one can be connected)

dwConnectableLines dd ? ; Connectable lines

Before calling this function the user should set **bLineNumber** to specify the mixer line about which to obtain information. Bit 7 indicates whether the line is an input or output line (0=output, 1=input). Bits 6 through 0 contain a line number.

All other bytes within the buffer should be set to 0 before calling this function.

The BIOS returns in **dwFlags** a set of flags that indicate support of particular functions or the format of additional buffer information. Currently only bit 0 is defined. All other bits should be ignored until defined.

<0> - Patchable line

0 - Line is not patchable

1 - Line is patchable

If a line is patchable then the devices connected to it are selectable by the application software.

The BIOS returns in **dwLinePatches** a bit map of the devices that *can* be connected to the mixer line. In this bit map, bit 0 corresponds to audio device 0, bit 1 to audio device 1, and so on. If the bit is '1' then the device can be connected to the line. If the line is patchable then one or more of these devices can be connected to it. If the line is not patchable then the devices indicated here are always connected to the line.

If bLineNumber is an input line then the devices indicated in dwLinePatches are source devices in the device list returned by 'Get audio source device information'. If bLineNumber is an output line then the devices indicated in dwLinePatches are sink devices in the list of sink devices returned by 'Get audio sink device information'.

The BIOS returns in dwStaticDevices a bit map of the devices that are always connected to the mixer line. The BIOS returns in dwPatchableDevicesSet0 a bit map of devices, one of which can be connected to the mixer line. The same applies to each of dwPatchableDevicesSet1, Set2, and Set3.

The BIOS returns in dwConnectableLines a bit map of the lines that can be connect to the mixer line. If bLineNumber is an input line, then dwConnectableLines specifies the output lines to which it can be connected. If bLineNumber is an output line, then dwConnectableLines specifies the input lines to which it can be connected.

The BIOS returns in **dwLineControls** a bit map of the supported controls. If bit 0 is '1' then control type 0 is supported on the line, etc. Control types are defined in a later section.

BH=7 - Get line connections

This function returns the mixer lines to which a given mixer line is connected. If the caller is inquiring about an input line, then this function returns the output lines to which it is connected. If the caller is inquiring about an output line, then this function returns the input lines to which it is connected.

This function uses a 256-byte user supplied buffer.

In: BL = mixer number

DX:SI = pointer to buffer

Out:

The buffer information has the following format.

dwFlags dd ? ; Various flags

bLineNumber db ? ; Line number

dwLineConnections dd ? ; Connections to lines

Before calling this function the user sets **bLineNumber** to specify the mixer line for which to get connections. Bit 7 indicates whether the line is an input or output line (0=output, 1=input). Bits 6 through 0 contain a line number.

All other bytes within the buffer should be set to 0 before calling this function.

**dwFlags** is a set of flags used to indicate support of particular functions or the format of additional buffer information. Currently all bits are reserved and should be ignored until defined.

The BIOS returns in **dwLineConnections** a bit map of the output (input) lines to which the input (output) line is connected.

BH=8 - Set line connections

This function sets the mixer lines to which a given mixer line is connected. If the caller is setting connections for an input line, then this function connects the given output lines. If the caller is setting connections for an output line, then this function connects the given input lines.

This function uses a 256-byte user supplied buffer.

In: BL = mixer number

DX:SI = pointer to buffer

Out:

The buffer information has the following format.

dwFlags dd ? ; Various flags

bLineNumber db ? ; Line number

dwLineConnections dd ? ; Connections to lines

Before calling this function the user sets **bLineNumber** to specify the mixer line for which to set the connections. Bit 7 indicates whether the line is an input or output line (0=output, 1=input). Bits 6 through 0 contains a line number.

Before calling this function the user sets **dwLineConnections**. dwLineConnections is a bit map of the output (input) lines to which the input (output) line is connected. The BIOS will return the actual connections set in dwLineConnections. This may be different than the requested connections if these cannot be set..

**dwFlags** is a set of flags that alter the behavior of the function or specify the format of additional buffer information. Currently only bit 0 is defined as a test flag. All other bits are reserved and should be set to 0 before calling this function. If bit 0, the test flag, is set to 1 then this function will return without acutally changing the line connections but will still return in dwLineConnections the connections that would have been set.

All other bytes within the buffer should be set to 0 before calling this function.

BH=9 - Get line control

This function returns the current level of a line control.

This function uses a user-supplied 256-byte buffer.

