MXM
GRAPHICS MODULE

Specification

MXM Version 2.1 Graphics Module Software Specification

September 2007
SP-03494-001_v1.0

Document Change History

Document Number Version	Date	Reason for Change
1.0	September 10, 2007	Initial Release Made following changes from MXM 2.0 to MXM 2.1 Added support for DisplayPort Added support for high definition audio Added summary of required structures and methods Various typos and clarifications

Table of Contents

List of Tables

List of Figures

Applicable Documents

The following documents contain provisions which through reference in this text, constitute provisions of this standard. At the time of publication, the editions indicated were valid. However, users of this standard are advised to ensure they have the latest versions of referenced standards and documents.

- SP-03493-001 – *MXM version 2.1 Graphics Module Thermal Electromechanical Specification*
- http://www.acpi.info/ - *Advanced Configuration and Power Interface Specification* Revision 3.0a

Software Control of the MXM

The MXM v 2.1 Graphics Module™ (MXM) software includes a video BIOS (VBIOS) and an OS specific driver.

The VBIOS is stored in an EPROM located on the MXM v 2.1 graphics module. Merging the VBIOS with the system BIOS (SBIOS) in a single ROM on the motherboard is not supported with MXM v 2.1, since the module is removable.

Required system support for MXM v 2.1 software includes a set of MXM v 2.1 structures which define the System Information and include information such as:

- Display device output configurations
- TV output format
- MXM v 2.1 heat sink thermal rating
- System power supply capabilities

This structure is stored either in a separate MXM System Information ROM on the motherboard or in a table within the SBIOS. In the case of a separate ROM, the VBIOS reads the ROM through the MXM connector using the DDCC serial link. In the case of a table merged into the SBIOS, the VBIOS will obtain a pointer to the table via an INT 15h call to the SBIOS at POST time.

Other system specific information relating to the graphics adapter, such as the Plug & Play Sub-System Vendor ID or Sub-System Device ID are outside of the scope of the MXM specification. They must be configured prior to loading MXM VBIOS or driver and must work even when the adapter is secondary display adapter (for example non-VGA or no VBIOS POST). Contact the PCI/PCIe SIG, or the GPU vendor for more details on programming these Plug & Play registers.

The MXM v 2.1 software will interpret the contents of the MXM v 2.1 structure. The operating system does not need any knowledge of the MXM. Advanced control of the MXM v 2.1 graphics module, including calibration and control of thermal sensors, is handled by the Display Driver. If system thermal control requires thermal data from the MXM v 2.1, it can be provided to the system through several mechanisms (for example, to the SBIOS), via processes which are specified later in this document.

Note: The panel and backlight power will be controlled by the MXM VBIOS and drivers.

Because the MXM v 2.1 module is removable, the SBIOS should not rely on specific properties, such as a fixed PCI device ID for the module.

MXM Structure

The MXM v 2.1 structure consists of a mandatory header structure followed by a variable number of substructures. This is followed by a mandatory checksum byte at the end. Data is assumed to be in little Endian format.

MXM Header Structure

Table 1 lists the header at the beginning of the MXM structure.

Table 1. MXM Header Structure

Offset	Field Definition	Description
0000 – 0003	"MXM_"	"MXM" header string
0004	Version = 0x02	MXM v 2.1 structure version number. Hex value
0005	Revision = 0x01	MXM v 2.1 structure revision number. Hex value
0006 - 0007	MXM structure length	Byte length of MXM structure where length includes the checksum but does not include the MXM header

Note: In the case where multiple structures are embedded in a ROM, the next structure will be located immediately after the last byte of the current structure. Computing the number of bytes to skip is version specific. Refer to the relevant version MXM specification for details on computing total size.

This does not apply to structures returned via system methods. System methods returns only a single structure based on a caller supplied version number.

MXM Versioning and Interoperability

When an MXM adapter is installed into a system, the resulting system capabilities are the intersection set of the MXM adapter and the base platform. Only the capabilities that are present in both will be available. For example, if A) the MXM supports the High-Definition Multimedia Interface (HDMI™) output technology and B) the base platform supports routing HDMI to an HDMI connector, then that likely means that HDMI can be supported. However, if the base platform (including dock) did not have an HDMI connector then no HDMI capability should be expected regardless of the MXM adapters' capabilities.

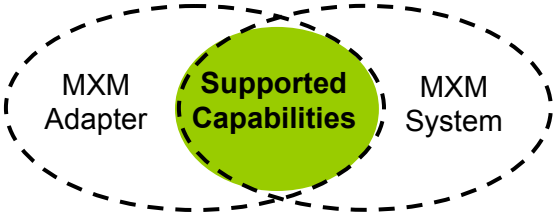

Figure 1. MXM Capabilities

Both the MXM adapter and the base platform convey software version information indicating the highest level of the MXM v 2.1 software interface that each supports. Between the MXM v 2.1 software interfaces, backwards compatibility within a major version is required. For example, a 2.x platform shall support any adapter 2.x ~ 2.0. Forward compatibility is not required. However, all structures and members, within 2.x will remain backward compatible with previous 2.x implementations.

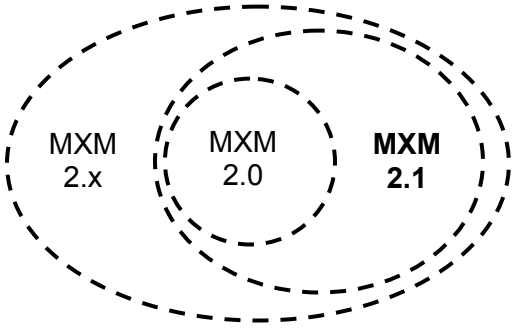

Figure 2. MXM v 2.1 Software Compatibility

MXM v 2.1 Interface Requirements

Prior to operating system or driver loading, the VBIOS may need to have access to all MXM v 2.1 functionality in the system as well as in the adapter. Additionally, MXM adapters may need to operate as secondary display adapters (VGA resources are disabled, or it is enumerated as a non-VGA device), meaning that the VBIOS may not be available. Therefore, Int15h methods alone are insufficient. Both the Int15h and ACPI interfaces are required to be supported when system methods are used for output and DDC selection. Refer to sections: *MXM v 2.1 INT 15H System BIOS Callbacks* and *MXM v 2.1 ACPI Methods* for descriptions on the Int15h and ACPI system methods.

The following summarizes the required software support:

- MXM Structure
  - Header and Checksum
  - At least one Output Device structure if the adapter has an output
  - Cooling capability structure
  - At least one Input Power structure
  - Backlight structure if GPU PWM backlight is used
- Int15h system methods
  - Func 0 & Func 1 are required methods
  - Func 2 is required if an EDID less internal flat panel is required
  - If DDC or Output Mux is used with system methods then Func 3 is required
- ACPI system methods
  - MXMI and MXMS are required methods
  - If a Display DDC or Output Mux is used with system methods, the MXMX method is required for each display on the mux

Unless otherwise noted, structures and methods not listed above are optional and not required.

MXM Output Device Structure

There will be multiple MXM output device structures (Table 2), defining one output device each. An output device shall be enumerated for each supported integrated display and display connector. For example, each DVI output that can be routed to either an onboard connector or a docking connector must have one dedicated output device structure for each connector. In the case of a DVI-I output connector, there are separate output device structures for the analog and digital outputs.

The **DESCRIPTOR** field defines what kind of structure information is being stored. A descriptor of 0x00 means this is an output device structure. Each device entry is a 48 bit field, which defines the device type, connection and DDC port (if applicable) of the device.

Note: Field Bits [27:23] are overloaded and have different meanings depending on whether an analog TV or digital display is being enumerated.

The ordering of Output Devices implies default boot device, and the detection order. For example the first Output Device is the default boot display device. The second Output Device will be the boot display device if the first is not attached. The third if the second is not attached, and so on.

Note: The type, topology connection and meaning of the enumerated outputs shall match what is enumerated through other system interfaces including ACPI_DOD, Int15…..etc.

Table 2. MXM Output Device Structure

Field Definition	Description
Descriptor[03:00]	Descriptor Type 0x00. Output Device Structure
Device Type[07:04]	Device Type 0x00 – Analog CRT 0x01 – Analog TV/HDTV 0x02 –TMDS or HDMI 0x03 - LVDS 0x04 ~ 0x05 - Reserved for future use 0x06 – DisplayPort 0x07 ~ 0x0E – Reserved for future use
DDC/Aux Port [11:08]	DDC or Aux Port connection 0x00 – DDCA 0x01 – DDCB 0x02 – DDCC 0x03 ~ 0x07 – Reserved for future use 0x08 – Aux0[1] 0x09 ~ 0x0E – Reserved for future use 0x0F – Not applicable
Connector Type [16:12]	Connector type 0x00 – VGA 0x01 – LVDS 0x02 – HDMI 0x03 – DVI-D 0x04 – DVI-I Analog port 0x05 – DVI-I Digital port 0x06 - DisplayPort external[2] connector 0x07 – DisplayPort internal connector 0x08 – Composite connector on TV_CVBS 0x09 – Composite connector on TV_Y 0x0A – S-video connector on TV_C and TV_Y 0x0B – HDTV connector on HDTV_Y, HDTV_Pr, HDTV_Pb 0x0C – D-connector 0x0D ~ 0x1E – Reserved for future use 0x1F – Not applicable
Connector Location [18:17]	The display output is located: 0x00 – Internal connection, not a user accessible connector 0x01 – A connector integrated in the chassis 0x02 – A connector on a docking station 0x03 – A connector (internal or external) integrated in the chassis which is not available when docked 0x04 – Reserved for future use
Digital Connection [22:19]	Digital signal MXM pin connection 0x00 – Reserved for future use 0x01 – Single-link DVI_A 0x02 – Single-link DVI_B 0x03 – Single-link DVI_C

[1] Only Aux0 is usable with the DisplayPort Link0. The Aux pins are shared for both DisplayPort Aux as well as Legacy DDC, hence it is not necessary to separately enumerate DDC pins.
[2] Refer to the DisplayPort Specification. The DisplayPort Internal connectors are typically used for chassis-internal panels, while the DisplayPort External connector is typically for user accessible external displays.

Field Definition	Description
	0x04 – Dual-link DVI_A + DVI_B 0x05 – Dual-link DVI_C (a single/dual-link capable port) 0x06 – LVDS, single-link 18-bit 0x07 – LVDS, dual-link default: 18-bit[3] 0x08 – LVDS, single-link default: 24-bit (see footnote) 0x09 – LVDS, dual-link default: 24-bit (see footnote) 0x0A – DisplayPort[4] Link0 0x0B ~ 0x0E – Reserved for future use 0x0F – Not applicable
TV Format [27:23]	Default format of TV output for Analog TV/HDTV 0x00 = NTSC_M (US) 0x01 = NTSC_J (Japan) 0x02 = PAL_M (Brazilian format) 0x03 = PAL_BDGHI 0x04 = PAL_N (Paraguay and Uruguay format) 0x05 = PAL_NC (Argentina format) 0x06 = Reserved for future use 0x07 = Reserved for future use 0x08 = HD576i 0x09 = HD480i 0x0A = HD480p 0x0B = HD576p 0x0C = HD720p 0x0D = HD1080i 0x0E = HD1080p 0x0F = Not specified. Determined at run time 0x10 ~ 0x1E = Reserved for future use 0x1F = Not applicable
Digital Audio Connection [24:23]	Audio for HDMI and DisplayPort 0x00 – Audio over SPDIF connection 0x01 – High definition audio connection 0x02 – Audio over PCIe bus 0x03 – No audio connection, or not applicable
Digital Drive Strength [25]	Drive strength when using digital connection 0x00 – Use higher drive strength for long signal runs 0x01 – Default or not applicable
Digital Reserved [27:26]	Reserved for future digital connection use 0x03 – Reserved for future use
GPIO for Output select [32:28]	Optional, set to 0x1F if unused. Specifies the logical GPIO used to select Device Output operation
Polarity for GPIO Output Select [33]	Specifies the polarity of the GPIO required to select the Device Output. Note: "Logical" values imply the asserted state as set by software 0 – A logical '0' on the GPIO selects the Output 1 – A logical '1' on the GPIO selects the Output
System Output Method [34]	System Methods (Int15 and ACPI DSS) to select display output 0 – Use GPIO's to select Output as per Bits [32:28]

[3] The default matches the system planar electrical design. However, the actual panel attached at assembly time may differ. For example, a single-link 18-bit panel may be connected into a socket accepting both single and dual or 18 and 24-bit panels. To override the output device connection default, either the panel or the system methods must export a Panel Digital Extension (DI-EXT) EDID block.
[4] DisplayPort outputs are MXM v 2.1 specific using previously reserved pins.

Field Definition	Description
	1 – Int15 and /or ACPI methods can select Output
GPIO for DDC select [39:35]	Optional, set to 0x1F if unused, Specifies the logical GPIO used to select DDC for device Note that a logical '1' on the indicated GPIO will steer a DDC mux to select this Output Device's DDC lanes.
System DDC Method [40]	System Methods (Int15 and ACPI MXMX) to select DDC 0 – Use GPIO's to select DDC as per Bits [39:35] 1 – Int15 and/or ACPI methods can select DDC
GPIO for Device Detection [45:41]	Optional, set to 0x1F if unused. Specifies the logical GPIO used to select Device Detection
Polarity for GPIO Device Detection [46]	Specifies the polarity of the GPIO which indicates device presence. Note: "Logical" values imply the asserted state as set by software 0 – A logical '0' on the GPIO indicates device is present 1 – A logical '1' on the GPIO indicates device is present
System Hot Plug Notify [47]	The display can cause ACPI Hot Plug Notification 0 – No notification 1 – Uses ACPI Notify for Display Plug/Unplug event

MXM System Cooling Capability Structure

The MXM System Cooling Capability Structure (Table 3) defines the thermal power dissipation capability of the MXM thermal solution contained in the system. This structure is required for all MXM systems.

The **DESCRIPTOR** field defines what kind of structure information is being stored. A descriptor of 0x01 means this is a thermal design power structure. Each cooling capability entry is a 32-bit field.