In: BL = mixer number

DX:SI = pointer to buffer

Out:

The buffer information has the following format.

dwFlags dd ? ; Various flags

bLineNumber db ? ; Line number

bControl db ? ; Control number

wControlLevel dw ? ; Control level

Before calling this function the user sets **bLineNumber** to specify the mixer line for which to get a control level. Bit 7 indicates whether the line is an input or output line (0=output, 1=input). Bits 6 through 0 contain a line number.

Before calling this function the user sets **bControl** to the number of the control for which to get the level. Controls and their numbers are defined in a later section.

All other bytes within the buffer should be set to 0 before calling this function.

**dwFlags** is a set of flags used to indicate support of particular functions or the format of additional buffer information. Currently all bits are reserved and should be ignored until defined.

The BIOS returns in **wControlLevel** the current level of the control specified in bControl. The interpretation of the level is dependent on the control and is defined in a later section.

BH=0Ah - Set line control

This function sets the level of a line control.

This function uses a 256-byte user supplied buffer.

In: BL = mixer number

DX:SI = pointer to buffer

Out:

The buffer information has the following format.

dwFlags dd ? ; Various flags

bLineNumber db ? ; Line number

bControl db ? ; Control number

wControlLevel dw ? ; Control level

Before calling this function the user sets **bLineNumber** to specify the mixer line for which to set a control level. Bit 7 indicates whether the line is an input or output line (0=output, 1=input). Bits 6 through 0 contains a line number.

Before calling this function the user sets **bControl** to the number of the control for which to set the level. Controls and their numbers are defined in a later section.

Before calling this function the user sets **wControlLevel** to the desired level of the control specified in bControl. The interpretation of the level is dependent on the control and is defined in a later section. The BIOS will return the actual control level set in wControlLevel. This may be different than the requested control level if the requested level cannot be set.

**dwFlags** is a set of flags that alter the behavior of the function or specify the format of additional buffer information. Currently only bit 0 is defined as a test flag. All other bits are reserved and should be set to 0 before calling this function. If bit 0, the test flag, is set to 1 then this function will return without acutally changing the control level but will still return in wControLevel the level that would have been set.

All other bytes within the buffer should be set to 0 before calling this function.

BH=0Bh - Get line patch

This function returns the device(s) currently connected to a line.

This function uses a 256-byte user supplied buffer.

In: BL = mixer number

DX:SI = pointer to buffer

Out:

The buffer has the following format.

dwFlags dd ?

bLineNumber db ?

dwLinePatches dd ?

Before calling this function the user sets **bLineNumber** to specify the mixer line for which to get the current patches. Bit 7 indicates whether the line is an input or output line (0=output, 1=input). Bits 6 through 0 contains a line number.

All other bytes within the buffer should be set to 0 before calling this function.

The BIOS returns in **dwLinePatches** a bit map of the devices currently patched to the line. If bLineNumber is an input line then these bits refer to the devices returned by 'Get audio source device information'. If bLineNumber is an output line then these bits refer to the devices returned by 'Get audio sink device information'.

**dwFlags** is a set of flags used to indicate support of particular functions or the format of additional buffer information. Currently all bits are reserved and should be ignored until defined.

BH=0Ch - Set line patch

This function sets the device(s) to patch to a line.

This function uses a 256-byte user supplied buffer.

In: BL = mixer number

DX:SI = pointer to buffer

Out:

The buffer has the following format.

dwFlags dd ?

bLineNumber db ?

dwLinePatches dd ?

Before calling this function the user sets **bLineNumber** to specify the mixer line for which to set patches. Bit 7 indicates whether the line is an input or output line (0=output, 1=input). Bits 6 through 0 contains a line number.

Before calling this function the user sets **dwLinePatches**. dwLinePatches is a bit map of the source (sink) devices to connect to the input (output) line specified in bLineNumber. The BIOS returns in dwLinePatches the actual patches connected. This may be different than the requested patches if the requested patches cannot be connected.

**dwFlags** is a set of flags that alter the behavior of the function or specify the format of additional buffer information. Currently only bit 0 is defined as a test flag. All other bits are reserved and should be set to 0 before calling this function. If bit 0, the test flag, is set to 1 then this function will return without acutally changing the line patches but will still return in dwLinePatches the patches that would have been connected.

All other bytes within the buffer should be set to 0 before calling this function.

### 4.8.4 Device types

Here is a list of the currently defined device types

0 - No connection

1 - CD audio

2 - Microphone

3 - Mixer

4 - PC speaker

5 - Telephone

6 - Television audio

7 - Waveform player/recorder

8 - Line input/output jack

9 - Speaker jack

10 - Headphone jack

11 - Synthesizer

Some devices are sources. Some are sinks. Some can be both sources and sinks.