Table 3. MXM System Cooling Capability Structure

Field Definition	Description
Descriptor [03:00]	Descriptor Type 0x01. System Cooling Capability Structure
Type [07:04]	Type of cooling capability information 0x00 – Maximum cooling capability available through the required thermal transfer area
Value [17:08]	Power rating of this type of device. Value is in 100 milliWatts (100 mW) Example, a value of 0x78 (120) is 12.0 watts and a value of 0x145(325) is 32.5 watts.
Reserved [31:18]	

MXM Thermal Structure

By default the MXM v 2.1 module hardware and software will regulate its own temperature by varying the performance of the GPU. The optional structure allows the system to provide additional system requirements, for example, if the system thermal limits are lower than the MXM adapters normal temperature limits.

If the system has additional requirements for the module (for example, a lower maximum temperature than the default), the MXM thermal structure (Table 4) specifies this information in GPU junction temperature. Multiple thermal structures may be included (one per supported type).

The **DESCRIPTOR** field defines what kind of structure information is being stored. A descriptor of 0x02 means this is a thermal structure. Each thermal entry is a 32-bit field.

Table 4. MXM Thermal Structure

Field Definition	Description
Descriptor [03:00]	Descriptor Type 0x02. Thermal Structure
Type [07:04]	Type of thermal information 0x00 – Maximum temperature 0x01 – Temperature to assert the MXM THERM# signal 0x02 ~ 0x0F – Reserved for future use
Value [17:08]	Temperature in degrees Celsius
Scale [19:18]	Specifies the scale used for the temperature value. Range of values are 00b = 1.0x, 01b = 0.1x, 10b = 0.01x and 11b = 0.001x. The default value is 00b
Reserved [31:20]	

MXM Input Power Structure

The MXM Input Power Structure (Table 5) defines the maximum continuous available input power provided by the system for the **PWR_SRC** input power rail. At least one structure is required for all MXM systems. An additional MXM input power structure is required for all platform supported **AC/BATT#** levels. If only one structure is present it applies globally.

The **DESCRIPTOR** field defines what kind of structure information is being stored. A descriptor of 0x03 means this is an MXM input power structure. Each input power entry is a 32-bit field.

The **TYPE** field defines the power source level. There will be a separate input power structure for each available power source type or capability level. The **TYPE** field is 4 bits wide.

The **VALUE** field in combination with the **SCALE** field defines the upper limit on power supplied by the input power type based on the system and module connector capabilities. Power (in Watts) is calculated by multiplying the value in the **VALUE** field by the value in the **SCALE** field.

The **SCALE** field defines multiplication factor for the **VALUE** field. Range is from Watts to milliWatts. The definition of this field is as follows:

00b = 1.0x, 01b = 0.1x, 10b = 0.01x, 11b = 0.001x

Table 5. MXM Input Power Structure

Field Definition	Description
Descriptor [03:00]	Descriptor Type 0x03. Input Power Structure
Type [07:04]	Input Power Level 0x00 – AC/BATT# = 0 (example: battery power) 0x01 – AC/BATT# = 1 (example: AC power) 0x02 ~ 0x07 – Reserved for future hardware events 0x08 ~ 0x0F – Reserved for software power events 1 ~ 8 (see ACPI Notification)
Value [17:08]	Value of input power for a 4 Amp connector limit. Power (in Watts) calculated by multiplying the value in this field by the value in the Scale field. This field is required in all cases.
Value [27:18]	Value of input power for a 16 Amp connector limit. Power (in Watts) calculated by multiplying the value in this field by the value in the Scale field. This field must be 0 if a 16 Amp connector capability is not present in the platform.
Scale [29:28]	Specifies the scale used for the input power value. Range of values are - 00b = 1.0x, - 01b = 0.1x, - 10b = 0.01x and - 11b = 0.001x. The default value is 00b
Reserved [31:30]	

MXM GPIO Device Structure

This structure applies when an external I/O extender device is included in the system. It is mandatory that the external GPIO device is on DDCC.

The MXM GPIO Device Structure (Table 6) is used to define which MXM GPIO pins are attached to an external device.

This structure consists of a header structure with the **DESCRIPTOR, TYPE**, and other info. This is followed by a series of GPIO pin entries which enumerate the function and usage of all the MXM GPIO pins used.

The Logical pin assignment used in the Output Device Structure matches the Logical pin number declared in the GPIO Pin Entry Structure.

The physical pin assignment of the GPIO pins within a GPIO Expander is arranged beginning with 0 for the first GPIO pin structure entry after a GPIO device structure, and incrementing sequentially for each GPIO pin entry thereafter, until the next GPIO device structure.

The **DESCRIPTOR** field defines what kind of structure information is being stored. A descriptor of 0x04 means this is a GPIO device structure. Each GPIO device entry is a 32-bit field, while each GPIO pin is a 16-bit field.

Number GPIO pin entries field defines the number of GPIO pin entry structures which are defined.

Table 6. MXM GPIO Device Structure

Field Definition	Description
Descriptor [03:00]	Descriptor Type 0x04. MXM GPIO device structure
TYPE [11:04]	Type of GPIO device attached 0x00 – Philips PCA9555 0x01 – Philips PCA9536
I2C Address [19:12]	7-bit serial link communication address left justified to bits 7:1, with a 0 in bit 0.
Reserved [27:20]	
Number GPIO pin entries [31:28]	Number of GPIO pin entries.

Each GPIO pin entry has the following 16-bit structure (Table 7):

Table 7. GPIO Pin Entry Structure

Name	Description
Logical GPIO Number [03:00]	Logical GPIO number associated to this GPIO pin
Reserved [07:04]	
Function [15:08]	This identifies the function of the GPIO pin. 00 = Undefined 01 = Used for DDC Bus Expander, Output Mux or Display Detect (refer to the *MXM Output Device Structure* for Output, DDC Select, Detect and Polarity) 05 = Japanese D connector line 1 06 = Japanese D connector line 2 07 = Japanese D connector line 3 08 = Japanese D connector plug insertion detect 09 = Japanese D connector spare line 1 10 = Japanese D connector spare line 2 11 = Japanese D connector spare line 3 31 = LCD Self Test 32 = LCD Lamp Status 36 = HDTV Select: Allows steering the lines driven between SDTV (Logical '1') and HDTV (Logical '1') 37 = HDTV Alt-Detect: Allows detection of the connectors that are not steered by HDTV Select. That is, if HDTV Select is currently steered towards SDTV, then this GPIO would allow us detect the presence of the HDTV connection. Other function types are reserved.

Refer to the *MXM version 2.1 Graphics Module Thermal Electromechanical* specification for the GPIO function hardware definitions.

Each of these GPIO usage models implies an electrical signaling requirement, which can be accomplished through careful programming of the GPIO expander.

Table 8. GPIO Pin Usage Methods

Usage Model	Type	Signaling
DDC Mux (combine GPIO Type & Output Display Struct)	Output	Open-Drain
Output Mux (combine GPIO Type & Output Display Struct)	Output	Push-Pull
Display Detect (combine GPIO Type & Output Display Struct)	Input	N/A
D connector line	Output	3-State
D connector plug insertion detect	Input	N/A
LCD Self Test	Output	Push-Pull
LCD Lamp Status	Output	Push-Pull
HDTV Select	Output	Push-Pull
HDTV Alt-Detect	Input	N/A

The various signaling types are accomplished in the current GPIO Expanders using a combination of Output and Input mode:

Push-Pull : Normal GPIO Output mode

Logical '0' = GPIO in Output Mode & output set to '0'

Logical '1' = GPIO in Output Mode & output set to '1'

3-State : Supports three output states using Output and Input Mode

Logical '00' = GPIO in Output Mode & output set to '0'

Logical '01' = GPIO in Output Mode & output set to '1'

Other = GPIO Input Mode for Hi-Z

Open-Drain : Set output to Low and toggle Input/Output Mode

Logical '0' = GPIO in Output Mode & output set to '0'

Logical '1' = GPIO in Input Mode

Refer to the *MXM version 2.1 Graphics Module Thermal Electromechanical* specification for more details.