### 4.8.5 Controls

A control modifies the audio signal entering a mixer input line or exiting a mixer output line. Here is a list of the currently defined control types.

0 - Volume

1 - Volume (individual left and right)

2 - Mute

3 - Crossover

Control levels are specified as 16-bit words. The interpretation of a control level depends on the control type in question.

Volume controls

Volume controls are, of course, used to control the loudness of an audio signal. The relative loudness of audio signals is often measured in decibels (dB). Under this API, a volume control has 256 levels ranging from 0 to 255 where 0 is the loudest and 255 is the softest. A change in one volume control unit corresponds to a change of 0.5 dB in the signal.

Given this, the API allows a volume control to have a range of 128 dB and a resolution of 0.5 dB. Practically, real volume controls have less range and less resolution. The BIOS is responsible for programming the volume control hardware to achieve a volume level as close to the desired volume level as possible. When a user calls the 'Set control level' function, he specifies a control level in the wControlLevel parameter. The actual level achieved by programming the hardware is returned in the wControlLevel parameter. Note that when a user calls the 'Get control level' function, the value returned in wControlLevel is the value last requested on a successful call to 'Set control level'.

There are two types of volume controls. The first, control type 0, controls the volume of both left and right channels together. In this case the lower byte of the control level is the volume level. The upper byte is ignored.

The second type of volume control, control type 1, controls the volumes of the left and right channels separately. In this case the lower byte of the control level specifies the left volume level and the upper byte specifies the right volume level.

Mute control

The mute control is used to temporarily mute the audio signal. When the control level is 0 the signal is not muted. When the control level is non-zero the signal is muted.

Crossover

The crossover control is used to individually map left and right channels to neither, either, or both channels. Only bits <3:0> are used in the control level. The bits are interpreted as follows.

0 - connect left to left

1 - connect left to right

2 - connect right to left

3 - connect right to right

Given this, the normal setting would be binary 1001. To swap left and right channels, the crossover would be 0110. To play only the left channel through both channels the crossover should be 0011.

It is possible for the hardware to support some but not all of the crossover combinations. If the BIOS is called to set a crossover that is not supported, then the call will either fail or the BIOS will set a different crossover. If the BIOS sets a different crossover than requested then the new crossover is returned in the wControlLevel parameter in the 'Set control level' call.

## 4.9. Timer Services

The timer services allow applications to safely use the system timer interrupt for timing purposes.

The timer services are accessed by executing INT 15h with AH=71h and AL=7. Individual functions are selected by placing a function code in BH. Here is a summary of the functions provided by the timer services.

0 - Get timer information

1 - Install timer callback

BH=0 - Get timer information

This function returns information about the timer services. The information is returned in a 256-byte user-supplied buffer.

In: DX:SI = Pointer to buffer

Out:

The caller should set all bytes within the buffer to 0 before calling this function.

The buffer information has the following format.

dwTimerInfoFlags dd ? ; Information flags.

The **dwTimerInfoFlags** is a set of flags used to indicate support of particular functions or the format of additional buffer information. Currently all bits are reserved and should be ignored until defined.

All other bytes returned in the buffer are reserved and should be ignored until defined.

BH=1 - Install system timer callback

This function is used to install a timer callback function. This function should be used instead of hooking the timer interrupt (INT 08h / IRQ 0) directly.

In: DX:SI = Pointer to callback function.

CX = Desired callback period in milliseconds.

Possible error codes are:

2 - Invalid period

The user sets CX to the desired callback period in milliseconds. For example, if this function is called with CX=45 then the callback function will be invoked every 45 milliseconds. The BIOS will not allow callbacks more often than every 4 ms. If the user specifies an invalid period, then the callback is cleared (set to null) and the install is ignored.

If the pointer to the callback function is null the BIOS will not call the callback function. Applications should set a null callback function before exiting.

The callback routine must return via a far return and can alter any of the registers AX, BX, CX, DX, BP, SI, DI, DS, or ES. The callback routine does not need to service the timer or interrupt controller hardware. The callback routine is entered with interrupts cleared (CLI) . The callback routine, unless very short, should enable interrupts (STI). If the callback routine uses much stack space then it should switch to its own internal stack. If the callback routine enables interrupts then it should be prepared to be reentered. This can happen if, for example, the callback period is set to 30 ms yet the callback routine takes more than 30 ms to execute.