MXM Vendor Specific Structure

This is an optional GPU vendor specific structure (VSS). The contents of the structure are defined by the vendor, where each structure is tagged to indicate to which vendor it applies. There may be multiple structures, for example, with one VSS per supported vendor. Contact the GPU vendor for any additional details on VSS requirements or contents.

The **DESCRIPTOR** field defines what kind of structure information is being stored. A descriptor of 0x05 means this is a vendor specific structure. Each vendor structure entry is a 64-bit field.

The **TYPE** field indicates the vendor using the 16-bit Vendor Identifier as defined by the Plug & Play specification.

Table 9. MXM Vendor Specific Structure

Field Definition	Description
Descriptor [03:00]	Descriptor Type 0x05. GPU vendor specific structure
Type [19:04]	VID – Plug & Play. GPU vendor identifier
Reserved [63:20]	GPU vendor specific contents.

MXM Backlight Control System

This is an optional field that describes the settings for the LCD panel integrated into the chassis. If no PWM or I2C based LCD inverter is supported then the field is not required.

The **DESCRIPTOR** field defines what kind of structure information is being stored. A descriptor of 0x06 means this is a backlight PWM structure. Each backlight entry is a 64-bit field.

The **TYPE** field indicates the method of backlight control supported in the system.

Table 10. MXM Backlight Control Structure

Field Definition	Description
Descriptor [03:00]	Descriptor Type 0x06. Backlight control structure
Type [07:04]	Backlight Inverter Control Type 0 – PWM 1~ Reserved for future use
Max Duty Cycle [23:08]	The maximum duty cycle for PWM Inverter in 1/10 % representing maximum brightness
Min Duty Cycle [39:24]	The minimum duty cycle for PWM Inverter in 1/10 % representing the lowest brightness setting, not including the "off" state.
Duty Cycle Freq [57:40]	PWM Base Frequency in Hz
Reserved [63:58]	Reserved for future use. Must be zero.

MXM Checksum Byte

The MXM checksum byte is the two's complement of the 8-bit sum of the entire MXM v 2.1 structure (including header) placed at the last byte in the MXM table.

Using a Serial ROM to Access the MXM v 2.1 Structure

Apart from the system methods (Int15 and ACPI) the MXM v 2.1 structure can be accessed from a serial ROM which is placed on the motherboard. Communication is through serial link DDCC and the ROM device is assumed to be at address ACh/ADh. The MXM v 2.1 structure is read out of the ROM by the VBIOS during POST, as a series of sequential bytes starting at offset 0 in the ROM.

The use of a serial ROM on the motherboard is optional. The driver and VBIOS will attempt to locate and use the serial ROM before locating the system methods. If both the ROM and system methods are present the structure information will be used from the ROM. However, the other system methods may still be used. For example, the output and DDC lane steering will be used if supported.

In systems in which more than one MXM v 2.1 module may be supported, the serial ROM must not be used. Implementation of the INT 15h and ACPI methods is required in this case as this provides a means to directly distinguish the data for different modules.

Accessing MXM ROM via WMI

As the ROM is provided by the system to the module, and connected on a per-module basis, accessing all possible ROM through the primary VGA display device is not viable. For example, if the secondary device is an MXM card from a different vendor, or does not have VBIOS enabled.

If a ROM is present the module can support returning the ROM data to user-mode application on a per-instance basis, using the WMI format:

```
// WMI MOF Declaration
[
    WMI,
    Dynamic,
    Provider("WMIProv"),
    Locale("MS\\0x409"),
    GUID("{{FEC3A638-6A9F-43f7-BD5A-E1BA4D011E84}}")
]

class MXM20ROMdata
{
    [key, read] String InstanceName;
    [read] Boolean Active;

    [read, WmiDataId(1), Description("MXM 2.1 sizeof ROM")
    ] uint32 RomSize;

    [read, WmiDataId(2), Description("MXM 2.1 ROM bytes")
    ] uint8 RomBytes[];
};
```

See section *Accessing MXM ACPI Methods via WMI* for more details on WMI support in MXM.

MXM v 2.1 INT 15H System BIOS Callbacks

A set of SBIOS callback functions has been defined in order to allow the communication of system information between the VBIOS and the SBIOS.

If the SBIOS does not support any of the functions described, it should return from the callback with something other than 005Fh in AX.

Primary and Secondary Adapters

When using SBIOS callbacks it may be important to specify which adapter is being referenced in a multi-MXM adapter system.

- In a single MXM adapter system the MXM is always the primary.
- When a non-MXM graphics subsystem and a single MXM adapter are present, the primary MXM adapter is implicitly the only MXM adapter present.
- When multiple MXM adapters are present the primary MXM adapter shall be the first VGA enabled MXM adapter.
- If a non-MXM graphics subsystem is present together with more than one MXM adapter and one of the MXM adapters is a VGA enabled adapter then that shall be the primary MXM. If none of the MXM adapters are VGA enabled then the primary MXM adapter shall be the first MXM adapter enumerated based on the PCI bus/slot number ordering.

Function 0 – Return Specification Support Level

This is a required function that allows the VBIOS to get information from the SBIOS about the level of the MXM software specification that the system supports, and the support information for individual functions.

Entry:

AX = 5F80h

BL = 00h

BH = Adapter Index

0 = Primary MXM adapter (default)

1 = Secondary MXM adapter

CL = Revision of the MXM software specification that is supported by the MXM module

Format is binary coded decimal, for example:

```
10h = 1.0, 20h = 2.0, 21h = 2.1, etc.
```

Return:

AX = 005Fh to indicate that the system bios supports this function

BL = Revision of the MXM software specification that is supported by the system

Format is binary coded decimal, for example:

```
10h = 1.0, 20h = 2.0, 21h = 2.1, etc.
```

CX = MXM functions supported

Bit 0 = 1

Bit 1 = 1 if Function 1 is supported, 0 if not supported

Bit 2 = 1 if Function 2 is supported, 0 if not supported

Bit 3 = 1 if Function 3 is supported, 0 if not supported

Function 1 – Return a Pointer to the MXM Structure

This is a required function that will return a pointer to the MXM structure, which is stored in the SBIOS ROM area or some other memory location which is accessible in real mode during video POST.

Entry:

AX = 5F80h

BL = 01h

CL = Revision of the MXM software specification that is supported by the MXM module

Return:

AX = 005Fh to indicate that the system bios supports this function

ES:DI = Pointer to the MXM structure in real mode memory (< 1MB)

Function 2 – Return a Pointer to the EDID Structure for the Internal Panel

This is a required function for systems containing an internal flat panel without an EDID on DDC/Aux. This function allows the VBIOS to receive a pointer to the EDID structure that should be used for the internal flat panel in the system. This is required in cases where the panel being used does not have an EDID structure which can be read through DDC/Aux lines. This structure resides in the SBIOS ROM area or in another memory location that is accessible in real mode during video POST.

Entry:

AX = 5F80h

BL = 02h

BH = Adapter Index

0 = Primary MXM adapter (default)

1 = Secondary MXM adapter

Return:

AX = 005Fh	to indicate that the system bios supports this function
BL =	EDID structures returned
	00 = 128byte EDID 1.3 followed by a 128byte DI-EXT block
	02 = 128byte EDID 1.3 structure only

ES:DI = Pointer to the EDID structure in real mode memory (< 1MB)

The EDID structure shall comply with VESA E-EDID multi-block format. The first block of 128 bytes shall be a VESA 1.3 EDID. A digital extension in VESA DI-EXT format may optionally also be included. The VBIOS will attempt to read the EDID using this INT 15h callback first, then attempt to read the panel EDID via DDC, only if this function fails. If present, EDID information shall override MXM structure information. For example, as relates to link width or pixel depth.

Function 3 – Select Output Device

This is an optional function, which is only required if a multiplexer is used for controlling DDC or display outputs. The function is used when performing a display switch to an output device that is controlled by a multiplexer connected to an external GPIO. In order to allow the SBIOS to properly set up selection of the DDC port and/or the data output pins, the VBIOS will call this function before attempting to access the DDC port for the device or to set up and enable output to the device.

When selecting multiplexed DDC lanes, the VBIOS will call to acquire, and call again to release when the channel is no longer needed. If DDC lanes are shared between displays the SBIOS is responsible for creating a Mutex to co-ordinate between VBIOS (thru Int15h) and Driver (thru MXMX) access. If the VBIOS has acquired the mutex first then a simultaneous attempt thru MXMX should fail until the mutex is released, and vice-versa VBIOS should fail if the channel was already acquired through MXMX.

Entry:

AX = 5F80h

BL = 03h

BH = Adapter Index

0 = Primary MXM adapter (default)

1 = Secondary MXM adapter

CL = Selection

0 – Acquire shared Display DDC

1 – Display Output

3 – Both

4 – Release shared Display DDC

CH = Device index (range 0 – 7) according to the order in which the MXM Output Device Structure for this device appears in the MXM Data Structure

Exit:

AX = 005Fh	to indicate that the SBIOS supports this function
BL	For CL=0 the SBIOS shall return a 0h if the mutex was successfully acquired. When non-zero the mutex was not aquired

Function 4 – Boot Message

This is an optional function which returns a pointer to a sign-on message which VBIOS will display during POST, or indicate the VBIOS should not display any boot message. No delay is specifiable, it is up to the OEM SBIOS to control execution, and thereby the duration the screen retains the message.

Entry:

AX = 5F80h

BL = 04h

Return:

AX = 005Fh	To indicate that the system bios supports this function
BX = Mode	If zero then the Pointer is a (zero-terminated) Sign-On Text String.
ES:DI = Pointer	String/Image in real mode memory (< 1MB). A zero length string indicates the normal sign-on message should be suppressed e.g. to support a graphical splash screen.

MXM v 2.1 ACPI Methods

Where supported, methods within the ACPI namespace of the graphics adapter provide access to platform specific MXM functionality known by the SBIOS. Refer to implementation specific documentation on ACPI video extensions (*Advanced Configuration and Power Interface Specification* Revision 3.0a) for additional details.

Figure 3 is an example of the altered and new methods in the namespace (affected methods are bold type).

```
SB
|- PCI
    |- WMI1             // WMI Device
        |- WMMX         // WMI Method wrapper
    |- VGA              // Define the VGA controller in the namespace
        |- _DOS         // Method to control display output switching
        |- _DOD         // Method to retrieve info on child output devices
        |- _ROM         // Method to retrieve the ROM image for this device
        |- MXMI         // Method for probing MXM Support
        |- MXMS         // Method for passing the MXM Structure
        |- CRT          // Child device CRT
            |- _ADR     // Hardware ID for this device
            |- _DCS     // Get current hardware status
            |- _DGS     // Query desired hardware active \ inactive state
            |- _DSS     // Set hardware active \ inactive state
            |- MXMX     // Method for selecting display data channel
        |- HDMI         // Child device HDMI
            |- _ADR     // Hardware ID for this device
            |- _DCS     // Get current hardware status
            |- _DGS     // Query desired hardware active \ inactive state
            |- _DSS     // Set hardware active \ inactive state
            |- MXMX     // Method for selecting display data channel
        |- LCD          // Child device LCD
            |- _ADR     // Hardware ID for this device
            |- _DDC     // Get EDID information from the monitor device
            |- _DCS     // Get current hardware status
            |- _DGS     // Query desired hardware active \ inactive state
            |- _DSS     // Set hardware active \ inactive state
            |- _BCL     // Brightness control levels
            |- _BCM     // Brightness control method
        |- TV           // Child Device TV
            |- _ADR     // Hardware ID for this device
            |- _DDC     // Get EDID information from the monitor device
            |- _DCS     // Get current hardware status
            |- _DGS     // Query desired hardware active \ inactive state
            |- _DSS     // Set hardware active \ inactive state
```

Figure 3. Namespace with MXM Methods Example

ACPI Notification

An optional notification event may be used to indicate a configuration change, indicating a re-evaluation of either MXMI or MXMS system methods is required. Notify codes 0xD1 ~0xD8 are reserved for indicating input power events 1 ~ 8.

Note: The Output and GPIO device structures cannot be changed in such an event.

Display Hot Plug Notification is signaled through Notify (VGA, 0x81).

Returning the EDID Structure for the LVDS Panel via ACPI

When using ACPI video extensions, the EDID for an integrated panel without an actual DDC connection, is available through the _DDC method of the display. For example, SB.PCI.VGA.LCD._DDC(0) using the namespace example above. Multiple block E-EDID's are possible by using an EDID 1.3 structure together with extensions such as the DI-EXT block.

Retrieving the Backlight Control Settings for the LVDS Panel via ACPI

When using ACPI video extensions, the _BCL, _BCM, and _BQC methods shall be provided allowing the control necessary for backlight operation.

Selecting the Display Output via ACPI

If the MXM Output Device structure bit 34 indicates system methods are used for output switching then all necessary GPIO switching shall be performed inside _DSS in order to select the targeted display output.

However, when selecting a multiplexed display output data channel (DDC or Aux) use the MXMX method within the Display Device.