# 5. Notes

The system does not include a real-time clock device. Therefore the INT 15h functions AH=83h Event Wait and AH=86h Wait cannot be supported. The real-time clock interrupt 70h and alarm interrupt 4Ah are not generated by the system hardware or BIOS.

The system power-on initialization routine will perform a scan of the adapter ROM space. This scan will be performed the same as an AT-compatible BIOS except that the address range is only in the 64 Kb space from 0C0000h to 0CFFFFh.

The BIOS does not provide multitasking provisions implemented by INT 15h functions AH=90h and AH=91h. That is, the BIOS neither issues these interrupt function requests nor provides any special service routines for them.

INT 15h joystick functions AH=84h are not supported.

Applications should not hook INT 08h or program the 8254 timer chip. Application should instead use the timer services documented on page 36.

# Appendix A - VIS Information

This section describes the format of the system information returned by the VIS system information function. The buffer format is based on the RIFF format described in the Multimedia Development Kit Programmer's Reference Chapter 8. This format is used for its generality and extensibility.

The information is returned in a buffer that is divided into chunks. The format of a chunk is as follows.

ChunkID db 4 dup(?) ; 4 character chunk ID.

dwChunkSize dd xx ; Size of chunk in bytes

ChunkData db xx dup(?) ; Chunk data

ChunkID is a four byte ASCII character code that identifies the information contained within the chunk. IDs that are less than four characters are padded on the right with spaces. There are no embedded spaces within the ID.

dwChunkSize contains the size of the chunk data. This does not include the ChunkID or dwChunkSize fields.

The ChunkData is word aligned relative to the start of the buffer. If dwChunkSize is odd then a pad byte of 0 is added to the end of ChunkData so that the next ChunkData is also word aligned. dwChunkSize does not include the pad byte if present.

The format of the chunk data is dependent upon the ChunkID and is defined as each ChunkID is defined. Often, the chunk data is divided into smaller chunks, each having the same general format described above. These smaller chunks can be further subdivided into yet smaller chunks. The various chunks and subchunks form a tree structure.

A chunk can be specified by its path from the root of the chunk tree. The root chunk is always 'SINF' for System INFormation.

A program that wishes to obtain information would scan the chunk IDs for one that it recognizes. It would skip unfamiliar chunks using the size fields.

The following sections describe the various chunk IDs and the associated data.

## A.1. SINF/OEM chunk

This chunk contains information identifying the manufacturer of the system.

The OEM chunk is made up of two subchunks having the IDs OID and OINF. Specifying a path from the root of the chunk tree, these would be SINF/OEM/OID and SINF/OEM/OINF.

The first data word of the SINF/OEM/OID chunk contains a numeric ID that uniquely identifies the manufacturer of the system. The data following the numeric ID is an ASCIIZ string that contains the descriptive name of the product.

The format of the data in the SINF/OEM/OINF chunk is defined by the manufacturer identified by the numeric ID in SINF/OEM/OID.

## A.2. SINF/AUDI chunk

This chunk contains information about the audio system.

The SINF/AUDI chunk contains the GMB subchunk (SINF/AUDIO/GMB). The presence of this chunk indicates that the system includes a synthesizer BIOS interface (GMB=General MIDI BIOS). All VIS systems include a synthesizer BIOS interface. The first data word of this chunk indicates the level of BIOS support, currently 1.

If the SINF/AUDI/OPL3 chunk is present then the system includes the OPL3 synthesizer chip. The first data word of the chunk contains the port base address of the synthesizer chip. The byte at offset 2 from the start of the chunk data indicates the IRQ line to which the chip is connected. If the value is 0xFF, then the chip is not connected to an IRQ.

The SINF/AUDI chunk may also contain other subchunks describing other parts of the audio system. These will be defined when necessary.

## A.3. Example

Here is an example information buffer:

db 'SINF' ; 4

dd 74 ; 4

db 'OEM ' ; 4

dd 36 ; 4

db 'OID ' ; 4

dd 18 ; 4

db 0 ; 1

db 'Tandy VIS System',0 ; 17

db 'OINF' ; 4

dd 1 ; 4

db 0 ; 1

db 0 ; Pad

db 'AUDI' ; 4

dd 22 ; 4

db 'GMB ' ; 4

dd 1 ; 4

db 1 ; 1

db 0 ; Pad

db 'OPL3' ; 4

dd 3 ; 4

dw 388h ; 2

db 7 ; 1

db 0 ; Pad