Accessing MXM ACPI methods via WMI

In order to allow access to the new MXM data from user mode and kernel software across Windows*NT O/S's the following headers shall be included in the graphics adapter namespace and any other devices that contain MXM methods in order to facilitate access via WMI. The GUID for WMI MXMX methods is:

{F6CB5C3C-9CAE-4EBD-B577-931EA32A2CC0}

The WMI GUIID for Notify event 0xD1

{F28A9357-CF4B-4A1A-8893-BB1F58EEA1AF}

For more details on using WMI to access ACPI methods refer to:
http://www.microsoft.com/whdc/system/pnppwr/wmi/wmi-acpi.mspx

```
Device(WMI1)   // placed within PCI Bus scope
{
    Name(_HID, "*pnp0c14") // pnp0c14 is the ID assigned to WMI mapper
    Name(_UID, "MXM2")     // use a unique UID for each instance

    // Description of data and events supported
    Name(_WDG, Buffer() {
        // Methods GUID {F6CB5C3C-9CAE-4ebd-B577-931EA32A2CC0}
        0x3C, 0x5C, 0xCB, 0xF6, 0xAE, 0x9C, 0xbd, 0x4e, 0xB5, 0x77, 0x93,
        0x1E, 0xA3, 0x2A, 0x2C, 0xC0,
        0x4D, 0x58,     // Object ID "MX" = method "WMMX"
        1,              // Instance Count
        0x02,           // Flags (WMIACPI_REGFLAG_METHOD)

        // Notify GUID {F28A9357-CF4B-4a1a-8893-BB1F58EEA1AF}
        0x57, 0x93, 0x8A, 0xF2, 0x4B, 0xCF, 0x1A, 0x4A, 0x88, 0x93, 0xBB,
        0x1F, 0x58, 0xEE, 0xA1, 0xAF,
        0xD1, 0,        // Notification ID Notify(VGA, 0xD1)
        1,              // Instance Count
        0x08,            // Flags (WMIACPI_REGFLAG_EVENT)

        // MOF data {05901221-D566-11d1-B2F0-00A0C9062910}
        0x21, 0x12, 0x90, 0x05, 0x66, 0xd5, 0xd1, 0x11, 0xb2, 0xf0,
        0x00, 0xa0, 0xc9, 0x06, 0x29, 0x10,
        0x58, 0x4D,     // Object ID "XM"
        1,              // Instance Count = 1
        0x00            // Flags
    })

    // Method Execution
    // MXM Native Methods are called via WMMX Index
    // ONLY include the methods that you actually have !
    Method(WMMX, 3)
    {
        If (LGreaterEqual(SizeOf(Arg2), 4))
        {
            CreateDWordField(Arg2, 0, FUNC)
            CreateDWordField(Arg2, 4, ARGS)
            If (LEqual(FUNC, 0x494D584D))     // "MXMI"
            {
                Return(\_SB_.PCI0.VGA0.MXMI(ARGS))
            }
            ElseIf (LEqual(FUNC, 0x534D584D)) // "MXMS"
            {
                Return(\_SB_.PCI0.VGA0.MXMS(ARGS))
            }
```

```
        ElseIf (LEqual(FUNC, 0x584D584D)) // "MXMX"
        {
            // Only implement these for devices with MXMX methods
            If (LGreaterEqual(SizeOf(Arg2), 8))
            {
                CreateDWordField(Arg2, 8, SARG)
                // Where CRT0._ADR = 0x80000100
                If (LEqual(ARGS, 0x80000100)) {
                // Example Only ! Use the actual location of the device
                    Return(\_SB_.PCI0.VGA0.CRT0.MXMX(SARG))
                }
                // Where LCD0._ADR = 0x0110
                ElseIf (LEqual(ARGS, 0x0110)) {
                    Return(\_SB_.PCI0.VGA0.LCD0.MXMX(SARG))
                }
                // Where LCD0._ADR = 0x80000210
                ElseIf (LEqual(ARGS, 0x80000210)) {
                    Return(\_SB_.PCI0.VGA0.HDV0.MXMX(SARG))
                }
            }
        }
        ElseIf (LEqual(FUNC, 0x4454474D)) // “MGTD”
        {
            Return(MGTD(ARGS))
        }
        ElseIf (LEqual(FUNC, 0x4454534D)) // “MSTD”
        {
            Return(MSTD(ARGS))
        }
    }
    Return(0)
}

// This is compiled form of the associated MOF declaration
Name(WQXM, Buffer() {
0x46,0x4F,0x4D,0x42,0x01,0x00,0x00,0x00,0x8B,0x02,0x00,0x00,0x0C,0x08,0x00,0x00,
0x44,0x53,0x00,0x01,0x1A,0x7D,0xDA,0x54,0x18,0xD2,0x83,0x00,0x01,0x06,0x18,0x42,
0x10,0x05,0x10,0x8A,0xE6,0x80,0x42,0x04,0x92,0x43,0xA4,0x30,0x30,0x28,0x0B,0x20,
0x86,0x90,0x0B,0x26,0x26,0x40,0x04,0x84,0xBC,0x0A,0xB0,0x29,0xC0,0x24,0x88,0xFA,
0xF7,0x87,0x28,0x09,0x0E,0x25,0x04,0x42,0x12,0x05,0x98,0x17,0xA0,0x5B,0x80,0x61,
0x01,0xB6,0x05,0x98,0x16,0xE0,0x18,0x92,0x4A,0x03,0xA7,0x04,0x96,0x02,0x21,0xA1,
0x02,0x94,0x0B,0xF0,0x2D,0x40,0x3B,0xA2,0x24,0x0B,0xB0,0x0C,0x23,0x02,0x8F,0x82,
0xA1,0x71,0x68,0xEC,0x30,0x2C,0x13,0x4C,0x83,0x38,0x8C,0xB2,0x91,0x45,0x60,0xDC,
0x4E,0x05,0xC8,0x15,0x20,0x4C,0x80,0x78,0x54,0x61,0x34,0x07,0x45,0xE0,0x42,0x63,
0x64,0x40,0xC8,0xA3,0x00,0xAB,0xA3,0xD0,0xA4,0x12,0xD8,0xBD,0x00,0x8D,0x02,0xB4,
0x09,0x70,0x28,0x40,0xA1,0x00,0x6B,0x18,0x72,0x06,0x21,0x5B,0xD8,0xC2,0x68,0x50,
0x80,0x45,0x14,0x8D,0xE0,0x2C,0x2A,0x9E,0x93,0x50,0x02,0xDA,0x1B,0x82,0xF0,0x8C,
0xD9,0x18,0x9E,0x10,0x83,0x54,0x86,0x21,0x88,0xB8,0x11,0x8E,0xA5,0xFD,0x41,0x10,
0xF9,0xAB,0xD7,0xB8,0x1D,0x69,0x34,0xA8,0xB1,0x26,0x38,0x76,0x8F,0xE6,0x84,0x3B,
0x17,0x20,0x7D,0x6E,0x02,0x39,0xBA,0xD3,0xA8,0x73,0xD0,0x64,0x78,0x0C,0x2B,0xC1,
0x7F,0x80,0x4F,0x01,0x78,0xD7,0x80,0x9A,0xFE,0xC1,0x33,0x41,0x70,0xA8,0x21,0x7A,
0xD4,0xE1,0x4E,0xE0,0xBC,0x8E,0x84,0x41,0x1C,0xD1,0x71,0x63,0x67,0x75,0x32,0x07,
0x5D,0xAA,0x00,0xB3,0x07,0x00,0x0D,0x2E,0xC1,0x69,0x9F,0x49,0xE8,0xF7,0x80,0xF3,
0xE9,0x79,0x6C,0x6C,0x10,0xA8,0x91,0xF9,0xFF,0x0F,0xED,0x41,0x9E,0x56,0xCC,0x90,
0xCF,0x02,0x87,0xC5,0xC4,0x1E,0x19,0xE8,0x78,0xC0,0x7F,0x00,0x78,0x34,0x88,0xF0,
0x66,0xE0,0xF9,0x9A,0x60,0x50,0x08,0x39,0x19,0x0F,0x4A,0xCC,0xF9,0x80,0xCC,0x25,
0xC4,0x43,0xC0,0x31,0xC4,0x08,0x7A,0x46,0x45,0x23,0x6B,0x22,0x3E,0x03,0x78,0xDC,
0x96,0x05,0x42,0x09,0x0C,0xEC,0x73,0xC3,0x3B,0x84,0x61,0x71,0xA3,0x09,0xEC,0xF3,
0x85,0x05,0x0E,0x0A,0x05,0xEB,0xBB,0x42,0xCC,0xE7,0x81,0xE3,0x3C,0x60,0x0B,0x9F,
0x28,0x01,0x3E,0x24,0x8F,0x06,0xDE,0x20,0xE1,0x5B,0x3F,0x02,0x10,0xE0,0x27,0x06,
0x13,0x58,0x1E,0x30,0x7A,0x94,0xF6,0x2B,0x00,0x21,0xF8,0x8B,0xC5,0x53,0xC0,0xEB,
0x40,0x84,0x63,0x81,0x29,0x72,0x6C,0x68,0x78,0x7E,0x70,0x88,0x1E,0xF5,0x5C,0xC2,
0x1F,0x4D,0x94,0x53,0x38,0x1C,0x1F,0x39,0x8C,0x10,0xFE,0x49,0xE3,0xC9,0xC3,0x9A,
0xEF,0x00,0x9A,0xD2,0x5B,0xC0,0xFB,0x83,0x47,0x80,0x11,0x20,0xE1,0x68,0x82,0x89,
0x7C,0x3A,0x01,0xD5,0xFF,0xFF,0x74,0x02,0xB8,0xBA,0x01,0x14,0x37,0x6A,0x9D,0x49,
0x7C,0x2C,0xF1,0xAD,0xE4,0xBC,0x43,0xC5,0x7F,0x93,0x78,0x3A,0xF1,0x34,0x1E,0x4C,
0x42,0x44,0x89,0x18,0x21,0xA2,0xEF,0x27,0x46,0x08,0x15,0x31,0x6C,0xA4,0x37,0x80,
0xE7,0x13,0xE3,0x84,0x08,0xF4,0x74,0xC2,0x42,0x3E,0x34,0xA4,0xE1,0x74,0x02,0x50,
0xE0,0xFF,0x7F,0x3A,0x81,0x1F,0xF5,0x74,0x82,0x1E,0xAE,0x4F,0x19,0x18,0xE4,0x03,
0xF2,0xA9,0xC3,0xF7,0x1F,0x13,0xF8,0x78,0xC2,0x45,0x1D,0x4F,0x50,0xA7,0x07,0x1F,
0x4F,0xD8,0x19,0xE1,0x2C,0x1E,0x03,0x7C,0x3A,0xC1,0xDC,0x13,0x7C,0x3A,0x01,0xDB,
0x68,0x60,0x1C,0x4F,0xC0,0x77,0x74,0xC1,0x1D,0x4F,0xC0,0x30,0x18,0x18,0xE7,0x13,
```

```
0xE0,0x31,0x5E,0xDC,0x31,0xC0,0x43,0xE0,0x03,0x78,0xDC,0x38,0x3D,0x2B,0x9D,0x14,
0xF2,0x24,0xC2,0x07,0x85,0x39,0xB0,0xE0,0x14,0xDA,0xF4,0xA9,0xD1,0xA8,0x55,0x83,
0x32,0x35,0xCA,0x34,0xA8,0xD5,0xA7,0x52,0x63,0xC6,0xCE,0x19,0x0E,0xF8,0x10,0xD0,
0x89,0xC0,0xF2,0x9E,0x0D,0x02,0xB1,0x0C,0x0A,0x81,0x58,0xFA,0xAB,0x45,0x20,0x0E,
0x0E,0xA2,0xFF,0x3F,0x88,0x23,0xD2,0x0A,0xC4,0xFF,0x7F,0x7F
})

// ------------------------------------------------------------------
/* Associated MOF Declaration

[WMI, Dynamic, Provider("WMIProv"),
 Locale("MS\\0x409"),
 GUID("{F6CB5C3C-9CAE-4ebd-B577-931EA32A2CC0}")]
class MXM20Method
{
    [key, read]
     String InstanceName;
    [read] Boolean Active;

    [WmiMethodId(1),
     Description("MXM20Method")
    ] uint32 MXM20Method;
};

[WMI, Dynamic, Provider("WMIProv"),
 Locale("MS\\0x409"),
 GUID("{F28A9357-CF4B-4a1a-8893-BB1F58EEA1AF}")]
class MXM20EventCA : WMIEvent
{
    [key, read]
     String InstanceName;
    [read] Boolean Active;

    [WmiDataId(1),
     Description("MXM20EventCA")
    ] uint32 MXM20EventCA;
};

*/
```

MXMI – Return Specification Support Level

This a required method which returns information about the level of the MXM software specification that the system supports. Calling this method with an argument of "0" shall always return the default or highest level supported. If additional versions of the MXM software interface are supported, the method shall indicate the specific version in the return value when queried with that version value as an input argument.

Arguments

Arg0: Binary Coded Decimal of the MXM adapters supported interface specification version

Return Values

Binary Coded Decimal of the *Base Platform Interface Specification* version

Sample Code

```
Method (MXMI, 1) {
   If (LEqual(Arg0, 0x20)) {
      Return (0x20)  // MXM 2.0 Backwards compatible
   }
   Else {
      Return (0x21)  // Supports MXM Version 2.1
   }
}
```

MXMS – Return the MXM Structure

This is a required method which returns the size of the MXM structure, and the structure (which may be up to a maximum 4 Kbytes in size). If supported a system may provide different version MXM software structures depending on the caller requested version.

Note: A page-locked physically contiguous buffer must be provided to retrieve the structure.

Arguments

Arg0: A 32 bit value

Bits 7:0

Requested version level in BCD format. If zero, then the structure version indicated in the return of MXMI will be supplied.

Bits 31:8

Reserved. Must be zero.

Return Values

0, invalid parameter, else the MXM structure

Sample Code

```
Name(MXM2, Buffer() {...}) // An MXM 2.0 structure

Name(MX21, Buffer() {...}) // An MXM 2.1 structure

If (LEqual (Arg0, 0x20)) {
   Return(MXM2)
}
Else {
   Return(MX21)
}
```

MXMX – Select Display Data Channel

This method is required on platforms that mux DDC lanes but is not required on other platforms. Given the limited number of pins on the MXM connector, some designs may multiplex the display data channel signals, sharing them between different display outputs. When present under a display in the namespace, this method abstracts the platform-specific mechanisms necessary for a client to select the correct signal lines multiplexer.

Additionally, in such designs these data lines may be a shared resource, and a mutex may be internally used to arbitrate access. Callers shall check the return value from an acquire request before assuming control. If the caller fails to acquire control they should back-off and retry after a reasonable interval (for example: 1~10 mS for on 100 Kbps I2C). In all cases, callers shall use this method to release control immediately after multiplexed signal lines are no longer needed.

Arguments

Arg0: Acquire/Release control of multiplexed signal lines

0 – Acquire control

1 – Release control

Return Values

0, Not Acquired.

Non-Zero, Success

Sample Code

```
Method (MXMX, 1, Serialized) {
    Return (0x1)    // No mutex needed, always
                    // always returns success
}
```

Use of _DOD

In order to identify the available display outputs in a notebook, or docking station, in an interoperable manner, the new ACPI 3.0 format shall be used for enumerating display ID's. This requires all Display ID's have set Bit [31] and use Bits [15:0] to indicate the display type, except for ID 0x110 which is assumed to be the integrated LCD, and only necessary for backwards compatibility.

Note: The type, topology connection and meaning of the enumerated outputs shall be consistent with other interfaces. For example, what is enumerated in system interfaces such as _DOD shall match devices enumerated from the MXM output device structures.

Refer to the *ACPI Specification* 3.0 for a detailed description of the _DOD method and fields.

Table 11. MXM Specific Fields in _DOD

Bits	Definition	
15:0	Device ID. The device ID must match the ID's specified by Video Chip Vendors. They must also be unique under VGA namespace.	
	Bit 3:0	Display Index
	Bit 7:4	Display Port Attachment
	Bit 11:8	Display Type
	Bit 15:12	Vendor specific fields
16	BIOS can detect the device.	
17	Non-VGA output device whose power is related to the VGA device.	
20:18	For VGA multiple-head devices, this specifies head or pipe ID	
30:21	0	
31	1 – Uses the ACPI 3.0 bit-field definitions above including bits 15:12	

MGTD – MXM Get Thermal Data

This is an optional query method query which if present, may be used by the driver software to obtain $T_{threshold}$ and $T_{resolution}$ values from the SBIOS. This query is typically performed once at startup. For example, when the device driver software is loaded by Windows.

Arguments

Arg0: Passed but reserved for future use

Return Values

32-bit bit-field as defined in Table 12

Table 12. Get Thermal Data Argument Bit Definitions

Bits	Definition
15:0	$T_{threshold}$ Temperature threshold value in °C (16-bit signed integer), from –100
31:16	$T_{resolution}$ Temperature resolution value in °C (16-bit signed integer), from +5 to +300

Sample Code

```
Method (MGTD, 1) {
   Return(0x000000269)   // Hypothetical
}
```

MSTD – MXM Set Thermal Data

This set method is required if query thermal data method is provided by the driver software, when appropriate, to notify the system of T_{sensor}. This set method is called depending on the actual values of $T_{threshold}$ and $T_{resolution}$.

Arguments

Arg0: 32-bit bit-field as defined in Table 13

Return Values

None

Sample Code

```
Method (MSTD, 1) {
    If (LGreaterEqual (Arg0, 0x69)) {
    Notify(_TPT,0x80)
    }
   Return(0)
}
```

Table 13. Set Thermal Data Argument Bit Definitions

Bits	Definition
31:0	T_{sensor} Current temperature value in °C

MXM Thermal Control for System Designers

The MXM thermal control protocol provides a vendor-independent means of implementing thermal management for (but not limited to) MXM system designs. The three main components involved are:

- The thermal sensor and controlling hardware on the MXM module (sensor)
- The SBIOS
- Graphics driver and application software (MXM v 2.1 software)

The thermal sensor is controlled by the MXM v 2.1 software, which contains support for all thermal sensors that may be used on MXM v 2.1 modules and performs any additional calibration which may be required. The MXM v 2.1 software supplies temperature information to the SBIOS through a simple mechanism described in the following section. Sample code is also provided to simplify the implementation task for system designers.

For system utilities running under any operating system, an alternative approach is to query the graphics driver directly for temperature information. The API and capabilities for such an API may vary per graphics vendor.

MXM Thermal Control Protocol

The protocol uses the following definitions. All temperatures are measured in degrees Celsius (°C).

- $T_{threshold}$ Temperature threshold at or above which SBIOS must be notified of temperature.
- $T_{resolution}$ The amount of change in temperature from that of the previous notification at which the SBIOS must be notified again.
- T_{sensor} Temperature measured by the thermal sensor.

The protocol specifies the following operations. Support for these operations must be implemented by both SBIOS and the MXM v 2.1 software.

MGTD - MXM Get Thermal Data

A query operation performed by MXM v 2.1 software to obtain $T_{threshold}$ and $T_{resolution}$ from the SBIOS. This query is performed only once at startup, when MXM software is loaded by the operating system.

MSTD - MXM Set Thermal Data

A set operation performed by MXM v 2.1 software, when appropriate, to notify SBIOS of T_{sensor}. This set method is called according to actual $T_{threshold}$ and $T_{resolution}$ values, as described in this section.

SBIOS support for the protocol is described in the *MXM ACPI Methods* section.

Using the Thermal Control Protocol

For the sake of discussion, it is convenient to create these additional definitions:

- $T_{up\_threshold} = T_{threshold}$
- $T_{down\_threshold} = T_{threshold} - T_{resolution}$

The SBIOS will be notified of temperature every $T_{resolution}$ steps from when the sensor temperature exceeds $T_{up\_threshold}$, until it drops below $T_{down\_threshold}$.

For example, if $T_{threshold} = 70$ and $T_{resolution} = 10$, then the following occurs when the sensor temperature *increases* from $T_{sensor} = 50$:

1. While $T_{sensor} < 70 \equiv T_{up\_threshold}$, nothing happens (SBIOS is not notified).
2. When $T_{sensor} \geq 70 \equiv T_{up\_threshold}$, SBIOS is notified by MXM v 2.1 software when T_{sensor} reaches 70 °C, and every time the sensor temperature changes by 10 °C above 70 °C (for example, 70 °C, 80 °C, 90 °C, etc.).
3. For the same $T_{threshold}$ and $T_{resolution}$, the following occurs when the sensor temperature *decreases* from $T_{sensor} = 90$:
4. While $T_{sensor} \geq 70 \equiv T_{up\_threshold}$, SBIOS is notified by MXM software every time the sensor temperature changes by 10 °C above 70 °C (for example, 80 °C, 90 °C).
5. When T_{sensor} crosses 60 (Tsensor <= $T_{down\_threshold}$), SBIOS is notified once by MXM software.
6. After this, while $T_{sensor} < 70$, nothing happens (SBIOS is not notified).
7. The protocol can be used to control a system fan, for instance:

- While no notifications occur, the system fan remains off.
- The fan starts when notifications of $T_{sensor} \geq T_{up\_threshold}$ begin, and continues given these notifications.
- The fan stops with any notification of $T_{sensor} \leq T_{down\_threshold}$ (upon which notifications will end).

Notice

ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Information furnished is believed to be accurate and reliable. However, NVIDIA Corporation assumes no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. No license is granted by implication or otherwise under any patent or patent rights of NVIDIA Corporation. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied. NVIDIA Corporation products are not authorized for use as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

Trademarks

NVIDIA, TTP and MXM Graphics Module, are trademarks of NVIDIA Corporation in the United States and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

HDMI and High-Definition Multimedia Interface are trademarks or registered trademarks of HDMI Licensing LLC.

Copyright

MXM
GRAPHICS MODULE